
Amazon Aurora

User Guide for Aurora



Amazon Aurora: User Guide for Aurora

Copyright © Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon's trademarks and trade dress may not be used in connection with any product or service that is not Amazon's, in any manner that is likely to cause confusion among customers, or in any manner that disparages or discredits Amazon. All other trademarks not owned by Amazon are the property of their respective owners, who may or may not be affiliated with, connected to, or sponsored by Amazon.

Table of Contents

What is Aurora?	1
Aurora DB clusters	3
Aurora versions	5
Relational databases that are available on Aurora	5
Differences in version numbers between community databases and Aurora	5
Amazon Aurora major versions	6
Amazon Aurora minor versions	7
Amazon Aurora patch versions	7
Learning what's new in each Amazon Aurora version	7
Specifying the Amazon Aurora database version for your database cluster	7
Default Amazon Aurora versions	7
Automatic minor version upgrades	8
How long Amazon Aurora major versions remain available	8
How often Amazon Aurora minor versions are released	9
How long Amazon Aurora minor versions remain available	9
Long-term support for selected Amazon Aurora minor versions	9
Manually controlling if and when your database cluster is upgraded to new versions	9
Required Amazon Aurora upgrades	10
Testing your DB cluster with a new Aurora version before upgrading	10
Regions and Availability Zones	11
AWS Regions	11
Availability Zones	15
Local time zone for DB clusters	16
Supported Aurora features by Region and engine	19
Backtracking in Aurora	19
Aurora global databases	21
Aurora machine learning	23
Aurora parallel queries	26
Amazon RDS Proxy	27
Aurora Serverless v1	29
Data API for Aurora Serverless	31
Aurora connection management	32
Types of Aurora endpoints	33
Viewing endpoints	35
Using the cluster endpoint	35
Using the reader endpoint	35
Using custom endpoints	36
Creating a custom endpoint	38
Viewing custom endpoints	40
Editing a custom endpoint	45
Deleting a custom endpoint	47
End-to-end AWS CLI example for custom endpoints	48
Using the instance endpoints	52
Endpoints and high availability	52
DB instance classes	54
DB instance class types	54
Supported DB engines	54
Determining DB instance class support in AWS Regions	59
Hardware specifications	62
Aurora storage and reliability	64
Overview of Aurora storage	64
Cluster volume contents	65
How storage resizes	65
Data billing	65

Reliability	66
Aurora security	67
Using SSL with Aurora DB clusters	68
High availability for Amazon Aurora	68
High availability for Aurora data	68
High availability for Aurora DB instances	68
High availability across AWS Regions with Aurora global databases	69
Fault tolerance	69
Replication with Aurora	70
Aurora Replicas	70
Aurora MySQL	71
Aurora PostgreSQL	72
DB instance billing for Aurora	72
On-Demand DB instances	74
Reserved DB instances	75
Setting up your environment	84
Sign up for AWS	84
Create an IAM user	84
Determine requirements	86
Provide access to the DB cluster	87
Getting started	89
Creating an Aurora MySQL DB cluster and connecting to it	89
Create an Aurora MySQL DB cluster	89
Connect to an instance in a DB cluster	94
Delete the sample DB cluster, DB subnet group, and VPC	96
Creating an Aurora PostgreSQL DB cluster and connecting to it	96
Create an Aurora PostgreSQL DB cluster	97
Connect to an instance in an Aurora PostgreSQL DB cluster	101
Delete the sample DB cluster, DB subnet group, and VPC	102
Tutorial: Create a web server and an Amazon Aurora DB cluster	103
Create a DB cluster	104
Create a web server	109
Tutorials and sample code	121
Tutorials in this guide	121
Tutorials in other AWS guides	121
Tutorials and sample code in GitHub	122
Configuring your Aurora DB cluster	124
Creating a DB cluster	125
Prerequisites	125
Creating a DB cluster	126
Available settings	137
Creating resources with AWS CloudFormation	146
Aurora and AWS CloudFormation templates	146
Learn more about AWS CloudFormation	146
Using Aurora Serverless v1	147
Advantages of Aurora Serverless v1	147
Use cases for Aurora Serverless v1	148
Limitations of Aurora Serverless v1	148
Configuration requirements for Aurora Serverless v1	149
Using TLS/SSL with Aurora Serverless v1	150
How Aurora Serverless v1 works	151
Creating an Aurora Serverless v1 DB cluster	161
Restoring an Aurora Serverless v1 DB cluster	166
Modifying an Aurora Serverless v1 DB cluster	170
Scaling Aurora Serverless v1 DB cluster capacity manually	172
Viewing Aurora Serverless v1 DB clusters	174
Deleting an Aurora Serverless v1 DB cluster	175

Aurora Serverless v1 and Aurora database engine versions	177
Using the Data API	178
Logging Data API calls with AWS CloudTrail	202
Using the query editor	204
Using Aurora Serverless v2 (preview)	212
How Aurora Serverless v2 (preview) works	212
Limitations of Aurora Serverless v2 (preview)	215
Creating an Aurora Serverless v2 (preview) DB cluster	216
Creating a snapshot of an Aurora Serverless v2 (preview) DB cluster	219
Modifying an Aurora Serverless v2 (preview) DB cluster	220
Deleting an Aurora Serverless v2 (preview) DB cluster	222
Restoring an Aurora Serverless v2 (preview) DB cluster	223
Using Aurora global databases	225
Overview of Aurora global databases	225
Advantages of Amazon Aurora global databases	226
Limitations of Aurora global databases	226
Getting started with Aurora global databases	228
Managing an Aurora global database	249
Connecting to an Aurora global database	254
Using write forwarding in an Aurora global database	255
Using failover in an Aurora global database	266
Monitoring an Aurora global database	276
Using Aurora global databases with other AWS services	279
Upgrading an Amazon Aurora global database	280
Connecting to a DB cluster	281
Connecting to Aurora MySQL	281
Connecting to Aurora PostgreSQL	285
Troubleshooting connections	287
Using RDS Proxy	288
Supported engines and Region availability	288
Quotas and limitations	288
Planning where to use RDS Proxy	290
RDS Proxy concepts and terminology	290
Getting started with RDS Proxy	295
Managing an RDS Proxy	306
Working with RDS Proxy endpoints	315
Monitoring RDS Proxy with CloudWatch	324
Working with RDS Proxy events	329
RDS Proxy examples	330
Troubleshooting RDS Proxy	332
Using RDS Proxy with AWS CloudFormation	337
Working with parameter groups	339
DB cluster and DB instance parameters	341
Creating a DB parameter group	342
Creating a DB cluster parameter group	343
Associating a DB parameter group with a DB instance	345
Associating a DB cluster parameter group with a DB cluster	346
Modifying parameters in a DB parameter group	347
Modifying parameters in a DB cluster parameter group	349
Resetting parameters in a DB parameter group	351
Resetting parameters in a DB cluster parameter group	353
Copying a DB parameter group	354
Copying a DB cluster parameter group	356
Listing DB parameter groups	357
Listing DB cluster parameter groups	358
Viewing parameter values for a DB parameter group	359
Viewing parameter values for a DB cluster parameter group	360

Comparing parameter groups	362
Specifying DB parameters	362
Migrating data to a DB cluster	366
Aurora MySQL	366
Aurora PostgreSQL	366
Managing an Aurora DB cluster	367
Stopping and starting a cluster	368
Overview of stopping and starting a cluster	368
Limitations	368
Stopping a DB cluster	369
While a DB cluster is stopped	370
Starting a DB cluster	370
Modifying an Aurora DB cluster	372
Modifying the DB cluster by using the console, CLI, and API	372
Modify a DB instance in a DB cluster	373
Available settings	375
Non-applicable settings	390
Adding Aurora Replicas	392
Managing performance and scaling	396
Storage scaling	396
Instance scaling	400
Read scaling	400
Managing connections	400
Managing query execution plans	401
Cloning a volume for an Aurora DB cluster	402
Overview of Aurora cloning	402
Limitations of Aurora cloning	403
How Aurora cloning works	403
Creating an Aurora clone	406
Cross-account cloning	415
Integrating with AWS services	426
Aurora MySQL	426
Aurora PostgreSQL	426
Using Auto Scaling with Aurora replicas	427
Using machine learning with Aurora	442
Maintaining an Aurora DB cluster	443
Viewing pending maintenance	443
Applying updates	445
The maintenance window	446
Adjusting the maintenance window for a DB cluster	448
Automatic minor version upgrades for Aurora DB clusters	449
Choosing the frequency of Aurora MySQL maintenance updates	449
Rebooting an Aurora DB cluster or instance	451
Rebooting a DB instance within an Aurora cluster	451
Rebooting an Aurora cluster (Aurora PostgreSQL and Aurora MySQL before version 2.10)	452
Rebooting an Aurora MySQL cluster (version 2.10 and higher)	452
Checking uptime for Aurora clusters and instances	453
Examples of Aurora reboot operations	455
Deleting Aurora clusters and instances	467
Deleting an Aurora DB cluster	467
Deletion protection for Aurora clusters	471
Deleting a stopped Aurora cluster	472
Deleting Aurora MySQL clusters that are read replicas	472
The final snapshot when deleting a cluster	472
Deleting a DB instance from an Aurora DB cluster	472
Tagging RDS resources	474
Overview	474

Using tags for access control with IAM	475
Using tags to produce detailed billing reports	475
Adding, listing, and removing tags	476
Using the AWS Tag Editor	478
Copying tags to DB cluster snapshots	478
Tutorial: Use tags to specify which Aurora DB clusters to stop	479
Working with ARNs	482
Constructing an ARN	482
Getting an existing ARN	485
Aurora updates	489
Identifying your Amazon Aurora version	489
Backing up and restoring an Aurora DB cluster	490
Overview of backing up and restoring	491
Backups	491
Backup window	491
Restoring data	493
Backtrack	493
Backup storage	494
Creating a DB cluster snapshot	495
Determining whether the snapshot is available	496
Restoring from a DB cluster snapshot	497
Parameter groups	497
Security groups	497
Aurora considerations	497
Restoring from a snapshot	498
Copying a snapshot	500
Limitations	500
Snapshot retention	500
Copying shared snapshots	501
Handling encryption	501
Incremental snapshot copying	501
Cross-Region copying	501
Parameter groups	502
Copying a DB cluster snapshot	502
Sharing a snapshot	510
Sharing public snapshots	510
Sharing encrypted snapshots	511
Sharing a snapshot	513
Exporting snapshot data to Amazon S3	518
Limitations	519
Overview of exporting snapshot data	519
Setting up access to an S3 bucket	520
Using a cross-account KMS key	522
Exporting a snapshot to an S3 bucket	523
Monitoring snapshot exports	526
Canceling a snapshot export	527
Failure messages	528
Troubleshooting PostgreSQL permissions errors	529
File naming convention	529
Data conversion	530
Point-in-time recovery	537
Deleting a snapshot	539
Deleting a DB cluster snapshot	539
Monitoring metrics in an Aurora DB cluster	541
Overview of monitoring	542
Monitoring plan	542
Performance baseline	542

Performance guidelines	542
Monitoring tools	543
Viewing cluster status and recommendations	546
Viewing a DB cluster	547
Viewing DB cluster status	553
Viewing DB instance status in an Aurora cluster	555
Viewing Amazon Aurora recommendations	558
Viewing metrics in the Amazon RDS console	563
Monitoring Aurora with CloudWatch	566
Viewing CloudWatch metrics	568
Creating CloudWatch alarms	571
Monitoring DB load with Performance Insights	573
Overview of Performance Insights	573
Enabling and disabling Performance Insights	577
Enabling the Performance Schema for Aurora MySQL	580
Performance Insights policies	582
Analyzing metrics with the Performance Insights dashboard	585
Retrieving metrics with the Performance Insights API	607
Logging Performance Insights calls using AWS CloudTrail	621
Analyzing performance with DevOps Guru for RDS	623
Benefits of DevOps Guru for RDS	623
How DevOps Guru for RDS works	624
Setting up DevOps Guru for RDS	624
Monitoring the OS with Enhanced Monitoring	626
Overview of Enhanced Monitoring	626
Setting up and enabling Enhanced Monitoring	627
Viewing OS metrics in the RDS console	630
Viewing OS metrics using CloudWatch Logs	632
Aurora metrics reference	633
CloudWatch metrics for Aurora	633
CloudWatch dimensions for Aurora	649
Availability of Aurora metrics in the Amazon RDS console	649
CloudWatch metrics for Performance Insights	652
Counter metrics for Performance Insights	653
OS metrics in Enhanced Monitoring	660
Monitoring events, logs, and database activity streams	666
Viewing logs, events, and streams in the Amazon RDS console	666
Monitoring Aurora events	671
Overview of events for Aurora	671
Viewing Amazon RDS events	674
Using Amazon RDS event notification	675
Creating a rule that triggers on an Amazon Aurora event	692
Monitoring Aurora logs	695
Viewing and listing database log files	695
Downloading a database log file	696
Watching a database log file	697
Publishing to CloudWatch Logs	697
Reading log file contents using REST	698
MySQL database log files	700
PostgreSQL database log files	706
Monitoring Aurora API calls in CloudTrail	710
CloudTrail integration with Amazon Aurora	710
Amazon Aurora log file entries	710
Monitoring Aurora with Database Activity Streams	714
Overview	714
Aurora MySQL network prerequisites	717
Starting a database activity stream	718

Getting activity stream status	720
Stopping a database activity stream	720
Monitoring activity streams	721
Managing access to activity streams	743
Working with Aurora MySQL	746
Overview of Aurora MySQL	746
Amazon Aurora MySQL performance enhancements	746
Aurora MySQL and spatial data	747
Aurora MySQL version 3 compatible with MySQL 8.0	748
Aurora MySQL version 2 compatible with MySQL 5.7	773
Security with Aurora MySQL	774
Master user privileges with Aurora MySQL	775
Using SSL/TLS with Aurora MySQL DB clusters	775
Updating applications for new SSL/TLS certificates	778
Determining whether any applications are connecting to your Aurora MySQL DB cluster using SSL	778
Determining whether a client requires certificate verification to connect	779
Updating your application trust store	780
Example Java code for establishing SSL connections	781
Migrating data to Aurora MySQL	781
Migrating from an external MySQL database to Aurora MySQL	784
Migrating from a MySQL DB instance to Aurora MySQL	797
Managing Aurora MySQL	812
Managing performance and scaling for Amazon Aurora MySQL	812
Backtracking a DB cluster	816
Testing Amazon Aurora using fault injection queries	829
Altering tables in Amazon Aurora using fast DDL	832
Displaying volume status for an Aurora DB cluster	837
Tuning Aurora MySQL with wait events and thread states	837
Essential concepts for Aurora MySQL tuning	838
Tuning Aurora MySQL with wait events	840
Tuning Aurora MySQL with thread states	876
Parallel query for Aurora MySQL	881
Overview of parallel query	883
Planning for a parallel query cluster	885
Creating a parallel query cluster	886
Turning parallel query on and off	890
Upgrading a parallel query cluster	893
Performance tuning	894
Creating schema objects	895
Verifying parallel query usage	895
Monitoring	898
Parallel query and SQL constructs	901
Advanced Auditing with Aurora MySQL	914
Enabling Advanced Auditing	915
Viewing audit logs	917
Audit log details	917
Single-master replication with Aurora MySQL	918
Aurora replicas	918
Options	919
Performance	920
Zero-downtime restart (ZDR)	920
Monitoring	922
Replicating Amazon Aurora MySQL DB clusters across AWS Regions	922
Replication between Aurora and MySQL or between Aurora and another Aurora DB cluster (binary log replication)	932
Using GTID-based replication	954

Working with multi-master clusters	958
Overview of multi-master clusters	958
Creating a multi-master cluster	963
Managing multi-master clusters	969
Application considerations	972
Performance considerations	981
Approaches to multi-master clusters	983
Integrating Aurora MySQL with AWS services	984
Authorizing Aurora MySQL to access AWS services	985
Loading data from text files in Amazon S3	997
Saving data into text files in Amazon S3	1004
Invoking a Lambda function from Aurora MySQL	1010
Publishing Aurora MySQL logs to CloudWatch Logs	1017
Using machine learning with Aurora MySQL	1020
Aurora MySQL lab mode	1032
Aurora lab mode features	1033
Best practices with Amazon Aurora MySQL	1033
Determining which DB instance you are connected to	1034
Best practices for using AWS features with Aurora MySQL	1034
Best practices for Aurora MySQL performance and scaling	1036
Best practices for Aurora MySQL high availability	1040
Best practices for limiting certain MySQL features with Aurora MySQL	1041
Aurora MySQL reference	1042
Configuration parameters	1042
MySQL parameters that don't apply to Aurora MySQL	1061
MySQL status variables that don't apply to Aurora MySQL	1062
Aurora MySQL wait events	1063
Aurora MySQL thread states	1067
Aurora MySQL isolation levels	1070
Aurora MySQL hints	1074
Stored procedures	1076
Aurora MySQL updates	1082
Version Numbers and Special Versions	1083
Preparing for Aurora MySQL version 1 end of life	1086
Upgrading Amazon Aurora MySQL DB clusters	1088
Database engine updates for Amazon Aurora MySQL version 3	1108
Database engine updates for Amazon Aurora MySQL version 2	1108
Database engine updates for Amazon Aurora MySQL version 1	1196
Database engine updates for Aurora MySQL Serverless clusters	1247
MySQL bugs fixed by Aurora MySQL updates	1250
Security vulnerabilities fixed in Amazon Aurora MySQL	1271
Working with Aurora PostgreSQL	1275
Security with Aurora PostgreSQL	1276
Restricting password management	1277
Securing Aurora PostgreSQL data with SSL/TLS	1277
Updating applications for new SSL/TLS certificates	1280
Determining whether applications are connecting to Aurora PostgreSQL DB clusters using SSL	1280
Determining whether a client requires certificate verification in order to connect	1281
Updating your application trust store	1281
Using SSL/TLS connections for different types of applications	1282
Migrating data to Aurora PostgreSQL	1283
Migrating an RDS for PostgreSQL DB instance using a snapshot	1284
Migrating an RDS for PostgreSQL DB instance using an Aurora read replica	1288
Babelfish for Aurora PostgreSQL	1297
Babelfish architecture	1297
Using Babelfish to migrate to PostgreSQL	1301
Creating an Aurora PostgreSQL cluster with Babelfish	1303

Connecting to a DB cluster with Babelfish turned on	1310
Querying a database for object information	1318
Querying Babelfish to find Babelfish details	1319
Differences between Aurora PostgreSQL with Babelfish and SQL Server	1322
Using Aurora PostgreSQL extensions with Babelfish	1335
Managing Babelfish error handling	1340
Configuring a database for Babelfish	1345
Babelfish collation support	1349
Troubleshooting for Babelfish	1355
Turning off Babelfish	1357
Babelfish versions	1358
Managing Aurora PostgreSQL	1360
Scaling Aurora PostgreSQL DB instances	1360
Maximum connections	1360
Temporary storage limits	1362
Testing Amazon Aurora PostgreSQL by using fault injection queries	1364
Displaying volume status for an Aurora DB cluster	1367
Specifying the RAM disk for the stats_temp_directory	1368
Scheduling maintenance with the pg_cron extension	1370
Tuning with wait events for Aurora PostgreSQL	1376
Essential concepts for Aurora PostgreSQL tuning	1377
Aurora PostgreSQL wait events	1380
Client:ClientRead	1381
Client:ClientWrite	1384
CPU	1385
IO:BufFileRead and IO:BufFileWrite	1389
IO:DataFileRead	1395
IO:XactSync	1401
ipc:damrecordtxack	1403
Lock:advisory	1403
Lock:extend	1405
Lock:Relation	1407
Lock:transactionid	1410
Lock:tuple	1413
lwlock:buffer_content (BufferContent)	1415
LWLock:buffer_mapping	1417
LWLock:BufferIO	1418
LWLock:lock_manager	1420
Timeout:PgSleep	1423
Best practices with Aurora PostgreSQL	1423
Fast failover	1423
Troubleshooting storage issues	1431
Replication with Aurora PostgreSQL	1431
Aurora Replicas	1431
Monitoring replication	1432
Using logical replication	1432
Integrating Aurora PostgreSQL with AWS services	1437
Importing S3 data into Aurora PostgreSQL	1438
Overview of importing S3 data	1438
Setting up access to an Amazon S3 bucket	1439
Using the aws_s3.table_import_from_s3 function to import Amazon S3 data	1444
Function reference	1446
Exporting PostgreSQL data to Amazon S3	1450
Overview of exporting to S3	1450
Verify that your Aurora PostgreSQL version supports exports	1451
Specifying the Amazon S3 file path to export to	1451
Setting up access to an Amazon S3 bucket	1452

Exporting query data using the <code>aws_s3.query_export_to_s3</code> function	1455
Troubleshooting access to Amazon S3	1457
Function reference	1457
Managing query execution plans for Aurora PostgreSQL	1460
Enabling query plan management	1461
Upgrading query plan management	1462
Basics	1462
Best practices for query plan management	1465
Examining plans in the <code>dba_plans</code> view	1466
Capturing execution plans	1469
Using managed plans	1470
Maintaining execution plans	1473
Parameter reference for query plan management	1477
Function reference for query plan management	1480
Publishing Aurora PostgreSQL logs to CloudWatch Logs	1487
Publishing logs to Amazon CloudWatch	1487
Monitoring log events in Amazon CloudWatch	1490
Analyze Aurora PostgreSQL logs using CloudWatch Logs Insights	1490
Using machine learning with Aurora PostgreSQL	1494
Prerequisites	1495
Enabling Aurora machine learning	1495
Using Amazon Comprehend for natural language processing	1497
Exporting data to Amazon S3 for SageMaker model training	1499
Using SageMaker to run your own ML models	1499
Best practices with Aurora machine learning	1502
Monitoring Aurora machine learning	1506
Function reference	1507
Manually setting up IAM roles using the AWS CLI	1509
Fast recovery after failover	1513
Configuring cluster cache management	1513
Monitoring the buffer cache	1516
Invoking a Lambda function from Aurora PostgreSQL	1517
Step 1: Configure outbound connections	1518
Step 2: Configure IAM for your cluster and Lambda	1518
Step 3: Install the extension	1519
Step 4: Use Lambda helper functions	1520
Step 5: Invoke a Lambda function	1521
Lambda function error messages	1524
Function reference	1524
Using <code>oracle_fdw</code> to access foreign data	1527
Turning on the <code>oracle_fdw</code> extension	1527
Example using a foreign server linked to an RDS for Oracle database	1527
Working with encryption in transit	1528
<code>pg_user_mapping</code> and <code>pg_user_mappings</code> permissions	1528
Managing partitions with the <code>pg_partman</code> extension	1530
Overview of the PostgreSQL <code>pg_partman</code> extension	1531
Enabling the <code>pg_partman</code> extension	1531
Configuring partitions using the <code>create_parent</code> function	1532
Configuring partition maintenance using the <code>run_maintenance_proc</code> function	1533
Using Kerberos authentication	1534
Availability	1534
Overview of Kerberos authentication	1535
Setting up	1536
Managing a DB cluster in a Domain	1545
Connecting with Kerberos authentication	1546
Aurora PostgreSQL reference	1547
Aurora PostgreSQL parameters	1547

Aurora PostgreSQL wait events	1570
Aurora PostgreSQL functions reference	1588
Aurora PostgreSQL updates	1597
Identifying versions of Amazon Aurora PostgreSQL	1598
Aurora PostgreSQL releases	1599
Extension versions for Aurora PostgreSQL	1668
Upgrading the PostgreSQL DB engine	1681
Using a long-term support (LTS) release	1690
Best practices with Aurora	1692
Basic operational guidelines for Amazon Aurora	1692
DB instance RAM recommendations	1692
Monitoring Amazon Aurora	1693
Working with DB parameter groups and DB cluster parameter groups	1693
Amazon Aurora best practices presentation video	1693
Performing an Aurora proof of concept	1694
Overview of an Aurora proof of concept	1694
1. Identify your objectives	1694
2. Understand your workload characteristics	1695
3. Practice with the console or CLI	1696
Practice with the console	1696
Practice with the AWS CLI	1696
4. Create your Aurora cluster	1697
5. Set up your schema	1698
6. Import your data	1698
7. Port your SQL code	1699
8. Specify configuration settings	1699
9. Connect to Aurora	1700
10. Run your workload	1701
11. Measure performance	1701
12. Exercise Aurora high availability	1703
13. What to do next	1704
Security	1706
Database authentication	1707
Password authentication	1708
IAM database authentication	1708
Kerberos authentication	1708
Data protection	1709
Data encryption	1709
Internetwork traffic privacy	1723
Identity and access management	1724
Audience	1724
Authenticating with identities	1724
Managing access using policies	1726
How Amazon Aurora works with IAM	1727
Identity-based policy examples	1730
Cross-service confused deputy prevention	1741
IAM database authentication	1743
Troubleshooting	1769
Logging and monitoring	1771
Compliance validation	1773
Resilience	1774
Backup and restore	1774
Replication	1774
Failover	1774
Infrastructure security	1776
Security groups	1776
Public accessibility	1776

VPC endpoints (AWS PrivateLink)	1777
Considerations	1777
Availability	1777
Creating an interface VPC endpoint	1778
Creating a VPC endpoint policy	1778
Security best practices	1779
Controlling access with security groups	1780
VPC security groups	1780
Security group scenario	1780
Creating a VPC security group	1781
Associating with a DB instance	1781
Associating with a DB cluster	1781
Master user account privileges	1782
Service-linked roles	1783
Service-linked role permissions for Amazon Aurora	1783
Using Amazon Aurora with Amazon VPC	1787
Working with a DB instance in a VPC	1787
Creating a VPC for Aurora	1793
Scenarios for accessing a DB instance in a VPC	1800
Tutorial: Create an Amazon VPC for use with a DB instance	1805
Quotas and constraints	1811
Quotas in Amazon Aurora	1811
Naming constraints in Amazon Aurora	1812
Amazon Aurora size limits	1813
Troubleshooting	1814
Can't connect to DB instance	1814
Testing the DB instance connection	1815
Troubleshooting connection authentication	1816
Security issues	1816
Error message "failed to retrieve account attributes, certain console functions may be impaired."	1816
Resetting the DB instance owner password	1816
DB instance outage or reboot	1816
Parameter changes not taking effect	1817
Aurora MySQL out of memory issues	1817
Aurora MySQL replication issues	1818
Diagnosing and resolving lag between read replicas	1818
Diagnosing and resolving a MySQL read replication failure	1819
Replication stopped error	1820
Amazon RDS API reference	1822
Using the Query API	1822
Query parameters	1822
Query request authentication	1822
Troubleshooting applications	1823
Retrieving errors	1823
Troubleshooting tips	1823
Document history	1824
AWS glossary	1862

What is Amazon Aurora?

Amazon Aurora (Aurora) is a fully managed relational database engine that's compatible with MySQL and PostgreSQL. You already know how MySQL and PostgreSQL combine the speed and reliability of high-end commercial databases with the simplicity and cost-effectiveness of open-source databases. The code, tools, and applications you use today with your existing MySQL and PostgreSQL databases can be used with Aurora. With some workloads, Aurora can deliver up to five times the throughput of MySQL and up to three times the throughput of PostgreSQL without requiring changes to most of your existing applications.

Aurora includes a high-performance storage subsystem. Its MySQL- and PostgreSQL-compatible database engines are customized to take advantage of that fast distributed storage. The underlying storage grows automatically as needed. An Aurora cluster volume can grow to a maximum size of 128 tebibytes (TiB). Aurora also automates and standardizes database clustering and replication, which are typically among the most challenging aspects of database configuration and administration.

Aurora is part of the managed database service Amazon Relational Database Service (Amazon RDS). Amazon RDS is a web service that makes it easier to set up, operate, and scale a relational database in the cloud. If you are not already familiar with Amazon RDS, see the [Amazon Relational Database Service User Guide](#).

The following points illustrate how Aurora relates to the standard MySQL and PostgreSQL engines available in Amazon RDS:

- You choose Aurora as the DB engine option when setting up new database servers through Amazon RDS.
- Aurora takes advantage of the familiar Amazon Relational Database Service (Amazon RDS) features for management and administration. Aurora uses the Amazon RDS AWS Management Console interface, AWS CLI commands, and API operations to handle routine database tasks such as provisioning, patching, backup, recovery, failure detection, and repair.
- Aurora management operations typically involve entire clusters of database servers that are synchronized through replication, instead of individual database instances. The automatic clustering, replication, and storage allocation make it simple and cost-effective to set up, operate, and scale your largest MySQL and PostgreSQL deployments.
- You can bring data from Amazon RDS for MySQL and Amazon RDS for PostgreSQL into Aurora by creating and restoring snapshots, or by setting up one-way replication. You can use push-button migration tools to convert your existing Amazon RDS for MySQL and Amazon RDS for PostgreSQL applications to Aurora.

Before using Amazon Aurora, you should complete the steps in [Setting up your environment for Amazon Aurora \(p. 84\)](#), and then review the concepts and features of Aurora in [Amazon Aurora DB clusters \(p. 3\)](#).

Topics

- [Amazon Aurora DB clusters \(p. 3\)](#)
- [Amazon Aurora versions \(p. 5\)](#)
- [Regions and Availability Zones \(p. 11\)](#)
- [Supported features in Amazon Aurora by AWS Region and Aurora DB engine \(p. 19\)](#)
- [Amazon Aurora connection management \(p. 32\)](#)
- [Aurora DB instance classes \(p. 54\)](#)
- [Amazon Aurora storage and reliability \(p. 64\)](#)

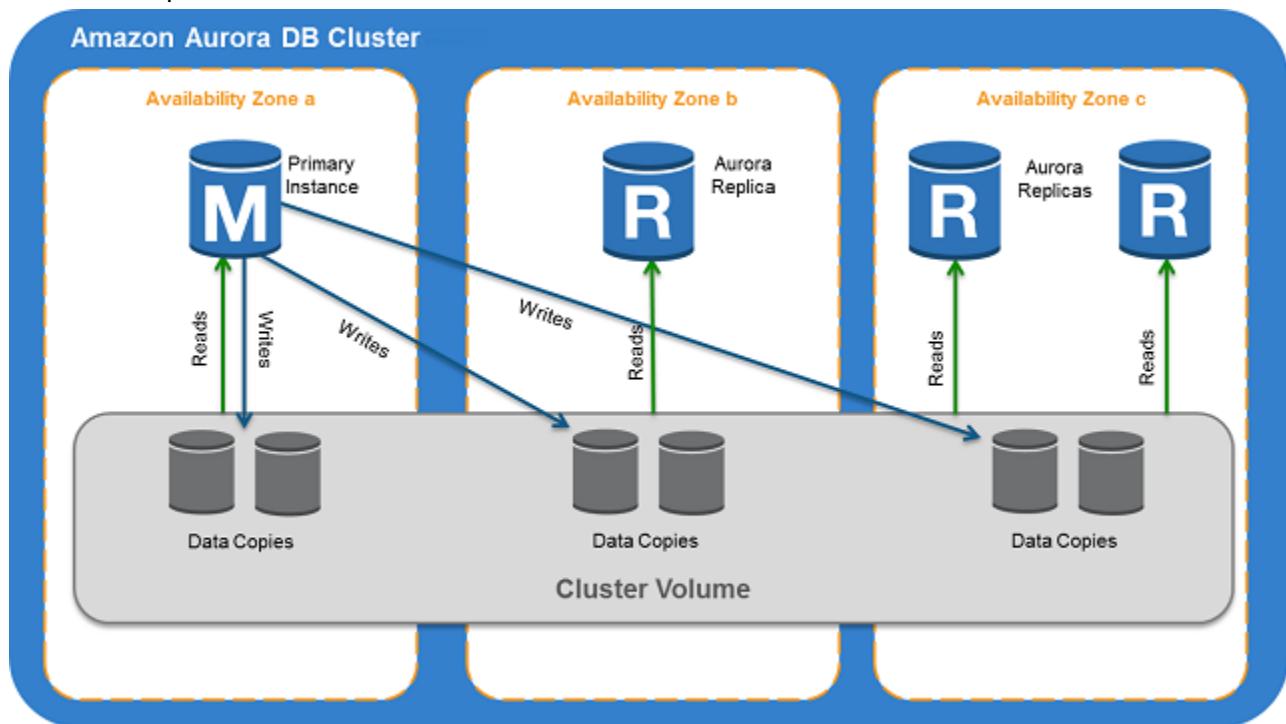
- [Amazon Aurora security \(p. 67\)](#)
- [High availability for Amazon Aurora \(p. 68\)](#)
- [Replication with Amazon Aurora \(p. 70\)](#)
- [DB instance billing for Aurora \(p. 72\)](#)

Amazon Aurora DB clusters

An Amazon Aurora *DB cluster* consists of one or more DB instances and a cluster volume that manages the data for those DB instances. An Aurora *cluster volume* is a virtual database storage volume that spans multiple Availability Zones, with each Availability Zone having a copy of the DB cluster data. Two types of DB instances make up an Aurora DB cluster:

- **Primary DB instance** – Supports read and write operations, and performs all of the data modifications to the cluster volume. Each Aurora DB cluster has one primary DB instance.
- **Aurora Replica** – Connects to the same storage volume as the primary DB instance and supports only read operations. Each Aurora DB cluster can have up to 15 Aurora Replicas in addition to the primary DB instance. Maintain high availability by locating Aurora Replicas in separate Availability Zones. Aurora automatically fails over to an Aurora Replica in case the primary DB instance becomes unavailable. You can specify the failover priority for Aurora Replicas. Aurora Replicas can also offload read workloads from the primary DB instance.

The following diagram illustrates the relationship between the cluster volume, the primary DB instance, and Aurora Replicas in an Aurora DB cluster.



Note

The preceding information applies to all the Aurora clusters that use single-master replication. These include provisioned clusters, parallel query clusters, global database clusters, serverless clusters, and all MySQL 8.0-compatible, 5.7-compatible, and PostgreSQL-compatible clusters. Aurora clusters that use multi-master replication have a different arrangement of read/write and read-only DB instances. All DB instances in a multi-master cluster can perform write operations. There isn't a single DB instance that performs all the write operations, and there aren't any read-only DB instances. Therefore, the terms *primary instance* and *Aurora Replica* don't apply to multi-master clusters. When we discuss clusters that might use multi-master replication, we refer to *writer* DB instances and *reader* DB instances.

The Aurora cluster illustrates the separation of compute capacity and storage. For example, an Aurora configuration with only a single DB instance is still a cluster, because the underlying storage volume involves multiple storage nodes distributed across multiple Availability Zones (AZs).

Amazon Aurora versions

Amazon Aurora reuses code and maintains compatibility with the underlying MySQL and PostgreSQL DB engines. However, Aurora has its own version numbers, release cycle, time line for version deprecation and end of life, and so on. The following section explains the common points and differences. This information can help you to decide such things as which version to choose and how to verify which features and fixes are available in each version. It can also help you to decide how often to upgrade and how to plan your upgrade process.

Topics

- [Relational databases that are available on Aurora \(p. 5\)](#)
- [Differences in version numbers between community databases and Aurora \(p. 5\)](#)
- [Amazon Aurora major versions \(p. 6\)](#)
- [Amazon Aurora minor versions \(p. 7\)](#)
- [Amazon Aurora patch versions \(p. 7\)](#)
- [Learning what's new in each Amazon Aurora version \(p. 7\)](#)
- [Specifying the Amazon Aurora database version for your database cluster \(p. 7\)](#)
- [Default Amazon Aurora versions \(p. 7\)](#)
- [Automatic minor version upgrades \(p. 8\)](#)
- [How long Amazon Aurora major versions remain available \(p. 8\)](#)
- [How often Amazon Aurora minor versions are released \(p. 9\)](#)
- [How long Amazon Aurora minor versions remain available \(p. 9\)](#)
- [Long-term support for selected Amazon Aurora minor versions \(p. 9\)](#)
- [Manually controlling if and when your database cluster is upgraded to new versions \(p. 9\)](#)
- [Required Amazon Aurora upgrades \(p. 10\)](#)
- [Testing your DB cluster with a new Aurora version before upgrading \(p. 10\)](#)

Relational databases that are available on Aurora

The following relational databases are available on Aurora:

- Amazon Aurora MySQL-Compatible Edition. For usage information, see [Working with Amazon Aurora MySQL \(p. 746\)](#). For a detailed list of available versions, see [Database engine updates for Amazon Aurora MySQL \(p. 1082\)](#).
- Amazon Aurora PostgreSQL-Compatible Edition. For usage information, see [Working with Amazon Aurora PostgreSQL \(p. 1275\)](#). For a detailed list of available versions, see [Amazon Aurora PostgreSQL updates \(p. 1597\)](#).

Differences in version numbers between community databases and Aurora

Each Amazon Aurora version is compatible with a specific community database version of either MySQL or PostgreSQL. You can find the community version of your database using the `version` function and the Aurora version using the `aurora_version` function.

Examples for Aurora MySQL and Aurora PostgreSQL are shown following.

```
mysql> select version();
+-----+
| version()      |
+-----+
| 5.7.12         |
+-----+

mysql> select aurora_version(), @@aurora_version;
+-----+-----+
| aurora_version() | @@aurora_version |
+-----+-----+
| 2.08.1          | 2.08.1           |
+-----+-----+
```

```
postgres=> select version();
-----
PostgreSQL 11.7 on x86_64-pc-linux-gnu, compiled by gcc (GCC) 4.9.3, 64-bit
(1 row)

postgres=> select aurora_version();
aurora_version
-----
3.2.2
```

For more information, see [Checking Aurora MySQL versions using SQL \(p. 1084\)](#) and [Identifying versions of Amazon Aurora PostgreSQL \(p. 1598\)](#).

Amazon Aurora major versions

Aurora versions use the *major.minor.patch* scheme. An *Aurora major version* refers to the MySQL or PostgreSQL community major version that Aurora is compatible with. The following example shows the mapping between community MySQL and PostgreSQL versions and the corresponding Aurora versions.

Community major version	Aurora major version
MySQL 5.6	Aurora MySQL 1
MySQL 5.7	Aurora MySQL 2
MySQL 8.0	Aurora MySQL 3
PostgreSQL 9.6	Aurora PostgreSQL 1
PostgreSQL 10	Aurora PostgreSQL 2. Not applicable for version 10.18 and higher versions. For these versions, the Aurora version is the same as the <i>major.minor</i> version of the PostgreSQL community version and a third digit in <i>patch</i> location.
PostgreSQL 11	Aurora PostgreSQL 3. Not applicable for version 11.13 and higher versions. For these versions, the Aurora version is the same as the <i>major.minor</i> version of the PostgreSQL community version and a third digit in <i>patch</i> location.
PostgreSQL 12	Aurora PostgreSQL 4. Not applicable for version 12.8 and higher versions. For these versions, the Aurora version is the same as the <i>major.minor</i>

Community major version	Aurora major version
	version of the PostgreSQL community version and a third digit in <i>patch</i> location.
PostgreSQL 13	Aurora PostgreSQL 4. Not applicable for version 13.3 and higher versions. For these versions, the Aurora version is the same as the <i>major.minor</i> version of the PostgreSQL community version and a third digit in <i>patch</i> location.

Amazon Aurora minor versions

Aurora versions use the *major.minor.patch* scheme. An *Aurora minor version* provides incremental community and Aurora-specific improvements to the service, for example new features and bug fixes.

Aurora minor versions are always mapped to a specific community version. However, some community versions might not have an Aurora equivalent.

Amazon Aurora patch versions

Aurora versions use the *major.minor.patch* scheme. An Aurora patch version includes important bug fixes added to a minor version after its initial release (for example, Aurora MySQL 2.04.0, 2.04.1, ..., 2.04.9). While each new minor version provides new Aurora features, new patch versions within a specific minor version are primarily used to resolve important issues.

For more information on patching, see [Maintaining an Amazon Aurora DB cluster \(p. 443\)](#).

Learning what's new in each Amazon Aurora version

Each new Aurora version comes with release notes that list the new features, fixes, other enhancements, and so on that apply to each version.

For Aurora MySQL release notes, see [Database engine updates for Amazon Aurora MySQL version 2 \(p. 1108\)](#) and [Database engine updates for Amazon Aurora MySQL version 1 \(p. 1196\)](#). For Aurora PostgreSQL release notes, see [Amazon Aurora PostgreSQL releases and engine versions \(p. 1599\)](#).

Specifying the Amazon Aurora database version for your database cluster

You can specify any currently available version (major and minor) when creating a new DB cluster using the **Create database** operation in the AWS Management Console, the AWS CLI, or the `CreateDBCluster` API operation. Not every Aurora database version is available in every AWS Region.

To learn how to create Aurora clusters, see [Creating an Amazon Aurora DB cluster \(p. 125\)](#). To learn how to change the version of an existing Aurora cluster, see [Modifying an Amazon Aurora DB cluster \(p. 372\)](#).

Default Amazon Aurora versions

When a new Aurora minor version contains significant improvements compared to a previous one, it's marked as the default version for new DB clusters. Typically, we release two default versions for each major version per year.

We recommend that you keep your DB cluster upgraded to the most current default minor version, because that version contains the latest security and functionality fixes.

Automatic minor version upgrades

You can stay up to date with Aurora minor versions by turning on **Auto minor version upgrade** for every DB instance in the Aurora cluster. Aurora only performs the automatic upgrade if all DB instances in your cluster have this setting turned on. Auto minor version upgrades are performed to the default minor version. We typically schedule automatic upgrades twice a year for DB clusters that have the **Auto minor version upgrade** setting set to **Yes**. These upgrades are started during the maintenance window that you specify for your cluster.

For more information, see [Enabling automatic upgrades between minor Aurora MySQL versions \(p. 1089\)](#) and [Automatic minor version upgrades for PostgreSQL \(p. 1688\)](#).

How long Amazon Aurora major versions remain available

Amazon Aurora major versions are made available at least until community end of life for the corresponding community version. You can use the following dates to plan your testing and upgrade cycles. These dates represent the minimum support period for each Aurora version. If Amazon extends support for an Aurora version for longer than originally planned, we plan to update this table to reflect the later date.

Database community version	Aurora version	Aurora version end of life no earlier than
MySQL 5.6	1	February 28, 2023, 00:00:00 UTC
MySQL 5.7	2	February 29, 2024
MySQL 8.0	3	
PostgreSQL 9.6	1	January 31, 2022
PostgreSQL 10	Varies depending on the minor version of the Aurora PostgreSQL release. For more information, see Amazon Aurora major versions (p. 6) .	January 31, 2023
PostgreSQL 11		January 31, 2024
PostgreSQL 12		January 31, 2025
PostgreSQL 13		January 31, 2026

Before each Aurora major version end of life, we provide a reminder at least 12 months in advance. We do so to communicate the detailed upgrade process. Details include the timing of certain milestones, the impact on your DB clusters, and the actions that we recommend that you take. We always recommend that you thoroughly test your applications against new database versions before performing a major version upgrade.

After this 12-month period, an automatic upgrade to the subsequent major version might be applied to any database cluster still running the older version. If so, the upgrade is started during scheduled maintenance windows.

How often Amazon Aurora minor versions are released

In general, Amazon Aurora minor versions are released quarterly. The release schedule might vary to pick up additional features or fixes.

How long Amazon Aurora minor versions remain available

We intend to make each Amazon Aurora minor version of a particular major version available for at least 12 months. At the end of this period, Aurora might apply an auto minor version upgrade to the subsequent default minor version. Such an upgrade is started during the scheduled maintenance window for any cluster that is still running the older minor version.

We might replace a minor version of a particular major version sooner than the usual 12-month period if there are critical matters such as security issues, or if the major version has reached end of life.

Before beginning automatic upgrades of minor versions that are approaching end of life, we generally provide a reminder three months in advance. We do so to communicate the detailed upgrade process. Details include the timing of certain milestones, the impact on your DB clusters, and the actions that we recommend that you take.

Long-term support for selected Amazon Aurora minor versions

For each Aurora major version, certain minor versions are designated as long-term-support (LTS) versions and made available for at least three years. That is, at least one minor version per major version is made available for longer than the typical 12 months. We generally provide a reminder six months before the end of this period. We do so to communicate the detailed upgrade process. Details include the timing of certain milestones, the impact on your DB clusters, and the actions that we recommend that you take.

LTS minor versions include only bug fixes (through patch versions). An LTS version doesn't include new features released after its introduction. Once a year, DB clusters running on an LTS minor version are patched to the latest patch version of the LTS release. We do this patching to help ensure that you benefit from cumulative security and stability fixes. We might patch an LTS minor version more frequently if there are critical fixes, such as for security, that need to be applied.

Note

If you want to remain on an LTS minor version for the duration of its lifecycle, make sure to turn off **Auto minor version upgrade** for your DB instances. To avoid automatically upgrading your DB cluster from the LTS minor version, set **Auto minor version upgrade** to **No** on all DB instances in your Aurora cluster.

For the version numbers of all Aurora LTS versions, see [Aurora MySQL long-term support \(LTS\) releases \(p. 1085\)](#) and [Aurora PostgreSQL long-term support \(LTS\) releases \(p. 1690\)](#).

Manually controlling if and when your database cluster is upgraded to new versions

Auto minor version upgrades are performed to the default minor version. We typically schedule automatic upgrades twice a year for DB clusters that have the **Auto minor version upgrade** setting set to **Yes**. These upgrades are started during customer-specified maintenance windows. If you want to turn

off automatic minor version upgrades, set **Auto minor version upgrade** to **No** on any DB instance within an Aurora cluster. Aurora performs an automatic minor version upgrade only if all DB instances in your cluster have the setting turned on.

Because major version upgrades involve some compatibility risk, they don't occur automatically. You must initiate these, except in the case of a major version upgrade due to end of life, as explained earlier. We always recommend that you thoroughly test your applications with new database versions before performing a major version upgrade.

For more information about upgrading a DB cluster to a new Aurora major version, see [Upgrading Amazon Aurora MySQL DB clusters \(p. 1088\)](#) and [Upgrading the PostgreSQL DB engine for Aurora PostgreSQL \(p. 1681\)](#).

Required Amazon Aurora upgrades

For certain critical fixes, we might perform a managed upgrade to a newer patch level within the same minor version. These required upgrades happen even if **Auto minor version upgrade** is turned off. Before doing so, we communicate the detailed upgrade process. Details include the timing of certain milestones, the impact on your DB clusters, and the actions that we recommend that you take. Such managed upgrades are performed automatically. Each such upgrade is started within the cluster maintenance window.

Testing your DB cluster with a new Aurora version before upgrading

You can test the upgrade process and how the new version works with your application and workload. Use one of the following methods:

- Clone your cluster using the Amazon Aurora fast database clone feature. Perform the upgrade and any post-upgrade testing on the new cluster.
- Restore from a cluster snapshot to create a new Aurora cluster. You can create a cluster snapshot yourself from an existing Aurora cluster. Aurora also automatically creates periodic snapshots for you for each of your clusters. You can then initiate a version upgrade for the new cluster. You can experiment on the upgraded copy of your cluster before deciding whether to upgrade your original cluster.

For more information on these ways to create new clusters for testing, see [Cloning a volume for an Aurora DB cluster \(p. 402\)](#) and [Creating a DB cluster snapshot \(p. 495\)](#).

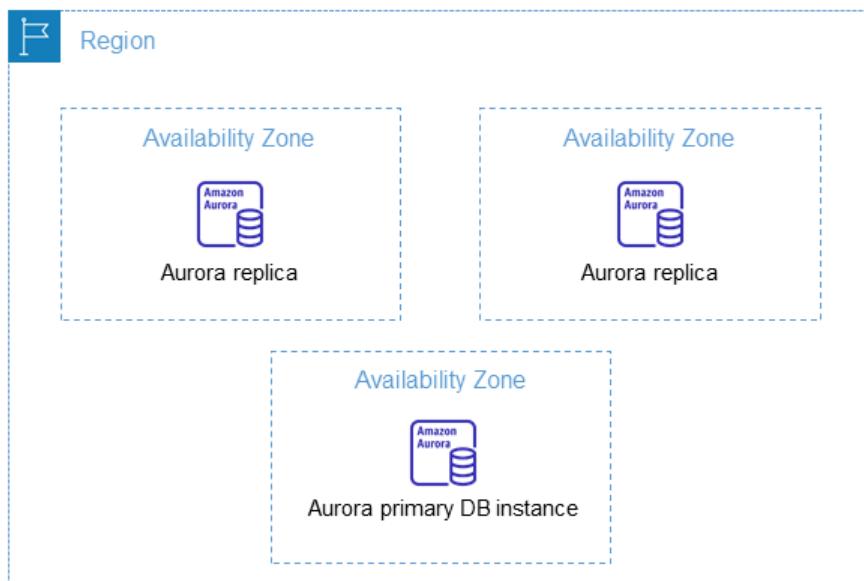
Regions and Availability Zones

Amazon cloud computing resources are hosted in multiple locations world-wide. These locations are composed of AWS Regions and Availability Zones. Each *AWS Region* is a separate geographic area. Each AWS Region has multiple, isolated locations known as *Availability Zones*.

Note

For information about finding the Availability Zones for an AWS Region, see [Describe Your Availability Zones](#) in the Amazon EC2 documentation.

Amazon operates state-of-the-art, highly-available data centers. Although rare, failures can occur that affect the availability of DB instances that are in the same location. If you host all your DB instances in a single location that is affected by such a failure, none of your DB instances will be available.



It is important to remember that each AWS Region is completely independent. Any Amazon RDS activity you initiate (for example, creating database instances or listing available database instances) runs only in your current default AWS Region. The default AWS Region can be changed in the console, by setting the `AWS_DEFAULT_REGION` environment variable, or it can be overridden by using the `--region` parameter with the AWS Command Line Interface (AWS CLI). For more information, see [Configuring the AWS Command Line Interface](#), specifically the sections about environment variables and command line options.

Amazon RDS supports special AWS Regions called AWS GovCloud (US) that are designed to allow US government agencies and customers to move more sensitive workloads into the cloud. The AWS GovCloud (US) Regions address the US government's specific regulatory and compliance requirements. For more information, see [What is AWS GovCloud \(US\)?](#)

To create or work with an Amazon RDS DB instance in a specific AWS Region, use the corresponding regional service endpoint.

Note

Aurora doesn't support Local Zones.

AWS Regions

Each AWS Region is designed to be isolated from the other AWS Regions. This design achieves the greatest possible fault tolerance and stability.

When you view your resources, you see only the resources that are tied to the AWS Region that you specified. This is because AWS Regions are isolated from each other, and we don't automatically replicate resources across AWS Regions.

Region availability

When you work with an Aurora DB cluster using the command line interface or API operations, make sure that you specify its regional endpoint.

Topics

- [Aurora MySQL Region availability \(p. 12\)](#)
- [Aurora PostgreSQL Region availability \(p. 14\)](#)

Aurora MySQL Region availability

The following table shows the AWS Regions where Aurora MySQL is currently available and the endpoint for each Region.

Region Name	Region	Endpoint	Protocol	
US East (Ohio)	us-east-2	rds.us-east-2.amazonaws.com	HTTPS	
US East (N. Virginia)	us-east-1	rds.us-east-1.amazonaws.com	HTTPS	
US West (N. California)	us-west-1	rds.us-west-1.amazonaws.com	HTTPS	
US West (Oregon)	us-west-2	rds.us-west-2.amazonaws.com	HTTPS	
Africa (Cape Town)	af-south-1	rds.af-south-1.amazonaws.com	HTTPS	
Asia Pacific (Hong Kong)	ap-east-1	rds.ap-east-1.amazonaws.com	HTTPS	
Asia Pacific (Jakarta)	ap-southeast-3	rds.ap-southeast-3.amazonaws.com	HTTPS	
Asia Pacific (Mumbai)	ap-south-1	rds.ap-south-1.amazonaws.com	HTTPS	
Asia Pacific (Osaka)	ap-northeast-3	rds.ap-northeast-3.amazonaws.com	HTTPS	

Region Name	Region	Endpoint	Protocol	
Asia Pacific (Seoul)	ap-northeast-2	rds.ap-northeast-2.amazonaws.com	HTTPS	
Asia Pacific (Singapore)	ap-southeast-1	rds.ap-southeast-1.amazonaws.com	HTTPS	
Asia Pacific (Sydney)	ap-southeast-2	rds.ap-southeast-2.amazonaws.com	HTTPS	
Asia Pacific (Tokyo)	ap-northeast-1	rds.ap-northeast-1.amazonaws.com	HTTPS	
Canada (Central)	ca-central-1	rds.ca-central-1.amazonaws.com	HTTPS	
Europe (Frankfurt)	eu-central-1	rds.eu-central-1.amazonaws.com	HTTPS	
Europe (Ireland)	eu-west-1	rds.eu-west-1.amazonaws.com	HTTPS	
Europe (London)	eu-west-2	rds.eu-west-2.amazonaws.com	HTTPS	
Europe (Milan)	eu-south-1	rds.eu-south-1.amazonaws.com	HTTPS	
Europe (Paris)	eu-west-3	rds.eu-west-3.amazonaws.com	HTTPS	
Europe (Stockholm)	eu-north-1	rds.eu-north-1.amazonaws.com	HTTPS	
Middle East (Bahrain)	me-south-1	rds.me-south-1.amazonaws.com	HTTPS	
South America (São Paulo)	sa-east-1	rds.sa-east-1.amazonaws.com	HTTPS	
AWS GovCloud (US-East)	us-gov-east-1	rds.us-gov-east-1.amazonaws.com	HTTPS	
AWS GovCloud (US-West)	us-gov-west-1	rds.us-gov-west-1.amazonaws.com	HTTPS	

Aurora PostgreSQL Region availability

The following table shows the AWS Regions where Aurora PostgreSQL is currently available and the endpoint for each Region.

Region Name	Region	Endpoint	Protocol	
US East (Ohio)	us-east-2	rds.us-east-2.amazonaws.com	HTTPS	
US East (N. Virginia)	us-east-1	rds.us-east-1.amazonaws.com	HTTPS	
US West (N. California)	us-west-1	rds.us-west-1.amazonaws.com	HTTPS	
US West (Oregon)	us-west-2	rds.us-west-2.amazonaws.com	HTTPS	
Africa (Cape Town)	af-south-1	rds.af-south-1.amazonaws.com	HTTPS	
Asia Pacific (Hong Kong)	ap-east-1	rds.ap-east-1.amazonaws.com	HTTPS	
Asia Pacific (Jakarta)	ap-southeast-3	rds.ap-southeast-3.amazonaws.com	HTTPS	
Asia Pacific (Mumbai)	ap-south-1	rds.ap-south-1.amazonaws.com	HTTPS	
Asia Pacific (Osaka)	ap-northeast-3	rds.ap-northeast-3.amazonaws.com	HTTPS	
Asia Pacific (Seoul)	ap-northeast-2	rds.ap-northeast-2.amazonaws.com	HTTPS	
Asia Pacific (Singapore)	ap-southeast-1	rds.ap-southeast-1.amazonaws.com	HTTPS	
Asia Pacific (Sydney)	ap-southeast-2	rds.ap-southeast-2.amazonaws.com	HTTPS	
Asia Pacific (Tokyo)	ap-northeast-1	rds.ap-northeast-1.amazonaws.com	HTTPS	

Region Name	Region	Endpoint	Protocol	
Canada (Central)	ca-central-1	rds.ca-central-1.amazonaws.com	HTTPS	
Europe (Frankfurt)	eu-central-1	rds.eu-central-1.amazonaws.com	HTTPS	
Europe (Ireland)	eu-west-1	rds.eu-west-1.amazonaws.com	HTTPS	
Europe (London)	eu-west-2	rds.eu-west-2.amazonaws.com	HTTPS	
Europe (Milan)	eu-south-1	rds.eu-south-1.amazonaws.com	HTTPS	
Europe (Paris)	eu-west-3	rds.eu-west-3.amazonaws.com	HTTPS	
Europe (Stockholm)	eu-north-1	rds.eu-north-1.amazonaws.com	HTTPS	
Middle East (Bahrain)	me-south-1	rds.me-south-1.amazonaws.com	HTTPS	
South America (São Paulo)	sa-east-1	rds.sa-east-1.amazonaws.com	HTTPS	
AWS GovCloud (US-East)	us-gov-east-1	rds.us-gov-east-1.amazonaws.com	HTTPS	
AWS GovCloud (US-West)	us-gov-west-1	rds.us-gov-west-1.amazonaws.com	HTTPS	

Availability Zones

An Availability Zone is an isolated location in a given AWS Region. Each Region has multiple Availability Zones (AZ) designed to provide high availability for the Region. An AZ is identified by the AWS Region code followed by a letter identifier (for example, `us-east-1a`). If you create your VPC and subnets rather than using the default VPC, you define each subnet in a specific AZ. When you create an Aurora DB cluster, Aurora creates the primary instance in one of the subnets in the VPC's DB subnet group, thus associating that instance with a specific AZ chosen by Aurora.

Each Aurora DB cluster hosts copies of its storage in three separate AZs. Every DB instance in the cluster must be in one of these three AZs. When you create a DB instance in your cluster, Aurora automatically chooses an appropriate AZ if you don't specify an AZ. If an AWS Region has fewer than three AZs, Aurora isn't available in that Region.

To learn how to specify the AZ when you create a cluster or add instances to it, see [VPC, subnets, and AZs \(p. 125\)](#).

Local time zone for Amazon Aurora DB clusters

By default, the time zone for an Amazon Aurora DB cluster is Universal Time Coordinated (UTC). You can set the time zone for instances in your DB cluster to the local time zone for your application instead.

To set the local time zone for a DB cluster, set the time zone parameter in the cluster parameter group for your DB cluster to one of the supported values listed later in this section. For Aurora MySQL, the name of this parameter is `time_zone`. For Aurora PostgreSQL, the name of this parameter is `timezone`. When you set the time zone parameter for a DB cluster, all instances in the DB cluster change to use the new local time zone. If other Aurora DB clusters are using the same cluster parameter group, then all instances in those DB clusters change to use the new local time zone also. For information on cluster-level parameters, see [Amazon Aurora DB cluster and DB instance parameters \(p. 341\)](#).

After you set the local time zone, all new connections to the database reflect the change. If you have any open connections to your database when you change the local time zone, you won't see the local time zone update until after you close the connection and open a new connection.

If you are replicating across AWS Regions, then the replication master DB cluster and the replica use different parameter groups (parameter groups are unique to an AWS Region). To use the same local time zone for each instance, you must set the time zone parameter in the parameter groups for both the replication master and the replica.

When you restore a DB cluster from a DB cluster snapshot, the local time zone is set to UTC. You can update the time zone to your local time zone after the restore is complete. If you restore a DB cluster to a point in time, then the local time zone for the restored DB cluster is the time zone setting from the parameter group of the restored DB cluster.

You can set your local time zone to one of the values listed in the following table. For some time zones, time values for certain date ranges can be reported incorrectly as noted in the table. For Australia time zones, the time zone abbreviation returned is an outdated value as noted in the table.

Time zone	Notes
Africa/Harare	This time zone setting can return incorrect values from 28 Feb 1903 21:49:40 GMT to 28 Feb 1903 21:55:48 GMT.
Africa/Monrovia	
Africa/Nairobi	This time zone setting can return incorrect values from 31 Dec 1939 21:30:00 GMT to 31 Dec 1959 21:15:15 GMT.
Africa/Windhoek	
America/Bogota	This time zone setting can return incorrect values from 23 Nov 1914 04:56:16 GMT to 23 Nov 1914 04:56:20 GMT.
America/Caracas	
America/Chihuahua	
America/Cuiaba	
America/Denver	
America/Fortaleza	If your DB cluster is in the South America (Sao Paulo) Region and the expected time doesn't show correctly for the recently changed Brazil time zone, reset the DB cluster's time zone parameter to <code>America/Fortaleza</code> .
America/Guatemala	

Time zone	Notes
America/Halifax	This time zone setting can return incorrect values from 27 Oct 1918 05:00:00 GMT to 31 Oct 1918 05:00:00 GMT.
America/Manaus	If your DB cluster is in the South America (Cuiaba) time zone and the expected time doesn't show correctly for the recently changed Brazil time zone, reset the DB cluster's time zone parameter to America/Manaus.
America/Matamoros	
America/Monterrey	
America/Montevideo	
America/Phoenix	
America/Tijuana	
Asia/Ashgabat	
Asia/Baghdad	
Asia/Baku	
Asia/Bangkok	
Asia/Beirut	
Asia/Calcutta	
Asia/Kabul	
Asia/Karachi	
Asia/Kathmandu	
Asia/Muscat	This time zone setting can return incorrect values from 31 Dec 1919 20:05:36 GMT to 31 Dec 1919 20:05:40 GMT.
Asia/Riyadh	This time zone setting can return incorrect values from 13 Mar 1947 20:53:08 GMT to 31 Dec 1949 20:53:08 GMT.
Asia/Seoul	This time zone setting can return incorrect values from 30 Nov 1904 15:30:00 GMT to 07 Sep 1945 15:00:00 GMT.
Asia/Shanghai	This time zone setting can return incorrect values from 31 Dec 1927 15:54:08 GMT to 02 Jun 1940 16:00:00 GMT.
Asia/Singapore	
Asia/Taipei	This time zone setting can return incorrect values from 30 Sep 1937 16:00:00 GMT to 29 Sep 1979 15:00:00 GMT.
Asia/Tehran	
Asia/Tokyo	This time zone setting can return incorrect values from 30 Sep 1937 15:00:00 GMT to 31 Dec 1937 15:00:00 GMT.
Asia/Ulaanbaatar	

Time zone	Notes
Atlantic/Azores	This time zone setting can return incorrect values from 24 May 1911 01:54:32 GMT to 01 Jan 1912 01:54:32 GMT.
Australia/Adelaide	The abbreviation for this time zone is returned as CST instead of ACDT/ACST.
Australia/Brisbane	The abbreviation for this time zone is returned as EST instead of AEDT/AEST.
Australia/Darwin	The abbreviation for this time zone is returned as CST instead of ACDT/ACST.
Australia/Hobart	The abbreviation for this time zone is returned as EST instead of AEDT/AEST.
Australia/Perth	The abbreviation for this time zone is returned as WST instead of AWDT/AWST.
Australia/Sydney	The abbreviation for this time zone is returned as EST instead of AEDT/AEST.
Brazil/East	
Canada/Saskatchewan	This time zone setting can return incorrect values from 27 Oct 1918 08:00:00 GMT to 31 Oct 1918 08:00:00 GMT.
Europe/Amsterdam	
Europe/Athens	
Europe/Dublin	
Europe/Helsinki	This time zone setting can return incorrect values from 30 Apr 1921 22:20:08 GMT to 30 Apr 1921 22:20:11 GMT.
Europe/Paris	
Europe/Prague	
Europe/Sarajevo	
Pacific/Auckland	
Pacific/Guam	
Pacific/Honolulu	This time zone setting can return incorrect values from 21 May 1933 11:30:00 GMT to 30 Sep 1945 11:30:00 GMT.
Pacific/Samoa	This time zone setting can return incorrect values from 01 Jan 1911 11:22:48 GMT to 01 Jan 1950 11:30:00 GMT.
US/Alaska	
US/Central	
US/Eastern	
US/East-Indiana	
US/Pacific	
UTC	

Supported features in Amazon Aurora by AWS Region and Aurora DB engine

Aurora MySQL- and PostgreSQL-compatible database engines support several Amazon Aurora and Amazon RDS features and options. The support varies across specific versions of each database engine, and across AWS Regions. You can use the tables in this section to identify Aurora database engine version support and availability in a given AWS Region for the following features:

Topics

- [Backtracking in Aurora \(p. 19\)](#)
- [Aurora global databases \(p. 21\)](#)
- [Aurora machine learning \(p. 23\)](#)
- [Aurora parallel queries \(p. 26\)](#)
- [Amazon RDS Proxy \(p. 27\)](#)
- [Aurora Serverless v1 \(p. 29\)](#)
- [Data API for Aurora Serverless \(p. 31\)](#)

Some of these features are Aurora-only capabilities. For example, Aurora Serverless, Aurora global databases, and support for integration with AWS machine learning services aren't supported by Amazon RDS. Other features, such as Amazon RDS Proxy, are supported by both Amazon Aurora and Amazon RDS.

The tables use the following patterns to specify version numbers and level of support:

- **Version x.y** – The specific version alone is supported.
- **Version x.y and higher** – The version and all minor versions are also supported. For example, "version 10.11 and higher" means that versions 10.11, 10.11.1, and 10.12 are also supported.
- - – The feature is not currently available for that particular Aurora feature for the given Aurora database engine, or in that specific AWS Region.

Backtracking in Aurora

By using backtracking in Aurora, you return the state of an Aurora cluster to a specific point in time, without restoring data from a backup. It completes within seconds, even for large databases. For more information, see [Backtracking an Aurora DB cluster \(p. 816\)](#).

Aurora backtracking is available for Aurora MySQL only. It's not available for Aurora PostgreSQL.

Region	Aurora MySQL 5.6	Aurora MySQL 5.7	Aurora MySQL 8.0
US East (Ohio)	Version 5.6.10a	Version 2.06 and higher	-
US East (N. Virginia)	Version 5.6.10a	Version 2.06 and higher	-
US West (N. California)	Version 5.6.10a	Version 2.06 and higher	-
US West (Oregon)	Version 5.6.10a	Version 2.06 and higher	-

Region	Aurora MySQL 5.6	Aurora MySQL 5.7	Aurora MySQL 8.0
Africa (Cape Town)	-	-	-
Asia Pacific (Hong Kong)	-	-	-
Asia Pacific (Jakarta)	-	-	-
Asia Pacific (Mumbai)	Version 5.6.10a	Version 2.06 and higher	-
Asia Pacific (Osaka)	Version 5.6.10a; version 1.22 and higher	Version 2.07.3 and higher	-
Asia Pacific (Seoul)	Version 5.6.10a	Version 2.06 and higher	-
Asia Pacific (Singapore)	Version 5.6.10a	Version 2.06 and higher	-
Asia Pacific (Sydney)	Version 5.6.10a	Version 2.06 and higher	-
Asia Pacific (Tokyo)	Version 5.6.10a	Version 2.06 and higher	-
Canada (Central)	Version 5.6.10a	Version 2.06 and higher	-
China (Beijing)	-	-	-
China (Ningxia)	-	-	-
Europe (Frankfurt)	Version 5.6.10a	Version 2.06 and higher	-
Europe (Ireland)	Version 5.6.10a	Version 2.06 and higher	-
Europe (London)	Version 5.6.10a	Version 2.06 and higher	-
Europe (Milan)	-	-	-
Europe (Paris)	Version 5.6.10a	Version 2.06 and higher	-
Europe (Stockholm)	-	-	-
Middle East (Bahrain)	-	-	-
South America (São Paulo)	-	-	-

Region	Aurora MySQL 5.6	Aurora MySQL 5.7	Aurora MySQL 8.0
AWS GovCloud (US-East)	-	-	-
AWS GovCloud (US-West)	-	-	-

Aurora global databases

An *Aurora global database* is a single database that spans multiple AWS Regions, enabling low-latency global reads and disaster recovery from any Region-wide outage. It provides built-in fault tolerance for your deployment because the DB instance relies not on a single AWS Region, but upon multiple Regions and different Availability Zones.

Support for this feature varies by Aurora database engine and version. The following table shows the Regions and Aurora database versions that support this feature. For more information, see [Using Amazon Aurora global databases \(p. 225\)](#).

Region	Aurora MySQL 5.6	Aurora MySQL 5.7	Aurora MySQL 8.0	Aurora PostgreSQL 10	Aurora PostgreSQL 11	Aurora PostgreSQL 12	Aurora PostgreSQL 13
US East (Ohio)	Version 5.6.10a; version 1.22 and higher	Version 2.07 and higher	Version 3.01.0 and higher	Version 10.11 and higher	Version 11.7 and higher	Version 12.4 and higher	Version 13.3 and higher
US East (N. Virginia)	Version 5.6.10a; version 1.22 and higher	Version 2.07 and higher	Version 3.01.0 and higher	Version 10.11 and higher	Version 11.7 and higher	Version 12.4 and higher	Version 13.3 and higher
US West (N. California)	Version 5.6.10a; version 1.22 and higher	Version 2.07 and higher	Version 3.01.0 and higher	Version 10.11 and higher	Version 11.7 and higher	Version 12.4 and higher	Version 13.3 and higher
US West (Oregon)	Version 5.6.10a; version 1.22 and higher	Version 2.07 and higher	Version 3.01.0 and higher	Version 10.11 and higher	Version 11.7 and higher	Version 12.4 and higher	Version 13.3 and higher
Africa (Cape Town)	-	-	-	-	-	-	-
Asia Pacific (Hong Kong)	-	-	-	-	-	-	-
Asia Pacific (Jakarta)	-	-	-	-	-	-	-

Region	Aurora MySQL 5.6	Aurora MySQL 5.7	Aurora MySQL 8.0	Aurora PostgreSQL 10	Aurora PostgreSQL 11	Aurora PostgreSQL 12	Aurora PostgreSQL 13
Asia Pacific (Mumbai)	Version 5.6.10a; Version 1.22 and higher	Version 2.07 and higher	Version 3.01.0 and higher	Version 10.11 and higher	Version 11.7 and higher	Version 12.4 and higher	Version 13.3 and higher
Asia Pacific (Osaka)	Version 1.22.3 and higher	Version 2.07.3 and higher	Version 3.01.0 and higher	Version 10.12 and higher	Version 11.7 and higher	Version 12.4 and higher	Version 13.3 and higher
Asia Pacific (Seoul)	Version 5.6.10a; Version 1.22 and higher	Version 2.07 and higher	Version 3.01.0 and higher	Version 10.11 and higher	Version 11.7 and higher	Version 12.4 and higher	Version 13.3 and higher
Asia Pacific (Singapore)	Version 5.6.10a; Version 1.22 and higher	Version 2.07 and higher	Version 3.01.0 and higher	Version 10.11 and higher	Version 11.7 and higher	Version 12.4 and higher	Version 13.3 and higher
Asia Pacific (Sydney)	Version 5.6.10a; Version 1.22 and higher	Version 2.07 and higher	Version 3.01.0 and higher	Version 10.11 and higher	Version 11.7 and higher	Version 12.4 and higher	Version 13.3 and higher
Asia Pacific (Tokyo)	Version 5.6.10a; Version 1.22 and higher	Version 2.07 and higher	Version 3.01.0 and higher	Version 10.11 and higher	Version 11.7 and higher	Version 12.4 and higher	Version 13.3 and higher
Canada (Central)	Version 5.6.10a; Version 1.22 and higher	Version 2.07 and higher	Version 3.01.0 and higher	Version 10.11 and higher	Version 11.7 and higher	Version 12.4 and higher	Version 13.3 and higher
China (Beijing)	Version 1.22.2 and higher	Version 2.07.2 and higher	Version 3.01.0 and higher	Version 10.12 and higher	Version 11.7 and higher	Version 12.4 and higher	Version 13.3 and higher
China (Ningxia)	Version 1.22.2 and higher	Version 2.07.2 and higher	Version 3.01.0 and higher	Version 10.12 and higher	Version 11.7 and higher	Version 12.4 and higher	Version 13.3 and higher
Europe (Frankfurt)	Version 5.6.10a; Version 1.22 and higher	Version 2.07 and higher	Version 3.01.0 and higher	Version 10.11 and higher	Version 11.7 and higher	Version 12.4 and higher	Version 13.3 and higher
Europe (Ireland)	Version 5.6.10a; Version 1.22 and higher	Version 2.07 and higher	Version 3.01.0 and higher	Version 10.11 and higher	Version 11.7 and higher	Version 12.4 and higher	Version 13.3 and higher

Region	Aurora MySQL 5.6	Aurora MySQL 5.7	Aurora MySQL 8.0	Aurora PostgreSQL 10	Aurora PostgreSQL 11	Aurora PostgreSQL 12	Aurora PostgreSQL 13
Europe (London)	Version 5.6.10a; Version 1.22 and higher	Version 2.07 and higher	Version 3.01.0 and higher	Version 10.11 and higher	Version 11.7 and higher	Version 12.4 and higher	Version 13.3 and higher
Europe (Milan)	-	-	-	-	-	-	-
Europe (Paris)	Version 5.6.10a; Version 1.22 and higher	Version 2.07 and higher	Version 3.01.0 and higher	Version 10.11 and higher	Version 11.7 and higher	Version 12.4 and higher	Version 13.3 and higher
Europe (Stockholm)	Version 1.22.2 and higher	Version 2.07.0 and higher	Version 3.01.0 and higher	Version 10.11 and higher	Version 11.7 and higher	Version 12.4 and higher	Version 13.3 and higher
Middle East (Bahrain)	-	-	-	-	-	-	-
South America (São Paulo)	Version 1.22.2 and higher	Version 2.07.1 and higher	Version 3.01.0 and higher	Version 10.11 and higher	Version 11.7 and higher	Version 12.4 and higher	Version 13.3 and higher
AWS GovCloud (US-East)	Version 5.6.10a; version 1.22 and higher	Version 2.07 and higher	Version 3.01.0 and higher	Version 10.11 and higher	Version 11.7 and higher	Version 12.4 and higher	Version 13.3 and higher
AWS GovCloud (US-West)	Version 5.6.10a; version 1.22 and higher	Version 2.07 and higher	Version 3.01.0 and higher	Version 10.11 and higher	Version 11.7 and higher	Version 12.4 and higher	Version 13.3 and higher

Aurora machine learning

Aurora machine learning provides simple, optimized, and secure integration between Aurora and AWS machine learning services without having to build custom integrations or move data around. Aurora exposes ML models as SQL functions, so you don't need to learn new programming languages or tools. Instead, you use standard SQL to build applications that call ML models, pass data to them, and return predictions as query results. For more information, see [Using machine learning \(ML\) capabilities with Amazon Aurora \(p. 442\)](#).

Region	Aurora MySQL 5.6	Aurora MySQL 5.7	Aurora MySQL 8.0	Aurora PostgreSQL 10	Aurora PostgreSQL 11	Aurora PostgreSQL 12	Aurora PostgreSQL 13
US East (Ohio)	-	Version 2.07 and higher	Version 3.01.0 and higher	Version 10.11	Version 11.6 and higher	Version 12.4 and higher	Version 13.3 and higher

Region	Aurora MySQL 5.6	Aurora MySQL 5.7	Aurora MySQL 8.0	Aurora PostgreSQL 10	Aurora PostgreSQL 11	Aurora PostgreSQL 12	Aurora PostgreSQL 13
US East (N. Virginia)	-	Version 2.07 and higher	Version 3.01.0 and higher	Version 10.11	Version 11.6 and higher	Version 12.4 and higher	Version 13.3 and higher
US West (N. California)	-	Version 2.07 and higher (SageMaker only)	Version 3.01.0 and higher (SageMaker only)	Version 10.11 (SageMaker only)	Version 11.6 and higher (SageMaker only)	Version 12.4 and higher (SageMaker only)	Version 13.3 and higher (SageMaker only)
US West (Oregon)	-	Version 2.07 and higher	Version 3.01.0 and higher	Version 10.11	Version 11.6 and higher	Version 12.4 and higher	Version 13.3 and higher
Africa (Cape Town)	-	-	-	-	-	-	-
Asia Pacific (Hong Kong)	-	Version 2.07 and higher (SageMaker only)	Version 3.01.0 and higher (SageMaker only)	Version 10.11	Version 11.6 and higher	Version 12.4 and higher	Version 13.3 and higher
Asia Pacific (Jakarta)	-	Version 2.07 and higher (SageMaker only)	Version 3.01.0 and higher (SageMaker only)	Version 10.11 (SageMaker only)	Version 11.6 and higher (SageMaker only)	Version 12.4 and higher	Version 13.3 and higher
Asia Pacific (Mumbai)	-	Version 2.07 and higher	Version 3.01.0 and higher	Version 10.11	Version 11.6 and higher	Version 12.4 and higher	Version 13.3 and higher
Asia Pacific (Osaka)	-	Version 2.07.3 and higher	Version 3.01.0 and higher	Version 10.11	Version 11.6 and higher	Version 12.4 and higher	Version 13.3 and higher
Asia Pacific (Seoul)	-	Version 2.07 and higher	Version 3.01.0 and higher	Version 10.11	Version 11.6 and higher	Version 12.4 and higher	Version 13.3 and higher
Asia Pacific (Singapore)	-	Version 2.07 and higher	Version 3.01.0 and higher	Version 10.11	Version 11.6 and higher	Version 12.4 and higher	Version 13.3 and higher
Asia Pacific (Sydney)	-	Version 2.07 and higher	Version 3.01.0 and higher	Version 10.11	Version 11.6 and higher	Version 12.4 and higher	Version 13.3 and higher
Asia Pacific (Tokyo)	-	Version 2.07 and higher	Version 3.01.0 and higher	Version 10.11	Version 11.6 and higher	Version 12.4 and higher	Version 13.3 and higher

Region	Aurora MySQL 5.6	Aurora MySQL 5.7	Aurora MySQL 8.0	Aurora PostgreSQL 10	Aurora PostgreSQL 11	Aurora PostgreSQL 12	Aurora PostgreSQL 13
Canada (Central)	-	Version 2.07 and higher	Version 3.01.0 and higher	Version 10.11	Version 11.6 and higher	Version 12.4 and higher	Version 13.3 and higher
China (Beijing)	-	Version 2.07 and higher (SageMaker only)	Version 3.01.0 and higher (SageMaker only)	Version 10.11 (SageMaker only)	Version 11.6 and higher (SageMaker only)	Version 12.4 and higher	Version 13.3 and higher
China (Ningxia)	-	Version 2.07 and higher (SageMaker only)	Version 3.01.0 and higher (SageMaker only)	Version 10.11 (SageMaker only)	Version 11.6 and higher (SageMaker only)	Version 12.4 and higher (SageMaker only)	Version 13.3 and higher (SageMaker only)
Europe (Frankfurt)	-	Version 2.07 and higher	Version 3.01.0 and higher	Version 10.11	Version 11.6 and higher	Version 12.4 and higher	Version 13.3 and higher
Europe (Ireland)	-	Version 2.07 and higher	Version 3.01.0 and higher	Version 10.11	Version 11.6 and higher	Version 12.4 and higher	Version 13.3 and higher
Europe (London)	-	Version 2.07 and higher	Version 3.01.0 and higher	Version 10.11	Version 11.6 and higher	Version 12.4 and higher	Version 13.3 and higher
Europe (Milan)	-	-	-	-	-	-	-
Europe (Paris)	-	Version 2.07 and higher (SageMaker only)	Version 3.01.0 and higher (SageMaker only)	Version 10.11 (SageMaker only)	Version 11.6 and higher (SageMaker only)	Version 12.4 and higher	Version 13.3 and higher
Europe (Stockholm)	-	Version 2.07 and higher (SageMaker only)	Version 3.01.0 and higher (SageMaker only)	Version 10.11 (SageMaker only)	Version 11.6 and higher (SageMaker only)	Version 12.4 and higher (SageMaker only)	Version 13.3 and higher (SageMaker only)
Middle East (Bahrain)	-	Version 2.07 and higher (SageMaker only)	Version 3.01.0 and higher (SageMaker only)	Version 10.11 (SageMaker only)	Version 11.6 and higher (SageMaker only)	Version 12.4 and higher (SageMaker only)	Version 13.3 and higher (SageMaker only)
South America (São Paulo)	-	Version 2.07 and higher (SageMaker only)	Version 3.01.0 and higher (SageMaker only)	Version 10.11 (SageMaker only)	Version 11.6 and higher (SageMaker only)	Version 12.4 and higher (SageMaker only)	Version 13.3 and higher (SageMaker only)

Region	Aurora MySQL 5.6	Aurora MySQL 5.7	Aurora MySQL 8.0	Aurora PostgreSQL 10	Aurora PostgreSQL 11	Aurora PostgreSQL 12	Aurora PostgreSQL 13
AWS GovCloud (US-East)	-	Version 2.07 and higher	Version 3.01.0 and higher	Version 10.11	Version 11.6 and higher	Version 12.4 and higher	Version 13.3 and higher
AWS GovCloud (US-West)	-	Version 2.07 and higher (SageMaker only)	Version 3.01.0 and higher (SageMaker only)	Version 10.11 (SageMaker only)	Version 11.6 and higher (SageMaker only)	Version 12.4 and higher (SageMaker only)	Version 13.3 and higher (SageMaker only)

Aurora parallel queries

Aurora parallel queries can speed up your queries by up to two orders of magnitude, while maintaining high throughput for your core transactional workload. Using the unique Aurora architecture, parallel queries can push down and parallelize query processing across thousands of CPUs in the Aurora storage layer. By offloading analytical query processing to the Aurora storage layer, parallel queries reduce network, CPU, and buffer pool contention for transactional workloads. For more information, see [Working with parallel query for Amazon Aurora MySQL \(p. 881\)](#). To learn more about Aurora MySQL versions available for parallel queries and any steps you might need to take based on that version to support parallel queries, see [Planning for a parallel query cluster \(p. 885\)](#).

Aurora parallel queries are available for Aurora MySQL only. However, PostgreSQL has its own parallel query feature that is available on Amazon RDS. The capability is enabled by default when a new PostgreSQL instance is created (versions 10.1 and higher). For more information, see [PostgreSQL on Amazon RDS](#).

Region	Aurora MySQL 5.6	Aurora MySQL 5.7	Aurora MySQL 8.0
US East (Ohio)	Version 1.23	Version 2.09 and higher	Version 3.01.0 and higher
US East (N. Virginia)	Version 1.23	Version 2.09 and higher	Version 3.01.0 and higher
US West (N. California)	Version 1.23	Version 2.09 and higher	Version 3.01.0 and higher
US West (Oregon)	Version 1.23	Version 2.09 and higher	Version 3.01.0 and higher
Africa (Cape Town)	Version 1.23	Version 2.09 and higher	Version 3.01.0 and higher
Asia Pacific (Hong Kong)	Version 1.23	Version 2.09 and higher	Version 3.01.0 and higher
Asia Pacific (Jakarta)	-	Version 2.10 and higher	Version 3.01.0 and higher
Asia Pacific (Mumbai)	Version 1.23	Version 2.09 and higher	Version 3.01.0 and higher
Asia Pacific (Osaka)	Version 1.23	Version 2.09 and higher	Version 3.01.0 and higher
Asia Pacific (Seoul)	Version 1.23	Version 2.09 and higher	Version 3.01.0 and higher
Asia Pacific (Singapore)	Version 1.23	Version 2.09 and higher	Version 3.01.0 and higher
Asia Pacific (Sydney)	Version 1.23	Version 2.09 and higher	Version 3.01.0 and higher
Asia Pacific (Tokyo)	Version 1.23	Version 2.09 and higher	Version 3.01.0 and higher

Region	Aurora MySQL 5.6	Aurora MySQL 5.7	Aurora MySQL 8.0
Canada (Central)	Version 1.23	Version 2.09 and higher	Version 3.01.0 and higher
China (Beijing)	Version 1.23	Version 2.09 and higher	Version 3.01.0 and higher
China (Ningxia)	Version 1.23	Version 2.09 and higher	Version 3.01.0 and higher
Europe (Frankfurt)	Version 1.23	Version 2.09 and higher	Version 3.01.0 and higher
Europe (Ireland)	Version 1.23	Version 2.09 and higher	Version 3.01.0 and higher
Europe (London)	Version 1.23	Version 2.09 and higher	Version 3.01.0 and higher
Europe (Milan)	Version 1.23	Version 2.09 and higher	Version 3.01.0 and higher
Europe (Paris)	Version 1.23	Version 2.09 and higher	Version 3.01.0 and higher
Europe (Stockholm)	Version 1.23	Version 2.09 and higher	Version 3.01.0 and higher
Middle East (Bahrain)	Version 1.23	Version 2.09 and higher	Version 3.01.0 and higher
South America (São Paulo)	Version 1.23	Version 2.09 and higher	Version 3.01.0 and higher
AWS GovCloud (US-East)	Version 1.23	Version 2.09 and higher	Version 3.01.0 and higher
AWS GovCloud (US-West)	Version 1.23	Version 2.09 and higher	Version 3.01.0 and higher

Amazon RDS Proxy

Amazon RDS Proxy is a fully managed, highly available database proxy that makes applications more scalable by pooling and sharing established database connections. For more information, see [Using Amazon RDS Proxy \(p. 288\)](#).

Region	Aurora MySQL 5.6	Aurora MySQL 5.7	Aurora MySQL 8.0	Aurora PostgreSQL 10	Aurora PostgreSQL 11	Aurora PostgreSQL 12
US East (Ohio)	Version 5.6.10a; version 1.22 and higher	Version 2.07 and higher	Version 3.01.0 and higher	Version 10.11 and higher	Version 11.6 and higher	Version 12.4 and higher
US East (N. Virginia)	Version 5.6.10a; version 1.22 and higher	Version 2.07 and higher	Version 3.01.0 and higher	Version 10.11 and higher	Version 11.6 and higher	Version 12.4 and higher
US West (N. California)	Version 5.6.10a; version 1.22 and higher	Version 2.07 and higher	Version 3.01.0 and higher	Version 10.11 and higher	Version 11.6 and higher	Version 12.4 and higher
US West (Oregon)	Version 5.6.10a; version 1.22 and higher	Version 2.07 and higher	Version 3.01.0 and higher	Version 10.11 and higher	Version 11.6 and higher	Version 12.4 and higher

Region	Aurora MySQL 5.6	Aurora MySQL 5.7	Aurora MySQL 8.0	Aurora PostgreSQL 10	Aurora PostgreSQL 11	Aurora PostgreSQL 12
Africa (Cape Town)	Version 5.6.10a; version 1.22 and higher	Version 2.07 and higher	Version 3.01.0 and higher	Version 10.11 and higher	Version 11.6 and higher	Version 12.4 and higher
Asia Pacific (Hong Kong)	Version 5.6.10a; version 1.22 and higher	Version 2.07 and higher	Version 3.01.0 and higher	Version 10.11 and higher	Version 11.6 and higher	Version 12.4 and higher
Asia Pacific (Jakarta)	-	-	-	-	-	-
Asia Pacific (Mumbai)	Version 5.6.10a; version 1.22 and higher	Version 2.07 and higher	Version 3.01.0 and higher	Version 10.11 and higher	Version 11.6 and higher	Version 12.4 and higher
Asia Pacific (Osaka)	Version 5.6.10a; version 1.22 and higher	Version 2.07 and higher	Version 3.01.0 and higher	Version 10.11 and higher	Version 11.6 and higher	Version 12.4 and higher
Asia Pacific (Seoul)	Version 5.6.10a; version 1.22 and higher	Version 2.07 and higher	Version 3.01.0 and higher	Version 10.11 and higher	Version 11.6 and higher	Version 12.4 and higher
Asia Pacific (Singapore)	Version 5.6.10a; version 1.22 and higher	Version 2.07 and higher	Version 3.01.0 and higher	Version 10.11 and higher	Version 11.6 and higher	Version 12.4 and higher
Asia Pacific (Sydney)	Version 5.6.10a; version 1.22 and higher	Version 2.07 and higher	Version 3.01.0 and higher	Version 10.11 and higher	Version 11.6 and higher	Version 12.4 and higher
Asia Pacific (Tokyo)	Version 5.6.10a; version 1.22 and higher	Version 2.07 and higher	Version 3.01.0 and higher	Version 10.11 and higher	Version 11.6 and higher	Version 12.4 and higher
Canada (Central)	Version 5.6.10a; version 1.22 and higher	Version 2.07 and higher	Version 3.01.0 and higher	Version 10.11 and higher	Version 11.6 and higher	Version 12.4 and higher
China (Beijing)	-	-	-	-	-	-
China (Ningxia)	-	-	-	-	-	-

Region	Aurora MySQL 5.6	Aurora MySQL 5.7	Aurora MySQL 8.0	Aurora PostgreSQL 10	Aurora PostgreSQL 11	Aurora PostgreSQL 12
Europe (Frankfurt)	Version 5.6.10a; version 1.22 and higher	Version 2.07 and higher	Version 3.01.0 and higher	Version 10.11 and higher	Version 11.6 and higher	Version 12.4 and higher
Europe (Ireland)	Version 5.6.10a; version 1.22 and higher	Version 2.07 and higher	Version 3.01.0 and higher	Version 10.11 and higher	Version 11.6 and higher	Version 12.4 and higher
Europe (London)	Version 5.6.10a; version 1.22 and higher	Version 2.07 and higher	Version 3.01.0 and higher	Version 10.11 and higher	Version 11.6 and higher	Version 12.4 and higher
Europe (Milan)	Version 5.6.10a; version 1.22 and higher	Version 2.07 and higher	Version 3.01.0 and higher	Version 10.11 and higher	Version 11.6 and higher	Version 12.4 and higher
Europe (Paris)	Version 5.6.10a; version 1.22 and higher	Version 2.07 and higher	Version 3.01.0 and higher	Version 10.11 and higher	Version 11.6 and higher	Version 12.4 and higher
Europe (Stockholm)	Version 5.6.10a; version 1.22 and higher	Version 2.07 and higher	Version 3.01.0 and higher	Version 10.11 and higher	Version 11.6 and higher	Version 12.4 and higher
Middle East (Bahrain)	Version 5.6.10a; version 1.22 and higher	Version 2.07 and higher	Version 3.01.0 and higher	Version 10.11 and higher	Version 11.6 and higher	Version 12.4 and higher
South America (São Paulo)	Version 5.6.10a; version 1.22 and higher	Version 2.07 and higher	Version 3.01.0 and higher	Version 10.11 and higher	Version 11.6 and higher	Version 12.4 and higher
AWS GovCloud (US-East)	-	-	-	-	-	-
AWS GovCloud (US-West)	-	-	-	-	-	-

Aurora Serverless v1

Aurora Serverless v1 is an on-demand, auto-scaling feature designed to be a cost-effective approach to running intermittent or unpredictable workloads on Amazon Aurora. It automatically starts up, shuts

down, and scales capacity up or down, as needed by your applications. For more information, see [Using Amazon Aurora Serverless v1 \(p. 147\)](#).

Region	Aurora MySQL 5.6	Aurora MySQL 5.7	Aurora MySQL 8.0	Aurora PostgreSQL 10	Aurora PostgreSQL 11
US East (Ohio)	Version 5.6.10a	Version 2.07.1	-	Version 10.12	-
US East (N. Virginia)	Version 5.6.10a	Version 2.07.1	-	Version 10.12	-
US West (N. California)	Version 5.6.10a	Version 2.07.1	-	Version 10.12	-
US West (Oregon)	Version 5.6.10a	Version 2.07.1	-	Version 10.12	-
Africa (Cape Town)	-	-	-	-	-
Asia Pacific (Hong Kong)	-	-	-	-	-
Asia Pacific (Jakarta)	-	-	-	-	-
Asia Pacific (Mumbai)	Version 5.6.10a	Version 2.07.1	-	Version 10.12	-
Asia Pacific (Osaka)	-	-	-	-	-
Asia Pacific (Seoul)	Version 5.6.10a	Version 2.07.1	-	Version 10.12	-
Asia Pacific (Singapore)	Version 5.6.10a	Version 2.07.1	-	Version 10.12	-
Asia Pacific (Sydney)	Version 5.6.10a	Version 2.07.1	-	Version 10.12	-
Asia Pacific (Tokyo)	Version 5.6.10a	Version 2.07.1	-	Version 10.12	-
Canada (Central)	Version 5.6.10a	Version 2.07.1	-	Version 10.12	-
China (Beijing)	-	-	-	-	-
China (Ningxia)	-	-	-	-	-
Europe (Frankfurt)	Version 5.6.10a	Version 2.07.1	-	Version 10.12	-
Europe (Ireland)	Version 5.6.10a	Version 2.07.1	-	Version 10.12	-
Europe (London)	Version 5.6.10a	Version 2.07.1	-	Version 10.12	-
Europe (Milan)	-	-	-	-	-
Europe (Paris)	Version 5.6.10a	Version 2.07.1	-	Version 10.12	-

Region	Aurora MySQL 5.6	Aurora MySQL 5.7	Aurora MySQL 8.0	Aurora PostgreSQL 10	Aurora PostgreSQL 11
Europe (Stockholm)	-	-	-	-	-
Middle East (Bahrain)	-	-	-	-	-
South America (São Paulo)	-	-	-	-	-
AWS GovCloud (US-East)	-	-	-	-	-
AWS GovCloud (US-West)	-	-	-	-	-

Data API for Aurora Serverless

The Data API for Aurora Serverless provides a web-services interface to an Aurora Serverless cluster. Rather than managing database connections from client applications, you can run SQL commands against an HTTPS endpoint. For more information, see [Using the Data API for Aurora Serverless \(p. 178\)](#).

Region	Aurora MySQL 5.6	Aurora MySQL 5.7	Aurora MySQL 8.0	Aurora PostgreSQL 10	Aurora PostgreSQL 11
US East (Ohio)	Version 5.6.10a	Version 2.07.1	-	Version 10.12	-
US East (N. Virginia)	Version 5.6.10a	Version 2.07.1	-	Version 10.12	-
US West (N. California)	Version 5.6.10a	Version 2.07.1	-	Version 10.12	-
US West (Oregon)	Version 5.6.10a	Version 2.07.1	-	Version 10.12	-
Africa (Cape Town)	-	-	-	-	-
Asia Pacific (Hong Kong)	-	-	-	-	-
Asia Pacific (Jakarta)	-	-	-	-	-
Asia Pacific (Mumbai)	Version 5.6.10a	Version 2.07.1	-	Version 10.12	-
Asia Pacific (Osaka)	-	-	-	-	-
Asia Pacific (Seoul)	Version 5.6.10a	Version 2.07.1	-	Version 10.12	-

Region	Aurora MySQL 5.6	Aurora MySQL 5.7	Aurora MySQL 8.0	Aurora PostgreSQL 10	Aurora PostgreSQL 11
Asia Pacific (Singapore)	Version 5.6.10a	Version 2.07.1	-	Version 10.12	-
Asia Pacific (Sydney)	Version 5.6.10a	Version 2.07.1	-	Version 10.12	-
Asia Pacific (Tokyo)	Version 5.6.10a	Version 2.07.1	-	Version 10.12	-
Canada (Central)	Version 5.6.10a	Version 2.07.1	-	Version 10.12	-
China (Beijing)	-	-	-	-	-
China (Ningxia)	-	-	-	-	-
Europe (Frankfurt)	Version 5.6.10a	Version 2.07.1	-	Version 10.12	-
Europe (Ireland)	Version 5.6.10a	Version 2.07.1	-	Version 10.12	-
Europe (London)	Version 5.6.10a	Version 2.07.1	-	Version 10.12	-
Europe (Milan)	-	-	-	-	-
Europe (Paris)	Version 5.6.10a	Version 2.07.1	-	Version 10.12	-
Europe (Stockholm)	-	-	-	-	-
Middle East (Bahrain)	-	-	-	-	-
South America (São Paulo)	-	-	-	-	-
AWS GovCloud (US-East)	-	-	-	-	-
AWS GovCloud (US-West)	-	-	-	-	-

Amazon Aurora connection management

Amazon Aurora typically involves a cluster of DB instances instead of a single instance. Each connection is handled by a specific DB instance. When you connect to an Aurora cluster, the host name and port that you specify point to an intermediate handler called an *endpoint*. Aurora uses the endpoint mechanism to abstract these connections. Thus, you don't have to hardcode all the hostnames or write your own logic for load-balancing and rerouting connections when some DB instances aren't available.

For certain Aurora tasks, different instances or groups of instances perform different roles. For example, the primary instance handles all data definition language (DDL) and data manipulation language (DML) statements. Up to 15 Aurora Replicas handle read-only query traffic.

Using endpoints, you can map each connection to the appropriate instance or group of instances based on your use case. For example, to perform DDL statements you can connect to whichever instance is the

primary instance. To perform queries, you can connect to the reader endpoint, with Aurora automatically performing load-balancing among all the Aurora Replicas. For clusters with DB instances of different capacities or configurations, you can connect to custom endpoints associated with different subsets of DB instances. For diagnosis or tuning, you can connect to a specific instance endpoint to examine details about a specific DB instance.

Topics

- [Types of Aurora endpoints \(p. 33\)](#)
- [Viewing the endpoints for an Aurora cluster \(p. 35\)](#)
- [Using the cluster endpoint \(p. 35\)](#)
- [Using the reader endpoint \(p. 35\)](#)
- [Using custom endpoints \(p. 36\)](#)
- [Creating a custom endpoint \(p. 38\)](#)
- [Viewing custom endpoints \(p. 40\)](#)
- [Editing a custom endpoint \(p. 45\)](#)
- [Deleting a custom endpoint \(p. 47\)](#)
- [End-to-end AWS CLI example for custom endpoints \(p. 48\)](#)
- [Using the instance endpoints \(p. 52\)](#)
- [How Aurora endpoints work with high availability \(p. 52\)](#)

Types of Aurora endpoints

An endpoint is represented as an Aurora-specific URL that contains a host address and a port. The following types of endpoints are available from an Aurora DB cluster.

Cluster endpoint

A *cluster endpoint* (or *writer endpoint*) for an Aurora DB cluster connects to the current primary DB instance for that DB cluster. This endpoint is the only one that can perform write operations such as DDL statements. Because of this, the cluster endpoint is the one that you connect to when you first set up a cluster or when your cluster only contains a single DB instance.

Each Aurora DB cluster has one cluster endpoint and one primary DB instance.

You use the cluster endpoint for all write operations on the DB cluster, including inserts, updates, deletes, and DDL changes. You can also use the cluster endpoint for read operations, such as queries.

The cluster endpoint provides failover support for read/write connections to the DB cluster. If the current primary DB instance of a DB cluster fails, Aurora automatically fails over to a new primary DB instance. During a failover, the DB cluster continues to serve connection requests to the cluster endpoint from the new primary DB instance, with minimal interruption of service.

The following example illustrates a cluster endpoint for an Aurora MySQL DB cluster.

```
mydbcluster.cluster-123456789012.us-east-1.rds.amazonaws.com:3306
```

Reader endpoint

A *reader endpoint* for an Aurora DB cluster provides load-balancing support for read-only connections to the DB cluster. Use the reader endpoint for read operations, such as queries. By processing those statements on the read-only Aurora Replicas, this endpoint reduces the overhead on the primary instance. It also helps the cluster to scale the capacity to handle simultaneous

`SELECT` queries, proportional to the number of Aurora Replicas in the cluster. Each Aurora DB cluster has one reader endpoint.

If the cluster contains one or more Aurora Replicas, the reader endpoint load-balances each connection request among the Aurora Replicas. In that case, you can only perform read-only statements such as `SELECT` in that session. If the cluster only contains a primary instance and no Aurora Replicas, the reader endpoint connects to the primary instance. In that case, you can perform write operations through the endpoint.

The following example illustrates a reader endpoint for an Aurora MySQL DB cluster.

```
mydbcluster.cluster-ro-123456789012.us-east-1.rds.amazonaws.com:3306
```

Custom endpoint

A *custom endpoint* for an Aurora cluster represents a set of DB instances that you choose. When you connect to the endpoint, Aurora performs load balancing and chooses one of the instances in the group to handle the connection. You define which instances this endpoint refers to, and you decide what purpose the endpoint serves.

An Aurora DB cluster has no custom endpoints until you create one. You can create up to five custom endpoints for each provisioned Aurora cluster. You can't use custom endpoints for Aurora Serverless clusters.

The custom endpoint provides load-balanced database connections based on criteria other than the read-only or read/write capability of the DB instances. For example, you might define a custom endpoint to connect to instances that use a particular AWS instance class or a particular DB parameter group. Then you might tell particular groups of users about this custom endpoint. For example, you might direct internal users to low-capacity instances for report generation or ad hoc (one-time) querying, and direct production traffic to high-capacity instances.

Because the connection can go to any DB instance that is associated with the custom endpoint, we recommend that you make sure that all the DB instances within that group share some similar characteristic. Doing so ensures that the performance, memory capacity, and so on, are consistent for everyone who connects to that endpoint.

This feature is intended for advanced users with specialized kinds of workloads where it isn't practical to keep all the Aurora Replicas in the cluster identical. With custom endpoints, you can predict the capacity of the DB instance used for each connection. When you use custom endpoints, you typically don't use the reader endpoint for that cluster.

The following example illustrates a custom endpoint for a DB instance in an Aurora MySQL DB cluster.

```
myendpoint.cluster-custom-123456789012.us-east-1.rds.amazonaws.com:3306
```

Instance endpoint

An *instance endpoint* connects to a specific DB instance within an Aurora cluster. Each DB instance in a DB cluster has its own unique instance endpoint. So there is one instance endpoint for the current primary DB instance of the DB cluster, and there is one instance endpoint for each of the Aurora Replicas in the DB cluster.

The instance endpoint provides direct control over connections to the DB cluster, for scenarios where using the cluster endpoint or reader endpoint might not be appropriate. For example, your client application might require more fine-grained load balancing based on workload type. In this case, you can configure multiple clients to connect to different Aurora Replicas in a DB cluster to distribute read workloads. For an example that uses instance endpoints to improve connection speed after a failover for Aurora PostgreSQL, see [Fast failover with Amazon Aurora PostgreSQL \(p. 1423\)](#). For

an example that uses instance endpoints to improve connection speed after a failover for Aurora MySQL, see [MariaDB Connector/J failover support - case Amazon Aurora](#).

The following example illustrates an instance endpoint for a DB instance in an Aurora MySQL DB cluster.

```
mydbinstance.123456789012.us-east-1.rds.amazonaws.com:3306
```

Viewing the endpoints for an Aurora cluster

In the AWS Management Console, you see the cluster endpoint, the reader endpoint, and any custom endpoints in the detail page for each cluster. You see the instance endpoint in the detail page for each instance. When you connect, you must append the associated port number, following a colon, to the endpoint name shown on this detail page.

With the AWS CLI, you see the writer, reader, and any custom endpoints in the output of the [describe-db-clusters](#) command. For example, the following command shows the endpoint attributes for all clusters in your current AWS Region.

```
aws rds describe-db-clusters --query '*[ ].  
{Endpoint:Endpoint,ReaderEndpoint:ReaderEndpoint,CustomerEndpoints:CustomEndpoints}'
```

With the Amazon RDS API, you retrieve the endpoints by calling the [DescribeDBClusterEndpoints](#) function.

Using the cluster endpoint

Because each Aurora cluster has a single built-in cluster endpoint, whose name and other attributes are managed by Aurora, you can't create, delete, or modify this kind of endpoint.

You use the cluster endpoint when you administer your cluster, perform extract, transform, load (ETL) operations, or develop and test applications. The cluster endpoint connects to the primary instance of the cluster. The primary instance is the only DB instance where you can create tables and indexes, run `INSERT` statements, and perform other DDL and DML operations.

The physical IP address pointed to by the cluster endpoint changes when the failover mechanism promotes a new DB instance to be the read/write primary instance for the cluster. If you use any form of connection pooling or other multiplexing, be prepared to flush or reduce the time-to-live for any cached DNS information. Doing so ensures that you don't try to establish a read/write connection to a DB instance that became unavailable or is now read-only after a failover.

Using the reader endpoint

You use the reader endpoint for read-only connections for your Aurora cluster. This endpoint uses a load-balancing mechanism to help your cluster handle a query-intensive workload. The reader endpoint is the endpoint that you supply to applications that do reporting or other read-only operations on the cluster.

The reader endpoint load-balances connections to available Aurora Replicas in an Aurora DB cluster. It doesn't load-balance individual queries. If you want to load-balance each query to distribute the read workload for a DB cluster, open a new connection to the reader endpoint for each query.

Each Aurora cluster has a single built-in reader endpoint, whose name and other attributes are managed by Aurora. You can't create, delete, or modify this kind of endpoint.

If your cluster contains only a primary instance and no Aurora Replicas, the reader endpoint connects to the primary instance. In that case, you can perform write operations through this endpoint.

Tip

Through RDS Proxy, you can create additional read-only endpoints for an Aurora cluster. These endpoints perform the same kind of load-balancing as the Aurora reader endpoint. Applications can reconnect more quickly to the proxy endpoints than the Aurora reader endpoint if reader instances become unavailable. The proxy endpoints can also take advantage of other proxy features such as multiplexing. For more information, see [Using reader endpoints with Aurora clusters \(p. 316\)](#).

Using custom endpoints

You use custom endpoints to simplify connection management when your cluster contains DB instances with different capacities and configuration settings.

Previously, you might have used the CNAMES mechanism to set up Domain Name Service (DNS) aliases from your own domain to achieve similar results. By using custom endpoints, you can avoid updating CNAME records when your cluster grows or shrinks. Custom endpoints also mean that you can use encrypted Transport Layer Security/Secure Sockets Layer (TLS/SSL) connections.

Instead of using one DB instance for each specialized purpose and connecting to its instance endpoint, you can have multiple groups of specialized DB instances. In this case, each group has its own custom endpoint. This way, Aurora can perform load balancing among all the instances dedicated to tasks such as reporting or handling production or internal queries. The custom endpoints provide load balancing and high availability for each group of DB instances within your cluster. If one of the DB instances within a group becomes unavailable, Aurora directs subsequent custom endpoint connections to one of the other DB instances associated with the same endpoint.

Topics

- [Specifying properties for custom endpoints \(p. 36\)](#)
- [Membership rules for custom endpoints \(p. 37\)](#)
- [Managing custom endpoints \(p. 37\)](#)

Specifying properties for custom endpoints

The maximum length for a custom endpoint name is 63 characters. You can see the name format following:

`endpointName.cluster-custom-customerDnsIdentifier.dnsSuffix`

Because custom endpoint names don't include the name of your cluster, you don't have to change those names if you rename a cluster. You can't reuse the same custom endpoint name for more than one cluster in the same region. Give each custom endpoint a name that is unique across the clusters owned by your user ID within a particular region.

Each custom endpoint has an associated type that determines which DB instances are eligible to be associated with that endpoint. Currently, the type can be READER, WRITER, or ANY. The following considerations apply to the custom endpoint types:

- Only DB instances that are read-only Aurora Replicas can be part of a READER custom endpoint. The READER type applies only to clusters using single-master replication, because those clusters can include multiple read-only DB instances.
- Both read-only Aurora Replicas and the read/write primary instance can be part of an ANY custom endpoint. Aurora directs connections to cluster endpoints with type ANY to any associated DB instance with equal probability. Because you can't determine in advance if you are connecting to the primary instance of a read-only Aurora Replica, use this kind of endpoint for read-only connections only. The ANY type applies to clusters using any replication topology.

- The `WRITER` type applies only to multi-master clusters, because those clusters can include multiple read/write DB instances.
- If you try to create a custom endpoint with a type that isn't appropriate based on the replication configuration for a cluster, Aurora returns an error.

Membership rules for custom endpoints

When you add a DB instance to a custom endpoint or remove it from a custom endpoint, any existing connections to that DB instance remain active.

You can define a list of DB instances to include in, or exclude from, a custom endpoint. We refer to these lists as *static* and *exclusion* lists, respectively. You can use the inclusion/exclusion mechanism to further subdivide the groups of DB instances, and to make sure that the set of custom endpoints covers all the DB instances in the cluster. Each custom endpoint can contain only one of these list types.

In the AWS Management Console, the choice is represented by the check box **Attach future instances added to this cluster**. When you keep check box clear, the custom endpoint uses a static list containing only the DB instances specified in the dialog. When you choose the check box, the custom endpoint uses an exclusion list. In this case, the custom endpoint represents all DB instances in the cluster (including any that you add in the future) except the ones left unselected in the dialog. The AWS CLI and Amazon RDS API have parameters representing each kind of list. When you use the AWS CLI or Amazon RDS API, you can't add or remove individual members to the lists; you always specify the entire new list.

Aurora doesn't change the DB instances specified in the static or exclusion lists when DB instances change roles between primary instance and Aurora Replica due to failover or promotion. For example, a custom endpoint with type `READER` might include a DB instance that was an Aurora Replica and then was promoted to a primary instance. The type of a custom endpoint (`READER`, `WRITER`, or `ANY`) determines what kinds of operations you can perform through that endpoint.

You can associate a DB instance with more than one custom endpoint. For example, suppose that you add a new DB instance to a cluster, or that Aurora adds a DB instance automatically through the autoscaling mechanism. In these cases, the DB instance is added to all custom endpoints for which it is eligible. Which endpoints the DB instance is added to is based on the custom endpoint type of `READER`, `WRITER`, or `ANY`, plus any static or exclusion lists defined for each endpoint. For example, if the endpoint includes a static list of DB instances, newly added Aurora Replicas aren't added to that endpoint. Conversely, if the endpoint has an exclusion list, newly added Aurora Replicas are added to the endpoint, if they aren't named in the exclusion list and their roles match the type of the custom endpoint.

If an Aurora Replica becomes unavailable, it remains associated with any custom endpoints. For example, it remains part of the custom endpoint when it is unhealthy, stopped, rebooting, and so on. However, you can't connect to it through those endpoints until it becomes available again.

Managing custom endpoints

Because newly created Aurora clusters have no custom endpoints, you must create and manage these objects yourself. You do so using the AWS Management Console, AWS CLI, or Amazon RDS API.

Note

You must also create and manage custom endpoints for Aurora clusters restored from snapshots. Custom endpoints are not included in the snapshot. You create them again after restoring, and choose new endpoint names if the restored cluster is in the same region as the original one.

To work with custom endpoints from the AWS Management Console, you navigate to the details page for your Aurora cluster and use the controls under the **Custom Endpoints** section.

To work with custom endpoints from the AWS CLI, you can use these operations:

- [create-db-cluster-endpoint](#)
- [describe-db-cluster-endpoints](#)
- [modify-db-cluster-endpoint](#)
- [delete-db-cluster-endpoint](#)

To work with custom endpoints through the Amazon RDS API, you can use the following functions:

- [CreateDBClusterEndpoint](#)
- [DescribeDBClusterEndpoints](#)
- [ModifyDBClusterEndpoint](#)
- [DeleteDBClusterEndpoint](#)

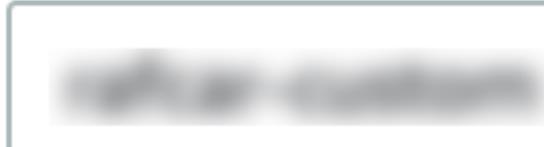
Creating a custom endpoint

Console

To create a custom endpoint with the AWS Management Console, go to the cluster detail page and choose the `Create custom endpoint` action in the **Endpoints** section. Choose a name for the custom endpoint, unique for your user ID and region. To choose a list of DB instances that remains the same even as the cluster expands, keep the check box **Attach future instances added to this cluster** clear. When you choose that check box, the custom endpoint dynamically adds any new instances as you add them to the cluster.

Create custom e

Endpoint name



Endpoint name is case insensitive,
First character must be a letter. Ca

Endpoint members



Filter database

You can't select the custom endpoint type of **ANY** or **READER** in the AWS Management Console. All the custom endpoints you create through the AWS Management Console have a type of **ANY**.

AWS CLI

To create a custom endpoint with the AWS CLI, run the [create-db-cluster-endpoint](#) command.

The following command creates a custom endpoint attached to a specific cluster. Initially, the endpoint is associated with all the Aurora Replica instances in the cluster. A subsequent command associates it with a specific set of DB instances in the cluster.

For Linux, macOS, or Unix:

```
aws rds create-db-cluster-endpoint --db-cluster-endpoint-identifier custom-endpoint-doc-sample \
--endpoint-type reader \
--db-cluster-identifier cluster_id

aws rds modify-db-cluster-endpoint --db-cluster-endpoint-identifier custom-endpoint-doc-sample \
--static-members instance_name_1 instance_name_2
```

For Windows:

```
aws rds create-db-cluster-endpoint --db-cluster-endpoint-identifier custom-endpoint-doc-sample ^
--endpoint-type reader ^
--db-cluster-identifier cluster_id

aws rds modify-db-cluster-endpoint --db-cluster-endpoint-identifier custom-endpoint-doc-sample ^
--static-members instance_name_1 instance_name_2
```

RDS API

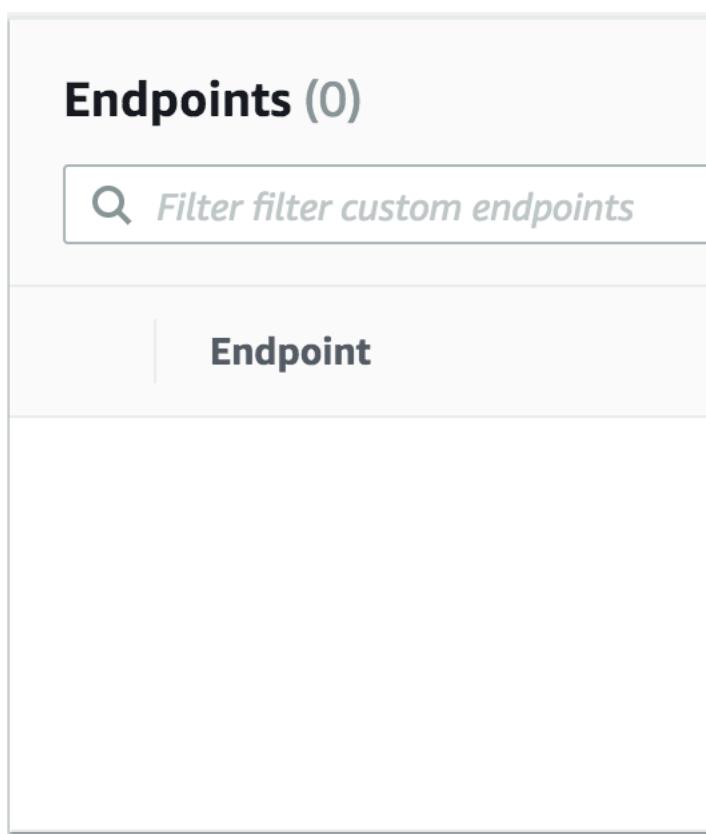
To create a custom endpoint with the RDS API, run the [CreateDBClusterEndpoint](#) operation.

Viewing custom endpoints

Console

To view custom endpoints with the AWS Management Console, go to the cluster detail page for the cluster and look under the **Endpoints** section. This section contains information only about custom endpoints. The details for the built-in endpoints are listed in the main **Details** section. To see the details for a specific custom endpoint, select its name to bring up the detail page for that endpoint.

The following screenshot shows how the list of custom endpoints for an Aurora cluster is initially empty.



After you create some custom endpoints for that cluster, they are shown under the **Endpoints** section.

Endpoints (2)



Filter filter custom endpoints

Endpoint



.cluster-custo



.cluster-custo

Clicking through to the detail page shows which DB instances the endpoint is currently associated with.

RDS > Clusters: [REDACTED]

Details

Endpoint name

[REDACTED]

Endpoint members



43

Filter endpoint members

To see the additional detail of whether new DB instances added to the cluster are automatically added to the endpoint also, bring up the **Edit** dialog for the endpoint.

AWS CLI

To view custom endpoints with the AWS CLI, run the [describe-db-cluster-endpoints](#) command.

The following command shows the custom endpoints associated with a specified cluster in a specified region. The output includes both the built-in endpoints and any custom endpoints.

For Linux, macOS, or Unix:

```
aws rds describe-db-cluster-endpoints --region region_name \  
--db-cluster-identifier cluster_id
```

For Windows:

```
aws rds describe-db-cluster-endpoints --region region_name ^  
--db-cluster-identifier cluster_id
```

The following shows some sample output from a `describe-db-cluster-endpoints` command. The `EndpointType` of WRITER or READER denotes the built-in read/write and read-only endpoints for the cluster. The `EndpointType` of CUSTOM denotes endpoints that you create and choose the associated DB instances. One of the endpoints has a non-empty `StaticMembers` field, denoting that it is associated with a precise set of DB instances. The other endpoint has a non-empty `ExcludedMembers` field, denoting that the endpoint is associated with all DB instances *other than* the ones listed under `ExcludedMembers`. This second kind of custom endpoint grows to accommodate new instances as you add them to the cluster.

```
{  
  "DBClusterEndpoints": [  
    {  
      "Endpoint": "custom-endpoint-demo.cluster-123456789012.ca-central-1.rds.amazonaws.com",  
      "Status": "available",  
      "DBClusterIdentifier": "custom-endpoint-demo",  
      "EndpointType": "WRITER"  
    },  
    {  
      "Endpoint": "custom-endpoint-demo.cluster-ro-123456789012.ca-central-1.rds.amazonaws.com",  
      "Status": "available",  
      "DBClusterIdentifier": "custom-endpoint-demo",  
      "EndpointType": "READER"  
    },  
    {  
      "CustomEndpointType": "ANY",  
      "DBClusterEndpointIdentifier": "powers-of-2",  
      "ExcludedMembers": [],  
      "DBClusterIdentifier": "custom-endpoint-demo",  
      "Status": "available",  
      "EndpointType": "CUSTOM",  
      "Endpoint": "powers-of-2.cluster-custom-123456789012.ca-central-1.rds.amazonaws.com",  
      "StaticMembers": [  
        "custom-endpoint-demo-04",  
        "custom-endpoint-demo-08",  
        "custom-endpoint-demo-01",  
        "custom-endpoint-demo-02"  
      ],  
      "DBClusterEndpointResourceIdentifier": "cluster-endpoint-W7PE3TLLFNSHXQKFU6J6NV5FHU",  
    }  
  ]}
```

```
    "DBClusterEndpointArn": "arn:aws:rds:ca-central-1:111122223333:cluster-endpoint:powers-of-2",
},
{
    "CustomEndpointType": "ANY",
    "DBClusterEndpointIdentifier": "eight-and-higher",
    "ExcludedMembers": [
        "custom-endpoint-demo-04",
        "custom-endpoint-demo-02",
        "custom-endpoint-demo-07",
        "custom-endpoint-demo-05",
        "custom-endpoint-demo-03",
        "custom-endpoint-demo-06",
        "custom-endpoint-demo-01"
    ],
    "DBClusterIdentifier": "custom-endpoint-demo",
    "Status": "available",
    "EndpointType": "CUSTOM",
    "Endpoint": "eight-and-higher.cluster-custom-123456789012.ca-central-1.rds.amazonaws.com",
    "StaticMembers": [],
    "DBClusterEndpointResourceIdentifier": "cluster-endpoint-W7PE3TLLFNSHYQKFU6J6NV5FHU",
    "DBClusterEndpointArn": "arn:aws:rds:ca-central-1:111122223333:cluster-endpoint:eight-and-higher"
}
]
```

RDS API

To view custom endpoints with the RDS API, run the [DescribeDBClusterEndpoints.html](#) operation.

Editing a custom endpoint

You can edit the properties of a custom endpoint to change which DB instances are associated with the endpoint. You can also change an endpoint between a static list and an exclusion list. If you need more details about these endpoint properties, see [Membership rules for custom endpoints \(p. 37\)](#).

You can't connect to or use a custom endpoint while the changes from an edit action are in progress. It might take some minutes before the endpoint status returns to **Available** and you can connect again.

Console

To edit a custom endpoint with the AWS Management Console, you can select the endpoint on the cluster detail page, or bring up the detail page for the endpoint, and choose the **Edit** action.

RDS > Clusters: [REDACTED]

Edit endpoint:

Endpoint members

Filter database

- | DB instance name

<input checked="" type="checkbox"/>	[REDACTED]
<input checked="" type="checkbox"/>	[REDACTED]

AWS CLI

To edit a custom endpoint with the AWS CLI, run the [modify-db-cluster-endpoint](#) command.

The following commands change the set of DB instances that apply to a custom endpoint and optionally switches between the behavior of a static or exclusion list. The `--static-members` and `--excluded-members` parameters take a space-separated list of DB instance identifiers.

For Linux, macOS, or Unix:

```
aws rds modify-db-cluster-endpoint --db-cluster-endpoint-identifier my-custom-endpoint \
--static-members db-instance-id-1 db-instance-id-2 db-instance-id-3 \
--region region_name

aws rds modify-db-cluster-endpoint --db-cluster-endpoint-identifier my-custom-endpoint \
--excluded-members db-instance-id-4 db-instance-id-5 \
--region region_name
```

For Windows:

```
aws rds modify-db-cluster-endpoint --db-cluster-endpoint-identifier my-custom-endpoint ^
--static-members db-instance-id-1 db-instance-id-2 db-instance-id-3 ^
--region region_name

aws rds modify-db-cluster-endpoint --db-cluster-endpoint-identifier my-custom-endpoint ^
--excluded-members db-instance-id-4 db-instance-id-5 ^
--region region_name
```

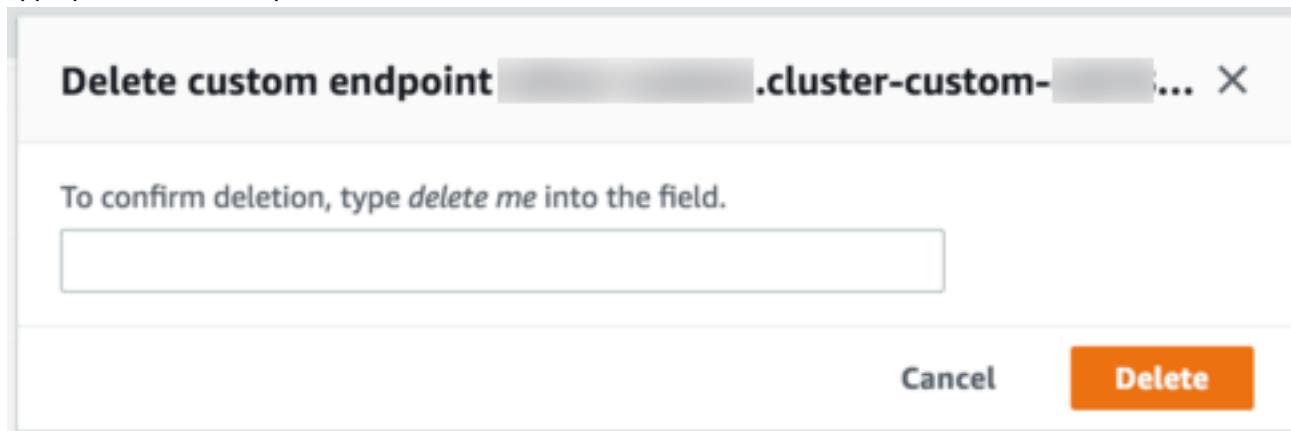
RDS API

To edit a custom endpoint with the RDS API, run the [ModifyDBClusterEndpoint.html](#) operation.

Deleting a custom endpoint

Console

To delete a custom endpoint with the AWS Management Console, go to the cluster detail page, select the appropriate custom endpoint, and select the **Delete** action.



AWS CLI

To delete a custom endpoint with the AWS CLI, run the [delete-db-cluster-endpoint](#) command.

The following command deletes a custom endpoint. You don't need to specify the associated cluster, but you must specify the region.

For Linux, macOS, or Unix:

```
aws rds delete-db-cluster-endpoint --db-cluster-endpoint-identifier custom-end-point-id \  
--region region_name
```

For Windows:

```
aws rds delete-db-cluster-endpoint --db-cluster-endpoint-identifier custom-end-point-id ^  
--region region_name
```

RDS API

To delete a custom endpoint with the RDS API, run the [DeleteDBClusterEndpoint](#) operation.

End-to-end AWS CLI example for custom endpoints

The following tutorial uses AWS CLI examples with Unix shell syntax to show you might define a cluster with several "small" DB instances and a few "big" DB instances, and create custom endpoints to connect to each set of DB instances. To run similar commands on your own system, you should be familiar enough with the basics of working with Aurora clusters and AWS CLI usage to supply your own values for parameters such as region, subnet group, and VPC security group.

This example demonstrates the initial setup steps: creating an Aurora cluster and adding DB instances to it. This is a heterogeneous cluster, meaning not all the DB instances have the same capacity. Most instances use the AWS instance class db.r4.4xlarge, but the last two DB instances use db.r4.16xlarge. Each of these sample `create-db-instance` commands prints its output to the screen and saves a copy of the JSON in a file for later inspection.

```
aws rds create-db-cluster --db-cluster-identifier custom-endpoint-demo --engine aurora \  
--engine-version 5.6.10a --master-username $MASTER_USER --master-user-password  
$MASTER_PW \  
--db-subnet-group-name $SUBNET_GROUP --vpc-security-group-ids $VPC_SECURITY_GROUP \  
--region $REGION  
  
for i in 01 02 03 04 05 06 07 08  
do  
    aws rds create-db-instance --db-instance-identifier custom-endpoint-demo-${i} \  
    --engine aurora --db-cluster-identifier custom-endpoint-demo --db-instance-class  
db.r4.4xlarge \  
    --region $REGION \  
    | tee custom-endpoint-demo-${i}.json  
done  
  
for i in 09 10  
do  
    aws rds create-db-instance --db-instance-identifier custom-endpoint-demo-${i} \  
    --engine aurora --db-cluster-identifier custom-endpoint-demo --db-instance-class  
db.r4.16xlarge \  
    --region $REGION \  
    | tee custom-endpoint-demo-${i}.json  
done
```

The larger instances are reserved for specialized kinds of reporting queries. To make it unlikely for them to be promoted to the primary instance, the following example changes their promotion tier to the lowest priority.

```
for i in 09 10
do
    aws rds modify-db-instance --db-instance-identifier custom-endpoint-demo-${i} \
        --region $REGION --promotion-tier 15
done
```

Suppose that you want to use the two "bigger" instances only for the most resource-intensive queries. To do this, you can first create a custom read-only endpoint. Then you can add a static list of members so that the endpoint connects only to those DB instances. Those instances are already in the lowest promotion tier, making it unlikely either of them will be promoted to the primary instance. If one of them is promoted to the primary instance, it becomes unreachable through this endpoint because we specified the `READER` type instead of the `ANY` type.

The following example demonstrates the create and modify endpoint commands, and sample JSON output showing the initial and modified state of the custom endpoint.

```
$ aws rds create-db-cluster-endpoint --region $REGION \
    --db-cluster-identifier custom-endpoint-demo \
    --db-cluster-endpoint-identifier big-instances --endpoint-type reader
{
    "EndpointType": "CUSTOM",
    "Endpoint": "big-instances.cluster-custom-123456789012.ca-central-1.rds.amazonaws.com",
    "DBClusterEndpointIdentifier": "big-instances",
    "DBClusterIdentifier": "custom-endpoint-demo",
    "StaticMembers": [],
    "DBClusterEndpointResourceIdentifier": "cluster-endpoint-W7PE3TLLFNSHXQKFU6J6NV5FHU",
    "ExcludedMembers": [],
    "CustomEndpointType": "READER",
    "Status": "creating",
    "DBClusterEndpointArn": "arn:aws:rds:ca-central-1:111122223333:cluster-endpoint:big-
instances"
}

$ aws rds modify-db-cluster-endpoint --db-cluster-endpoint-identifier big-instances \
    --static-members custom-endpoint-demo-09 custom-endpoint-demo-10 --region $REGION
{
    "EndpointType": "CUSTOM",
    "ExcludedMembers": [],
    "DBClusterEndpointIdentifier": "big-instances",
    "DBClusterEndpointResourceIdentifier": "cluster-endpoint-W7PE3TLLFNSHXQKFU6J6NV5FHU",
    "CustomEndpointType": "READER",
    "DBClusterEndpointArn": "arn:aws:rds:ca-central-1:111122223333:cluster-endpoint:big-
instances",
    "StaticMembers": [
        "custom-endpoint-demo-10",
        "custom-endpoint-demo-09"
    ],
    "Status": "modifying",
    "Endpoint": "big-instances.cluster-custom-123456789012.ca-central-1.rds.amazonaws.com",
    "DBClusterIdentifier": "custom-endpoint-demo"
}
```

The default `READER` endpoint for the cluster can connect to either the "small" or "big" DB instances, making it impractical to predict query performance and scalability when the cluster becomes busy. To divide the workload cleanly between the sets of DB instances, you can ignore the default `READER` endpoint and create a second custom endpoint that connects to all other DB instances. The following example does so by creating a custom endpoint and then adding an exclusion list. Any other DB instances you add to the cluster later will be added to this endpoint automatically. The `ANY` type means that this endpoint is associated with eight instances in total: the primary instance and another seven Aurora Replicas. If the example used the `READER` type, the custom endpoint would only be associated with the seven Aurora Replicas.

```
$ aws rds create-db-cluster-endpoint --region $REGION --db-cluster-identifier custom-endpoint-demo \
    --db-cluster-endpoint-identifier small-instances --endpoint-type any
{
    "Status": "creating",
    "DBClusterEndpointIdentifier": "small-instances",
    "CustomEndpointType": "ANY",
    "EndpointType": "CUSTOM",
    "Endpoint": "small-instances.cluster-custom-123456789012.ca-central-1.rds.amazonaws.com",
    "StaticMembers": [],
    "ExcludedMembers": [],
    "DBClusterIdentifier": "custom-endpoint-demo",
    "DBClusterEndpointArn": "arn:aws:rds:ca-central-1:111122223333:cluster-endpoint:small-instances",
    "DBClusterEndpointResourceIdentifier": "cluster-endpoint-6RDDXQOC3AKKZT2PRD7ST37BMY"
}

$ aws rds modify-db-cluster-endpoint --db-cluster-endpoint-identifier small-instances \
    --excluded-members custom-endpoint-demo-09 custom-endpoint-demo-10 --region $REGION
{
    "DBClusterEndpointIdentifier": "small-instances",
    "DBClusterEndpointArn": "arn:aws:rds:ca-central-1:111122223333:cluster-endpoint:small-instances",
    "DBClusterEndpointResourceIdentifier": "cluster-endpoint-6RDDXQOC3AKKZT2PRD7ST37BMY",
    "CustomEndpointType": "ANY",
    "Endpoint": "small-instances.cluster-custom-123456789012.ca-central-1.rds.amazonaws.com",
    "EndpointType": "CUSTOM",
    "ExcludedMembers": [
        "custom-endpoint-demo-09",
        "custom-endpoint-demo-10"
    ],
    "StaticMembers": [],
    "DBClusterIdentifier": "custom-endpoint-demo",
    "Status": "modifying"
}
```

The following example checks the state of the endpoints for this cluster. The cluster still has its original cluster endpoint, with `EndPointType` of `WRITER`, which you would still use for administration, ETL, and other write operations. It still has its original `READER` endpoint, which you wouldn't use because each connection to it might be directed to a "small" or "big" DB instance. The custom endpoints make this behavior predictable, with connections guaranteed to use one of the "small" or "big" DB instances based on the endpoint you specify.

```
$ aws rds describe-db-cluster-endpoints --region $REGION
{
    "DBClusterEndpoints": [
        {
            "EndpointType": "WRITER",
            "Endpoint": "custom-endpoint-demo.cluster-123456789012.ca-central-1.rds.amazonaws.com",
            "Status": "available",
            "DBClusterIdentifier": "custom-endpoint-demo"
        },
        {
            "EndpointType": "READER",
            "Endpoint": "custom-endpoint-demo.cluster-ro-123456789012.ca-central-1.rds.amazonaws.com",
            "Status": "available",
            "DBClusterIdentifier": "custom-endpoint-demo"
        }
    ]
}
```

```

        "Endpoint": "small-instances.cluster-custom-123456789012.ca-
central-1.rds.amazonaws.com",
        "CustomEndpointType": "ANY",
        "DBClusterEndpointArn": "arn:aws:rds:ca-central-1:111122223333:cluster-
endpoint:small-instances",
        "ExcludedMembers": [
            "custom-endpoint-demo-09",
            "custom-endpoint-demo-10"
        ],
        "DBClusterEndpointResourceIdentifier": "cluster-
endpoint-6RDDXQOC3AKKZT2PRD7ST37BMY",
        "DBClusterIdentifier": "custom-endpoint-demo",
        "StaticMembers": [],
        "EndpointType": "CUSTOM",
        "DBClusterEndpointIdentifier": "small-instances",
        "Status": "modifying"
    },
    {
        "Endpoint": "big-instances.cluster-custom-123456789012.ca-
central-1.rds.amazonaws.com",
        "CustomEndpointType": "READER",
        "DBClusterEndpointArn": "arn:aws:rds:ca-central-1:111122223333:cluster-
endpoint:big-instances",
        "ExcludedMembers": [],
        "DBClusterEndpointResourceIdentifier": "cluster-endpoint-
W7PE3TLLFNSHXQKFU6J6NV5FHU",
        "DBClusterIdentifier": "custom-endpoint-demo",
        "StaticMembers": [
            "custom-endpoint-demo-10",
            "custom-endpoint-demo-09"
        ],
        "EndpointType": "CUSTOM",
        "DBClusterEndpointIdentifier": "big-instances",
        "Status": "available"
    }
]
}

```

The final examples demonstrate how successive database connections to the custom endpoints connect to the various DB instances in the Aurora cluster. The `small-instances` endpoint always connects to the `db.r4.4xlarge` DB instances, which are the lower-numbered hosts in this cluster.

```

$ mysql -h small-instances.cluster-custom-123456789012.ca-central-1.rds.amazonaws.com -u
$MYUSER -p
mysql> select @@aurora_server_id;
+-----+
| @@aurora_server_id      |
+-----+
| custom-endpoint-demo-02 |
+-----+

$ mysql -h small-instances.cluster-custom-123456789012.ca-central-1.rds.amazonaws.com -u
$MYUSER -p
mysql> select @@aurora_server_id;
+-----+
| @@aurora_server_id      |
+-----+
| custom-endpoint-demo-07 |
+-----+

$ mysql -h small-instances.cluster-custom-123456789012.ca-central-1.rds.amazonaws.com -u
$MYUSER -p
mysql> select @@aurora_server_id;
+-----+

```

```
| @@aurora_server_id      |
+-----+
| custom-endpoint-demo-01 |
+-----+
```

The `big-instances` endpoint always connects to the `db.r4.16xlarge` DB instances, which are the two highest-numbered hosts in this cluster.

```
$ mysql -h big-instances.cluster-custom-123456789012.ca-central-1.rds.amazonaws.com -u
$MYUSER -p
mysql> select @@aurora_server_id;
+-----+
| @@aurora_server_id      |
+-----+
| custom-endpoint-demo-10 |
+-----+

$ mysql -h big-instances.cluster-custom-123456789012.ca-central-1.rds.amazonaws.com -u
$MYUSER -p
mysql> select @@aurora_server_id;
+-----+
| @@aurora_server_id      |
+-----+
| custom-endpoint-demo-09 |
+-----+
```

Using the instance endpoints

Each DB instance in an Aurora cluster has its own built-in instance endpoint, whose name and other attributes are managed by Aurora. You can't create, delete, or modify this kind of endpoint. You might be familiar with instance endpoints if you use Amazon RDS. However, with Aurora you typically use the writer and reader endpoints more often than the instance endpoints.

In day-to-day Aurora operations, the main way that you use instance endpoints is to diagnose capacity or performance issues that affect one specific instance in an Aurora cluster. While connected to a specific instance, you can examine its status variables, metrics, and so on. Doing this can help you determine what's happening for that instance that's different from what's happening for other instances in the cluster.

In advanced use cases, you might configure some DB instances differently than others. In this case, use the instance endpoint to connect directly to an instance that is smaller, larger, or otherwise has different characteristics than the others. Also, set up failover priority so that this special DB instance is the last choice to take over as the primary instance. We recommend that you use custom endpoints instead of the instance endpoint in such cases. Doing so simplifies connection management and high availability as you add more DB instances to your cluster.

How Aurora endpoints work with high availability

For clusters where high availability is important, use the writer endpoint for read/write or general-purpose connections and the reader endpoint for read-only connections. The writer and reader endpoints manage DB instance failover better than instance endpoints do. Unlike the instance endpoints, the writer and reader endpoints automatically change which DB instance they connect to if a DB instance in your cluster becomes unavailable.

If the primary DB instance of a DB cluster fails, Aurora automatically fails over to a new primary DB instance. It does so by either promoting an existing Aurora Replica to a new primary DB instance or creating a new primary DB instance. If a failover occurs, you can use the writer endpoint to reconnect to the newly promoted or created primary DB instance, or use the reader endpoint to reconnect to one of

the Aurora Replicas in the DB cluster. During a failover, the reader endpoint might direct connections to the new primary DB instance of a DB cluster for a short time after an Aurora Replica is promoted to the new primary DB instance.

If you design your own application logic to manage connections to instance endpoints, you can manually or programmatically discover the resulting set of available DB instances in the DB cluster. You can then confirm their instance classes after failover and connect to an appropriate instance endpoint.

For more information about failovers, see [Fault tolerance for an Aurora DB cluster \(p. 69\)](#).

Aurora DB instance classes

The DB instance class determines the computation and memory capacity of an Aurora DB instance. The DB instance class you need depends on your processing power and memory requirements.

For more information about instance class pricing, see [Amazon RDS pricing](#).

Topics

- [DB instance class types \(p. 54\)](#)
- [Supported DB engines for DB instance classes \(p. 54\)](#)
- [Determining DB instance class support in AWS Regions \(p. 59\)](#)
- [Hardware specifications for DB instance classes for Aurora \(p. 62\)](#)

DB instance class types

Amazon Aurora supports two types of instance classes: memory optimized and burstable performance. For more information about Amazon EC2 instance types, see [Instance types](#) in the Amazon EC2 documentation.

The following are the memory optimized DB instance classes available:

- **db.x2g** – Instance classes optimized for memory-intensive applications and powered by AWS Graviton2 processors. These offer low cost per GiB of memory.
- **db.r6g** – Instance classes powered by AWS Graviton2 processors. These are ideal for running memory-intensive workloads in open-source databases such as MySQL and PostgreSQL.
- **db.r5** – Latest generation instance classes optimized for memory-intensive applications. These offer improved networking performance. They are powered by the AWS Nitro System, a combination of dedicated hardware and lightweight hypervisor.
- **db.r3** – Instance classes that provide memory optimization.

The following are the burstable performance DB instance classes available:

- **db.t4g** – Newest-generation burstable instance classes powered by Arm-based AWS Graviton2 processors. These deliver better price performance than previous-generation burstable performance DB instance classes for a broad set of burstable workloads. T4g instances are configured for Unlimited mode, which means that they can burst beyond the baseline over a 24-hour window for an additional charge. We recommend using these instance classes only for development and test servers, or other nonproduction servers.
- **db.t3** – Next generation instance classes that provide a baseline performance level, with the ability to burst to full CPU usage. T3 instances are configured for Unlimited mode. These instance classes provide more computing capacity than the previous db.t2 instance classes. They are powered by the AWS Nitro System, a combination of dedicated hardware and lightweight hypervisor. We recommend using these instance classes only for development and test servers, or other nonproduction servers.
- **db.t2** – Instance classes that provide a baseline performance level, with the ability to burst to full CPU usage. T2 instances can be configured for Unlimited mode. We recommend using these instance classes only for development and test servers, or other nonproduction servers.

For DB instance class hardware specifications, see [Hardware specifications for DB instance classes for Aurora \(p. 62\)](#).

Supported DB engines for DB instance classes

The following are DB engine considerations for DB instance classes:

- **Aurora support for db.x2g**

- Aurora MySQL versions 2.09.2 and higher, 2.10.0 and higher, and 3.01.0 and higher support the db.x2g instance classes.
- Aurora PostgreSQL versions 11.9 and higher, 12.4 and higher, and 13.3 and higher support the db.x2g instance classes.

- **Aurora support for db.r6g**

- Aurora MySQL versions 2.09.2 and higher and 2.10.0 and higher support the db.r6g instance classes.
- Aurora PostgreSQL versions 13.3, 12.4 and higher and versions 11.9 and higher support the db.r6g instance classes.

- **Aurora support for db.t4g**

- Aurora MySQL versions 2.09.2 and higher, 2.10.0 and higher, and 3.01.0 and higher support the db.t4g instance classes, specifically db.t4g.large and db.t4g.medium.
- Aurora PostgreSQL versions 11.9 and higher, 12.4 and higher, and 13.3 and higher support the db.t4g instance classes, specifically db.t4g.large and db.t4g.medium.

- **Aurora support for db.t3**

- Aurora MySQL supports the db.t3.medium and db.t3.small instance classes for version 1.15 and higher, and all 2.x versions. Aurora MySQL supports the db.t3.large class in version 2.10 and higher.
- Aurora MySQL version 3 includes some changes to instance class support.
 - With Aurora MySQL version 3, you can't use db.r3, db.r4, or db.t2 instance classes.
 - With Aurora MySQL version 3, you can't use the db.t3.small instance class.

The smallest instance classes that you can use with version 3 are t3.medium and t4g.medium.

- For Aurora MySQL db.r5, db.r4, and db.t3 DB instance classes, no instances in the cluster can have pending instance-level system updates. To see pending system updates, use the following AWS Command Line Interface (AWS CLI) command.

```
aws rds describe-pending-maintenance-actions
```

- Aurora PostgreSQL version 13.3 supports db.t3 instance classes.

In the following table, you can find details about supported Amazon Aurora DB instance classes for the Aurora DB engines.

Instance class	Aurora MySQL	Aurora PostgreSQL
db.x2g – memory optimized instance classes powered by AWS Graviton2 processors		
db.x2g.16xlarge	2.09.2 and higher, 2.10.0 and higher, 3.01.0 and higher	13.3, 12.4 and higher, 11.9 and higher
db.x2g.12xlarge	2.09.2 and higher, 2.10.0 and higher, 3.01.0 and higher	13.3, 12.4 and higher, 11.9 and higher
db.x2g.8xlarge	2.09.2 and higher, 2.10.0 and higher, 3.01.0 and higher	13.3, 12.4 and higher, 11.9 and higher

Instance class	Aurora MySQL	Aurora PostgreSQL
db.x2g.4xlarge	2.09.2 and higher, 2.10.0 and higher, 3.01.0 and higher	13.3, 12.4 and higher, 11.9 and higher
db.x2g.2xlarge	2.09.2 and higher, 2.10.0 and higher, 3.01.0 and higher	13.3, 12.4 and higher, 11.9 and higher
db.x2g.xlarge	2.09.2 and higher, 2.10.0 and higher, 3.01.0 and higher	13.3, 12.4 and higher, 11.9 and higher
db.x2g.large	2.09.2 and higher, 2.10.0 and higher, 3.01.0 and higher	13.3, 12.4 and higher, 11.9 and higher
db.r6g – Memory-optimized instance classes powered by AWS Graviton2 processors		
db.r6g.16xlarge	2.09.2 and higher, 2.10.0 and higher, 3.01.0 and higher	13.3, 12.4 and higher, 11.9 and higher
db.r6g.12xlarge	2.09.2 and higher, 2.10.0 and higher, 3.01.0 and higher	13.3, 12.4 and higher, 11.9 and higher
db.r6g.8xlarge	2.09.2 and higher, 2.10.0 and higher, 3.01.0 and higher	13.3, 12.4 and higher, 11.9 and higher
db.r6g.4xlarge	2.09.2 and higher, 2.10.0 and higher, 3.01.0 and higher	13.3, 12.4 and higher, 11.9 and higher
db.r6g.2xlarge	2.09.2 and higher, 2.10.0 and higher, 3.01.0 and higher	13.3, 12.4 and higher, 11.9 and higher
db.r6g.xlarge	2.09.2 and higher, 2.10.0 and higher, 3.01.0 and higher	13.3, 12.4 and higher, 11.9 and higher
db.r6g.large	2.09.2 and higher, 2.10.0 and higher, 3.01.0 and higher	13.3, 12.4 and higher, 11.9 and higher
db.r5 – latest generation memory optimized instance classes		
db.r5.24xlarge	1.22 and higher, 2.06 and higher, 3.01.0 and higher	Yes
db.r5.16xlarge	1.22 and higher, 2.06 and higher, 3.01.0 and higher	Yes
db.r5.12xlarge	1.14.4 and higher, 3.01.0 and higher	Yes

Instance class	Aurora MySQL	Aurora PostgreSQL
db.r5.8xlarge	1.22 and higher, 2.06 and higher, 3.01.0 and higher	Yes
db.r5.4xlarge	1.14.4 and higher, 3.01.0 and higher	Yes
db.r5.2xlarge	1.14.4 and higher, 3.01.0 and higher	Yes
db.r5.xlarge	1.14.4 and higher, 3.01.0 and higher	Yes
db.r5.large	1.14.4 and higher, 3.01.0 and higher	Yes
db.r4 – memory optimized instance classes		
db.r4.16xlarge	1.14.4 and higher; not supported in 3.01.0 and higher	12.4 and higher, 11.4 and higher, 10.4 and higher, and 9.6.3 and higher
db.r4.8xlarge	1.14.4 and higher; not supported in 3.01.0 and higher	12.4 and higher, 11.4 and higher, 10.4 and higher, and 9.6.3 and higher
db.r4.4xlarge	1.14.4 and higher; not supported in 3.01.0 and higher	12.4 and higher, 11.4 and higher, 10.4 and higher, and 9.6.3 and higher
db.r4.2xlarge	1.14.4 and higher; not supported in 3.01.0 and higher	12.4 and higher, 11.4 and higher, 10.4 and higher, and 9.6.3 and higher
db.r4.xlarge	1.14.4 and higher; not supported in 3.01.0 and higher	12.4 and higher, 11.4 and higher, 10.4 and higher, and 9.6.3 and higher
db.r4.large	1.14.4 and higher; not supported in 3.01.0 and higher	12.4 and higher, 11.4 and higher, 10.4 and higher, and 9.6.3 and higher
db.r3 – memory optimized instance classes		
db.r3.8xlarge	All 1.x and 2.x versions; not supported in 3.01.0 and higher	No
db.r3.4xlarge	All 1.x and 2.x versions; not supported in 3.01.0 and higher	No

Instance class	Aurora MySQL	Aurora PostgreSQL
db.r3.2xlarge	All 1.x and 2.x versions; not supported in 3.01.0 and higher	No
db.r3.xlarge	All 1.x and 2.x versions; not supported in 3.01.0 and higher	No
db.r3.large	All 1.x and 2.x versions; not supported in 3.01.0 and higher	No
db.t4g – newest generation burstable performance instance classes powered by AWS Graviton2 processors		
db.t4g.2xlarge	No	No
db.t4g.xlarge	No	No
db.t4g.large	2.09.2 and higher, 2.10.0 and higher, 3.01.0 and higher	13.3, 12.4 and higher, 11.9 and higher
db.t4g.medium	2.09.2 and higher, 2.10.0 and higher, 3.01.0 and higher	13.3, 12.4 and higher, 11.9 and higher
db.t4g.small	No	No
db.t3 – Next generation burstable performance instance classes		
db.t3.2xlarge	No	No
db.t3.xlarge	No	No
db.t3.large	2.10 and higher, 3.01.0 and higher	13.3, 11.6 and higher, 10.11 and higher
db.t3.medium	1.14.4 and higher, 3.01.0 and higher	13.3, 10.11 and higher
db.t3.small	1.14.4 and higher; not supported in 3.01.0 and higher	No
db.t3.micro	No	No
db.t2 – burstable performance instance classes		
db.t2.medium	All 1.x and 2.x versions; not supported in 3.01.0 and higher	No
db.t2.small	All 1.x and 2.x versions; not supported in 3.01.0 and higher	No

Determining DB instance class support in AWS Regions

To determine the DB instance classes supported by each DB engine in a specific AWS Region, you can use the AWS Management Console, the [Amazon RDS Pricing](#) page, or the `describe-orderable-db-instance-options` AWS CLI command.

Note

When you perform operations with the AWS CLI, such as creating or modifying a DB cluster, it automatically shows the supported DB instance classes for a specific DB engine, DB engine version, and AWS Region.

Contents

- [Using the Amazon RDS pricing page to determine DB instance class support in AWS Regions \(p. 59\)](#)
- [Using the AWS CLI to determine DB instance class support in AWS Regions \(p. 59\)](#)
 - [Listing the DB instance classes that are supported by a specific DB engine version in an AWS Region \(p. 60\)](#)
 - [Listing the DB engine versions that support a specific DB instance class in an AWS Region \(p. 61\)](#)

Using the Amazon RDS pricing page to determine DB instance class support in AWS Regions

You can use the [Amazon RDS Pricing](#) page to determine the DB instance classes supported by each DB engine in a specific AWS Region.

To use the pricing page to determine the DB instance classes supported by each engine in a Region

1. Go to [Amazon RDS Pricing](#).
2. Choose **Amazon Aurora**.
3. In **DB Instances**, open **MySQL-Compatible Edition** or **PostgreSQL-Compatible Edition**.
4. To see the DB instance classes available in an AWS Region, choose the AWS Region in **Region** in the appropriate subsection.

Using the AWS CLI to determine DB instance class support in AWS Regions

You can use the AWS CLI to determine which DB instance classes are supported for specific DB engines and DB engine versions in an AWS Region.

To use the AWS CLI examples in this section, make sure that you enter valid values for the DB engine, DB engine version, DB instance class, and AWS Region. The following table shows the valid DB engine values.

Engine name	Engine value in CLI commands	More information about versions
MySQL 5.6-compatible Aurora	aurora	Database engine updates for Amazon Aurora MySQL version 1 (p. 1196)

Engine name	Engine value in CLI commands	More information about versions
MySQL 5.7-compatible and 8.0-compatible Aurora	aurora-mysql	Database engine updates for Amazon Aurora MySQL version 2 (p. 1108) , Database engine updates for Amazon Aurora MySQL version 3 (p. 1108)
Aurora PostgreSQL	aurora-postgresql	Amazon Aurora PostgreSQL releases and engine versions (p. 1599)

For information about AWS Region names, see [AWS Regions \(p. 11\)](#).

The following examples demonstrate how to determine DB instance class support in an AWS Region using the [describe-orderable-db-instance-options](#) AWS CLI command.

Topics

- [Listing the DB instance classes that are supported by a specific DB engine version in an AWS Region \(p. 60\)](#)
- [Listing the DB engine versions that support a specific DB instance class in an AWS Region \(p. 61\)](#)

[Listing the DB instance classes that are supported by a specific DB engine version in an AWS Region](#)

To list the DB instance classes that are supported by a specific DB engine version in an AWS Region, run the following command.

For Linux, macOS, or Unix:

```
aws rds describe-orderable-db-instance-options --engine engine --engine-version version \
  --query "OrderableDBInstanceOptions[].
{DBInstanceClass:DBInstanceClass,SupportedEngineModes:SupportedEngineModes[0]}" \
  --output table \
  --region region
```

For Windows:

```
aws rds describe-orderable-db-instance-options --engine engine --engine-version version ^
  --query "OrderableDBInstanceOptions[].
{DBInstanceClass:DBInstanceClass,SupportedEngineModes:SupportedEngineModes[0]}" ^
  --output table ^
  --region region
```

The output also shows the engine modes that are supported for each DB instance class.

For example, the following command lists the supported DB instance classes for version 12.4 of the Aurora PostgreSQL DB engine in US East (N. Virginia).

For Linux, macOS, or Unix:

```
aws rds describe-orderable-db-instance-options --engine aurora-postgresql --engine-version
  12.4 \
  --query "OrderableDBInstanceOptions[].
{DBInstanceClass:DBInstanceClass,SupportedEngineModes:SupportedEngineModes[0]}" \
  --output table \
```

```
--region us-east-1
```

For Windows:

```
aws rds describe-orderable-db-instance-options --engine aurora-postgresql --engine-version 12.4 ^
--query "OrderableDBInstanceOptions[].
{DBInstanceClass:DBInstanceClass,SupportedEngineModes:SupportedEngineModes[0]}" ^
--output table ^
--region us-east-1
```

[Listing the DB engine versions that support a specific DB instance class in an AWS Region](#)

To list the DB engine versions that support a specific DB instance class in an AWS Region, run the following command.

For Linux, macOS, or Unix:

```
aws rds describe-orderable-db-instance-options --engine engine --db-instance-class DB_instance_class \
--query "OrderableDBInstanceOptions[].
{EngineVersion:EngineVersion,SupportedEngineModes:SupportedEngineModes[0]}" \
--output table \
--region region
```

For Windows:

```
aws rds describe-orderable-db-instance-options --engine engine --db-instance-class DB_instance_class ^
--query "OrderableDBInstanceOptions[].
{EngineVersion:EngineVersion,SupportedEngineModes:SupportedEngineModes[0]}" ^
--output table ^
--region region
```

The output also shows the engine modes that are supported for each DB engine version.

For example, the following command lists the DB engine versions of the Aurora PostgreSQL DB engine that support the db.r5.large DB instance class in US East (N. Virginia).

For Linux, macOS, or Unix:

```
aws rds describe-orderable-db-instance-options --engine aurora-postgresql --db-instance-class db.r5.large \
--query "OrderableDBInstanceOptions[].
{EngineVersion:EngineVersion,SupportedEngineModes:SupportedEngineModes[0]}" \
--output table \
--region us-east-1
```

For Windows:

```
aws rds describe-orderable-db-instance-options --engine aurora-postgresql --db-instance-class db.r5.large ^
--query "OrderableDBInstanceOptions[].
{EngineVersion:EngineVersion,SupportedEngineModes:SupportedEngineModes[0]}" ^
--output table ^
--region us-east-1
```

Hardware specifications for DB instance classes for Aurora

The following terminology is used to describe hardware specifications for DB instance classes:

vCPU

The number of virtual central processing units (CPUs). A *virtual CPU* is a unit of capacity that you can use to compare DB instance classes. Instead of purchasing or leasing a particular processor to use for several months or years, you are renting capacity by the hour. Our goal is to make a consistent and specific amount of CPU capacity available, within the limits of the actual underlying hardware.

ECU

The relative measure of the integer processing power of an Amazon EC2 instance. To make it easy for developers to compare CPU capacity between different instance classes, we have defined an Amazon EC2 Compute Unit. The amount of CPU that is allocated to a particular instance is expressed in terms of these EC2 Compute Units. One ECU currently provides CPU capacity equivalent to a 1.0–1.2 GHz 2007 Opteron or 2007 Xeon processor.

Memory (GiB)

The RAM, in gibibytes, allocated to the DB instance. There is often a consistent ratio between memory and vCPU. As an example, take the db.r4 instance class, which has a memory to vCPU ratio similar to the db.r5 instance class. However, for most use cases the db.r5 instance class provides better, more consistent performance than the db.r4 instance class.

Max. Bandwidth (Mbps)

The maximum bandwidth in megabits per second. Divide by 8 to get the expected throughput in megabytes per second.

Note

This figure refers to I/O bandwidth for local storage within the DB instance. It doesn't apply to communication with the Aurora cluster volume.

Network Performance

The network speed relative to other DB instance classes.

In the following table, you can find hardware details about the Amazon RDS DB instance classes for Aurora.

For information about Aurora DB engine support for each DB instance class, see [Supported DB engines for DB instance classes \(p. 54\)](#).

Instance class	vCPU	ECU	Memory (GiB)	Max. bandwidth (mbps) of local storage	Network performance
db.x2g – memory-optimized instance classes					
db.x2g.16xlarge	64	—	1024	19,000	25 Gbps
db.x2g.12xlarge	48	—	768	14,250	20 Gbps
db.x2g.8xlarge	32	—	512	9,500	12 Gbps
db.x2g.4xlarge	16	—	256	4,750	Up to 10 Gbps
db.x2g.2xlarge	8	—	128	Up to 4,750	Up to 10 Gbps

Instance class	vCPU	ECU	Memory (GiB)	Max. bandwidth (mbps) of local storage	Network performance
db.x2g.xlarge	4	—	64	Up to 4,750	Up to 10 Gbps
db.x2g.large	2	—	32	Up to 4,750	Up to 10 Gbps
db.r6g – Memory-optimized instance classes powered by AWS Graviton2 processors					
db.r6g.16xlarge	64	—	512	19,000	25 Gbps
db.r6g.12xlarge	48	—	384	13,500	20 Gbps
db.r6g.8xlarge	32	—	256	9,000	12 Gbps
db.r6g.4xlarge	16	—	128	4,750	Up to 10 Gbps
db.r6g.2xlarge	8	—	64	Up to 4,750	Up to 10 Gbps
db.r6g.xlarge	4	—	32	Up to 4,750	Up to 10 Gbps
db.r6g.large	2	—	16	Up to 4,750	Up to 10 Gbps
db.r5 – latest generation memory optimized instance classes					
db.r5.24xlarge	96	347	768	19,000	25 Gbps
db.r5.16xlarge	64	264	512	13,600	20 Gbps
db.r5.12xlarge	48	173	384	9,500	10 Gbps
db.r5.8xlarge	32	132	256	6,800	10 Gbps
db.r5.4xlarge	16	71	128	4,750	Up to 10 Gbps
db.r5.2xlarge	8	38	64	Up to 4,750	Up to 10 Gbps
db.r5.xlarge	4	19	32	Up to 4,750	Up to 10 Gbps
db.r5.large	2	10	16	Up to 4,750	Up to 10 Gbps
db.r4 – memory optimized instance classes					
db.r4.16xlarge	64	195	488	14,000	25 Gbps
db.r4.8xlarge	32	99	244	7,000	10 Gbps
db.r4.4xlarge	16	53	122	3,500	Up to 10 Gbps
db.r4.2xlarge	8	27	61	1,700	Up to 10 Gbps
db.r4.xlarge	4	13.5	30.5	850	Up to 10 Gbps
db.r4.large	2	7	15.25	425	Up to 10 Gbps
db.r3 – memory optimized instance classes					
db.r3.8xlarge	32	104	244	—	10 Gbps
db.r3.4xlarge	16	52	122	2,000	High
db.r3.2xlarge	8	26	61	1,000	High

Instance class	vCPU	ECU	Memory (GiB)	Max. bandwidth (mbps) of local storage	Network performance
db.r3.xlarge	4	13	30.5	500	Moderate
db.r3.large	2	6.5	15.25	—	Moderate
db.t4g – Newest generation burstable performance instance classes					
db.t4g.large	2	—	8	Up to 2,780	Up to 5 Gbps
db.t4g.medium	2	—	4	Up to 2,085	Up to 5 Gbps
db.t3 – Next generation burstable performance instance classes					
db.t3.large	2	Variable	8	Up to 2,048	Up to 5 Gbps
db.t3.medium	2	Variable	4	Up to 1,536	Up to 5 Gbps
db.t3.small	2	Variable	2	Up to 1,536	Up to 5 Gbps
db.t2 – burstable performance instance classes					
db.t2.medium	2	Variable	4	—	Moderate
db.t2.small	1	Variable	2	—	Low

** The r3.8xlarge instance doesn't have dedicated EBS bandwidth and therefore doesn't offer EBS optimization. On this instance, network traffic and Amazon EBS traffic share the same 10-gigabit network interface.

Amazon Aurora storage and reliability

Following, you can learn about the Aurora storage subsystem. Aurora uses a distributed and shared storage architecture that is an important factor in performance, scalability, and reliability for Aurora clusters.

Topics

- [Overview of Aurora storage \(p. 64\)](#)
- [What the cluster volume contains \(p. 65\)](#)
- [How Aurora storage automatically resizes \(p. 65\)](#)
- [How Aurora data storage is billed \(p. 65\)](#)
- [Amazon Aurora reliability \(p. 66\)](#)

Overview of Aurora storage

Aurora data is stored in the *cluster volume*, which is a single, virtual volume that uses solid state drives (SSDs). A cluster volume consists of copies of the data across three Availability Zones in a single AWS Region. Because the data is automatically replicated across Availability Zones, your data is highly durable with less possibility of data loss. This replication also ensures that your database is more available during a failover. It does so because the data copies already exist in the other Availability Zones and continue to serve data requests to the DB instances in your DB cluster. The amount of replication is independent of the number of DB instances in your cluster.

What the cluster volume contains

The Aurora cluster volume contains all your user data, schema objects, and internal metadata such as the system tables and the binary log. For example, Aurora stores all the tables, indexes, binary large objects (BLOBs), stored procedures, and so on for an Aurora cluster in the cluster volume.

The Aurora shared storage architecture makes your data independent from the DB instances in the cluster. For example, you can add a DB instance quickly because Aurora doesn't make a new copy of the table data. Instead, the DB instance connects to the shared volume that already contains all your data. You can remove a DB instance from a cluster without removing any of the underlying data from the cluster. Only when you delete the entire cluster does Aurora remove the data.

How Aurora storage automatically resizes

Aurora cluster volumes automatically grow as the amount of data in your database increases. The maximum size of for an Aurora cluster volume is 128 tebibytes (TiB) or 64 TiB, depending on the DB engine version. For details about the maximum size for a specific version, see [Amazon Aurora size limits \(p. 1813\)](#). This automatic storage scaling is combined with a high-performance and highly distributed storage subsystem. These make Aurora a good choice for your important enterprise data when your main objectives are reliability and high availability.

To display the volume status, see [Displaying volume status for an Aurora MySQL DB cluster \(p. 837\)](#) or [Displaying volume status for an Aurora PostgreSQL DB cluster \(p. 1367\)](#). For ways to balance storage costs against other priorities, [Storage scaling \(p. 396\)](#) describes how to monitor the Amazon Aurora metrics `AuroraVolumeBytesLeftTotal` and `VolumeBytesUsed` in CloudWatch.

When Aurora data is removed, the space allocated for that data is freed. Examples of removing data include dropping or truncating a table. This automatic reduction in storage usage helps you to minimize storage charges.

Note

The storage limits and dynamic resizing behavior discussed here applies to persistent tables and other data stored in the cluster volume. Data for temporary tables is stored in the local DB instance and its maximum size depends on the instance class that you use.

Some storage features, such as the maximum size of a cluster volume and automatic resizing when data is deleted, depend on the Aurora version of your cluster. For more information, see [Storage scaling \(p. 396\)](#). You can also learn how to avoid storage issues and how to monitor the allocated storage and free space in your cluster.

How Aurora data storage is billed

Even though an Aurora cluster volume can grow up to 128 tebibytes (TiB), you are only charged for the space that you use in an Aurora cluster volume. In earlier Aurora versions, the cluster volume could reuse space that was freed up when you deleted data, but the allocated storage space would never decrease. Starting in Aurora MySQL 2.09.0 and 1.23.0, and Aurora PostgreSQL 3.3.0 and 2.6.0, when Aurora data is removed, such as by dropping a table or database, the overall allocated space decreases by a comparable amount. Thus, you can reduce storage charges by deleting tables, indexes, databases, and so on that you no longer need.

Tip

For earlier versions without the dynamic resizing feature, resetting the storage usage for a cluster involved doing a logical dump and restoring to a new cluster. That operation can take a long time for a substantial volume of data. If you encounter this situation, consider upgrading your cluster to a version that supports volume shrinking.

For pricing information about Aurora data storage, see [Amazon RDS for Aurora Pricing](#).

For information about how to minimize storage charges by monitoring storage usage for your cluster, see [Storage scaling \(p. 396\)](#). For pricing information about Aurora data storage, see [Amazon RDS for Aurora pricing](#).

Amazon Aurora reliability

Aurora is designed to be reliable, durable, and fault tolerant. You can architect your Aurora DB cluster to improve availability by doing things such as adding Aurora Replicas and placing them in different Availability Zones, and also Aurora includes several automatic features that make it a reliable database solution.

Topics

- [Storage auto-repair \(p. 66\)](#)
- [Survivable cache warming \(p. 66\)](#)
- [Crash recovery \(p. 66\)](#)

Storage auto-repair

Because Aurora maintains multiple copies of your data in three Availability Zones, the chance of losing data as a result of a disk failure is greatly minimized. Aurora automatically detects failures in the disk volumes that make up the cluster volume. When a segment of a disk volume fails, Aurora immediately repairs the segment. When Aurora repairs the disk segment, it uses the data in the other volumes that make up the cluster volume to ensure that the data in the repaired segment is current. As a result, Aurora avoids data loss and reduces the need to perform a point-in-time restore to recover from a disk failure.

Survivable cache warming

Aurora "warms" the buffer pool cache when a database starts up after it has been shut down or restarted after a failure. That is, Aurora preloads the buffer pool with the pages for known common queries that are stored in an in-memory page cache. This provides a performance gain by bypassing the need for the buffer pool to "warm up" from normal database use.

The Aurora page cache is managed in a separate process from the database, which allows the page cache to survive independently of the database. In the unlikely event of a database failure, the page cache remains in memory, which ensures that the buffer pool is warmed with the most current state when the database restarts.

Crash recovery

Aurora is designed to recover from a crash almost instantaneously and continue to serve your application data without the binary log. Aurora performs crash recovery asynchronously on parallel threads, so that your database is open and available immediately after a crash.

For more information about crash recovery, see [Fault tolerance for an Aurora DB cluster \(p. 69\)](#).

The following are considerations for binary logging and crash recovery on Aurora MySQL:

- Enabling binary logging on Aurora directly affects the recovery time after a crash, because it forces the DB instance to perform binary log recovery.
- The type of binary logging used affects the size and efficiency of logging. For the same amount of database activity, some formats log more information than others in the binary logs. The following settings for the `binlog_format` parameter result in different amounts of log data:
 - `ROW` – The most log data

- **STATEMENT** – The least log data
- **MIXED** – A moderate amount of log data that usually provides the best combination of data integrity and performance

The amount of binary log data affects recovery time. If there is more data logged in the binary logs, the DB instance must process more data during recovery, which increases recovery time.

- Aurora does not need the binary logs to replicate data within a DB cluster or to perform point in time restore (PITR).
- If you don't need the binary log for external replication (or an external binary log stream), we recommend that you set the `binlog_format` parameter to `OFF` to disable binary logging. Doing so reduces recovery time.

For more information about Aurora binary logging and replication, see [Replication with Amazon Aurora \(p. 70\)](#). For more information about the implications of different MySQL replication types, see [Advantages and disadvantages of statement-based and row-based replication](#) in the MySQL documentation.

Amazon Aurora security

Security for Amazon Aurora is managed at three levels:

- To control who can perform Amazon RDS management actions on Aurora DB clusters and DB instances, you use AWS Identity and Access Management (IAM). When you connect to AWS using IAM credentials, your AWS account must have IAM policies that grant the permissions required to perform Amazon RDS management operations. For more information, see [Identity and access management in Amazon Aurora \(p. 1724\)](#).

If you are using IAM to access the Amazon RDS console, you must first log on to the AWS Management Console with your IAM user credentials, and then go to the Amazon RDS console at <https://console.aws.amazon.com/rds>.

- Aurora DB clusters must be created in a virtual private cloud (VPC) based on the Amazon VPC service. To control which devices and Amazon EC2 instances can open connections to the endpoint and port of the DB instance for Aurora DB clusters in a VPC, you use a VPC security group. You can make these endpoint and port connections using Transport Layer Security (TLS)/Secure Sockets Layer (SSL). In addition, firewall rules at your company can control whether devices running at your company can open connections to a DB instance. For more information on VPCs, see [Amazon Virtual Private Cloud VPCs and Amazon Aurora \(p. 1787\)](#).
- To authenticate logins and permissions for an Amazon Aurora DB cluster, you can take either of the following approaches, or a combination of them.
 - You can take the same approach as with a stand-alone DB instance of MySQL or PostgreSQL.

Techniques for authenticating logins and permissions for stand-alone DB instances of MySQL or PostgreSQL, such as using SQL commands or modifying database schema tables, also work with Aurora. For more information, see [Security with Amazon Aurora MySQL \(p. 774\)](#) or [Security with Amazon Aurora PostgreSQL \(p. 1276\)](#).

- You can also use IAM database authentication for Aurora MySQL.

With IAM database authentication, you authenticate to your Aurora MySQL DB cluster by using an IAM user or IAM role and an authentication token. An *authentication token* is a unique value that is generated using the Signature Version 4 signing process. By using IAM database authentication, you can use the same credentials to control access to your AWS resources and your databases. For more information, see [IAM database authentication \(p. 1743\)](#).

For information about configuring security, see [Security in Amazon Aurora \(p. 1706\)](#).

Using SSL with Aurora DB clusters

Amazon Aurora DB clusters support Secure Sockets Layer (SSL) connections from applications using the same process and public key as Amazon RDS DB instances. For more information, see [Security with Amazon Aurora MySQL \(p. 774\)](#), [Security with Amazon Aurora PostgreSQL \(p. 1276\)](#), or [Using TLS/SSL with Aurora Serverless v1 \(p. 150\)](#).

High availability for Amazon Aurora

The Amazon Aurora architecture involves separation of storage and compute. Aurora includes some high availability features that apply to the data in your DB cluster. The data remains safe even if some or all of the DB instances in the cluster become unavailable. Other high availability features apply to the DB instances. These features help to make sure that one or more DB instances are ready to handle database requests from your application.

Topics

- [High availability for Aurora data \(p. 68\)](#)
- [High availability for Aurora DB instances \(p. 68\)](#)
- [High availability across AWS Regions with Aurora global databases \(p. 69\)](#)
- [Fault tolerance for an Aurora DB cluster \(p. 69\)](#)

High availability for Aurora data

Aurora stores copies of the data in a DB cluster across multiple Availability Zones in a single AWS Region. Aurora stores these copies regardless of whether the instances in the DB cluster span multiple Availability Zones. For more information on Aurora, see [Managing an Amazon Aurora DB cluster \(p. 367\)](#).

When data is written to the primary DB instance, Aurora synchronously replicates the data across Availability Zones to six storage nodes associated with your cluster volume. Doing so provides data redundancy, eliminates I/O freezes, and minimizes latency spikes during system backups. Running a DB instance with high availability can enhance availability during planned system maintenance, and help protect your databases against failure and Availability Zone disruption. For more information on Availability Zones, see [Regions and Availability Zones \(p. 11\)](#).

High availability for Aurora DB instances

For a cluster using single-master replication, after you create the primary instance, you can create up to 15 read-only Aurora Replicas. The Aurora Replicas are also known as reader instances.

During day-to-day operations, you can offload some of the work for read-intensive applications by using the reader instances to process `SELECT` queries. When a problem affects the primary instance, one of these reader instances takes over as the primary instance. This mechanism is known as *failover*. Many Aurora features apply to the failover mechanism. For example, Aurora detects database problems and activates the failover mechanism automatically when necessary. Aurora also has features that reduce the time for failover to complete. Doing so minimizes the time that the database is unavailable for writing during a failover.

To use a connection string that stays the same even when a failover promotes a new primary instance, you connect to the cluster endpoint. The *cluster endpoint* always represents the current primary instance in the cluster. For more information about the cluster endpoint, see [Amazon Aurora connection management \(p. 32\)](#).

Tip

Within each AWS Region, Availability Zones represent locations that are distinct from each other to provide isolation in case of outages. We recommend that you distribute the primary instance and reader instances in your DB cluster over multiple Availability Zones to improve the availability of your DB cluster. That way, an issue that affects an entire Availability Zone doesn't cause an outage for your cluster.

You can set up a Multi-AZ cluster by making a simple choice when you create the cluster. The choice is simple whether you use the AWS Management Console, the AWS CLI, or the Amazon RDS API. You can also make an existing Aurora cluster into a Multi-AZ cluster by adding a new reader instance and specifying a different Availability Zone.

High availability across AWS Regions with Aurora global databases

For high availability across multiple AWS Regions, you can set up Aurora global databases. Each Aurora global database spans multiple AWS Regions, enabling low latency global reads and disaster recovery from outages across an AWS Region. Aurora automatically handles replicating all data and updates from the primary AWS Region to each of the secondary Regions. For more information, see [Using Amazon Aurora global databases \(p. 225\)](#).

Fault tolerance for an Aurora DB cluster

An Aurora DB cluster is fault tolerant by design. The cluster volume spans multiple Availability Zones in a single AWS Region, and each Availability Zone contains a copy of the cluster volume data. This functionality means that your DB cluster can tolerate a failure of an Availability Zone without any loss of data and only a brief interruption of service.

If the primary instance in a DB cluster using single-master replication fails, Aurora automatically fails over to a new primary instance in one of two ways:

- By promoting an existing Aurora Replica to the new primary instance
- By creating a new primary instance

If the DB cluster has one or more Aurora Replicas, then an Aurora Replica is promoted to the primary instance during a failure event. A failure event results in a brief interruption, during which read and write operations fail with an exception. However, service is typically restored in less than 120 seconds, and often less than 60 seconds. To increase the availability of your DB cluster, we recommend that you create at least one or more Aurora Replicas in two or more different Availability Zones.

Tip

In Aurora MySQL 2.10 and higher, you can improve availability during a failover by having more than one reader DB instance in a cluster. In Aurora MySQL 2.10 and higher, Aurora restarts only the writer DB instance and the failover target during a failover. Other reader DB instances in the cluster remain available to continue processing queries through connections to the reader endpoint.

You can customize the order in which your Aurora Replicas are promoted to the primary instance after a failure by assigning each replica a priority. Priorities range from 0 for the first priority to 15 for the last priority. If the primary instance fails, Amazon RDS promotes the Aurora Replica with the better priority to the new primary instance. You can modify the priority of an Aurora Replica at any time. Modifying the priority doesn't trigger a failover.

More than one Aurora Replica can share the same priority, resulting in promotion tiers. If two or more Aurora Replicas share the same priority, then Amazon RDS promotes the replica that is largest in size. If two or more Aurora Replicas share the same priority and size, then Amazon RDS promotes an arbitrary replica in the same promotion tier.

If the DB cluster doesn't contain any Aurora Replicas, then the primary instance is recreated in the same AZ during a failure event. A failure event results in an interruption during which read and write operations fail with an exception. Service is restored when the new primary instance is created, which typically takes less than 10 minutes. Promoting an Aurora Replica to the primary instance is much faster than creating a new primary instance.

Suppose that the primary instance in your cluster is unavailable because of an outage that affects an entire AZ. In this case, the way to bring a new primary instance online depends on whether your cluster uses a multi-AZ configuration. If the cluster contains any reader instances in other AZs, Aurora uses the failover mechanism to promote one of those reader instances to be the new primary instance. If your provisioned cluster only contains a single DB instance, or if the primary instance and all reader instances are in the same AZ, you must manually create one or more new DB instances in another AZ. If your cluster uses Aurora Serverless, Aurora automatically creates a new DB instance in another AZ. However, this process involves a host replacement and thus takes longer than a failover.

Note

Amazon Aurora also supports replication with an external MySQL database, or an RDS MySQL DB instance. For more information, see [Replication between Aurora and MySQL or between Aurora and another Aurora DB cluster \(binary log replication\)](#) (p. 932).

Replication with Amazon Aurora

There are several replication options with Aurora. Each Aurora DB cluster has built-in replication between multiple DB instances in the same cluster. You can also set up replication with your Aurora cluster as the source or the target. When you replicate data into or out of an Aurora cluster, you can choose between built-in features such as Aurora global databases or the traditional replication mechanisms for the MySQL or PostgreSQL DB engines. You can choose the appropriate options based on which one provides the right combination of high availability, convenience, and performance for your needs. The following sections explain how and when to choose each technique.

Topics

- [Aurora Replicas](#) (p. 70)
- [Replication with Aurora MySQL](#) (p. 71)
- [Replication with Aurora PostgreSQL](#) (p. 72)

Aurora Replicas

When you create a second, third, and so on DB instance in an Aurora provisioned DB cluster, Aurora automatically sets up replication from the writer DB instance to all the other DB instances. These other DB instances are read-only and are known as Aurora Replicas. We also refer to them as reader instances when discussing the ways that you can combine writer and reader DB instances within a cluster.

Aurora Replicas have two main purposes. You can issue queries to them to scale the read operations for your application. You typically do so by connecting to the reader endpoint of the cluster. That way, Aurora can spread the load for read-only connections across as many Aurora Replicas as you have in the cluster. Aurora Replicas also help to increase availability. If the writer instance in a cluster becomes unavailable, Aurora automatically promotes one of the reader instances to take its place as the new writer.

An Aurora DB cluster can contain up to 15 Aurora Replicas. The Aurora Replicas can be distributed across the Availability Zones that a DB cluster spans within an AWS Region.

The data in your DB cluster has its own high availability and reliability features, independent of the DB instances in the cluster. If you aren't familiar with Aurora storage features, see [Overview of Aurora](#)

storage (p. 64). The DB cluster volume is physically made up of multiple copies of the data for the DB cluster. The primary instance and the Aurora Replicas in the DB cluster all see the data in the cluster volume as a single logical volume.

As a result, all Aurora Replicas return the same data for query results with minimal replica lag. This lag is usually much less than 100 milliseconds after the primary instance has written an update. Replica lag varies depending on the rate of database change. That is, during periods where a large amount of write operations occur for the database, you might see an increase in replica lag.

Aurora Replicas work well for read scaling because they are fully dedicated to read operations on your cluster volume. Write operations are managed by the primary instance. Because the cluster volume is shared among all DB instances in your DB cluster, minimal additional work is required to replicate a copy of the data for each Aurora Replica.

To increase availability, you can use Aurora Replicas as failover targets. That is, if the primary instance fails, an Aurora Replica is promoted to the primary instance. There is a brief interruption during which read and write requests made to the primary instance fail with an exception. When this happens, some of the Aurora Replicas might be rebooted, depending on the DB engine version. For information about the rebooting behavior of different Aurora DB engine versions, see [Rebooting an Amazon Aurora DB cluster or Amazon Aurora DB instance \(p. 451\)](#). Promoting an Aurora Replica this way is much faster than recreating the primary instance. If your Aurora DB cluster doesn't include any Aurora Replicas, then your DB cluster isn't available while your DB instance is recovering from the failure.

For high-availability scenarios, we recommend that you create one or more Aurora Replicas. These should be of the same DB instance class as the primary instance and in different Availability Zones for your Aurora DB cluster. For more information about Aurora Replicas as failover targets, see [Fault tolerance for an Aurora DB cluster \(p. 69\)](#).

When an Aurora Replica is deleted, its instance endpoint is removed immediately, and the Aurora Replica is removed from the reader endpoint. If there are statements running on the Aurora Replica that is being deleted, there is a three minute grace period. Existing statements can finish gracefully during the grace period. When the grace period ends, the Aurora Replica is shut down and deleted.

You can't create an encrypted Aurora Replica for an unencrypted Aurora DB cluster. You can't create an unencrypted Aurora Replica for an encrypted Aurora DB cluster.

Tip

You can use Aurora Replicas within an Aurora cluster as your only form of replication to keep your data highly available. You can also combine the built-in Aurora replication with the other types of replication. Doing so can help to provide an extra level of high availability and geographic distribution of your data.

For details on how to create an Aurora Replica, see [Adding Aurora Replicas to a DB cluster \(p. 392\)](#).

Replication with Aurora MySQL

In addition to Aurora Replicas, you have the following options for replication with Aurora MySQL:

- Aurora MySQL DB clusters in different AWS Regions.
 - You can replicate data across multiple Regions by using an Aurora global database. For details, see [High availability across AWS Regions with Aurora global databases \(p. 69\)](#)
 - You can create an Aurora read replica of an Aurora MySQL DB cluster in a different AWS Region, by using MySQL binary log (binlog) replication. Each cluster can have up to five read replicas created this way, each in a different Region.
- Two Aurora MySQL DB clusters in the same Region, by using MySQL binary log (binlog) replication.
- An RDS for MySQL DB instance as the source of data and an Aurora MySQL DB cluster, by creating an Aurora read replica of an RDS for MySQL DB instance. Typically, you use this approach for migration to Aurora MySQL, rather than for ongoing replication.

For more information about replication with Aurora MySQL, see [Single-master replication with Amazon Aurora MySQL \(p. 918\)](#).

Replication with Aurora PostgreSQL

In addition to Aurora Replicas, you have the following options for replication with Aurora PostgreSQL:

- An Aurora primary DB cluster in one Region and up to five read-only secondary DB clusters in different Regions by using an Aurora global database. Aurora PostgreSQL doesn't support cross-Region Aurora Replicas. However, you can use Aurora global database to scale your Aurora PostgreSQL DB cluster's read capabilities to more than one AWS Region and to meet availability goals. For more information, see [Using Amazon Aurora global databases \(p. 225\)](#).
- Two Aurora PostgreSQL DB clusters in the same Region, by using PostgreSQL's logical replication feature.
- An RDS for PostgreSQL DB instance as the source of data and an Aurora PostgreSQL DB cluster, by creating an Aurora read replica of an RDS for PostgreSQL DB instance. Typically, you use this approach for migration to Aurora PostgreSQL, rather than for ongoing replication.

For more information about replication with Aurora PostgreSQL, see [Replication with Amazon Aurora PostgreSQL \(p. 1431\)](#).

DB instance billing for Aurora

Amazon RDS instances in an Aurora cluster are billed based on the following components:

- DB instance hours (per hour) – Based on the DB instance class of the DB instance (for example, db.t2.small or db.m4.large). Pricing is listed on a per-hour basis, but bills are calculated down to the second and show times in decimal form. RDS usage is billed in one second increments, with a minimum of 10 minutes. For more information, see [Aurora DB instance classes \(p. 54\)](#).
- Storage (per GiB per month) – Storage capacity that you have provisioned to your DB instance. If you scale your provisioned storage capacity within the month, your bill is prorated. For more information, see [Amazon Aurora storage and reliability \(p. 64\)](#).
- I/O requests (per 1 million requests per month) – Total number of storage I/O requests that you have made in a billing cycle.
- Backup storage (per GiB per month) – *Backup storage* is the storage that is associated with automated database backups and any active database snapshots that you have taken. Increasing your backup retention period or taking additional database snapshots increases the backup storage consumed by your database. Per second billing doesn't apply to backup storage (metered in GB-month).

For more information, see [Backing up and restoring an Amazon Aurora DB cluster \(p. 490\)](#).

- Data transfer (per GB) – Data transfer in and out of your DB instance from or to the internet and other AWS Regions.

Amazon RDS provides the following purchasing options to enable you to optimize your costs based on your needs:

- **On-Demand Instances** – Pay by the hour for the DB instance hours that you use. Pricing is listed on a per-hour basis, but bills are calculated down to the second and show times in decimal form. RDS usage is now billed in one second increments, with a minimum of 10 minutes.
- **Reserved Instances** – Reserve a DB instance for a one-year or three-year term and get a significant discount compared to the on-demand DB instance pricing. With Reserved Instance usage, you can

launch, delete, start, or stop multiple instances within an hour and get the Reserved Instance benefit for all of the instances.

For Aurora pricing information, see the [Aurora pricing page](#).

Topics

- [On-Demand DB instances for Aurora \(p. 74\)](#)
- [Reserved DB instances for Aurora \(p. 75\)](#)

On-Demand DB instances for Aurora

Amazon RDS on-demand DB instances are billed based on the class of the DB instance (for example, db.t2.small or db.m4.large). For Amazon RDS pricing information, see the [Amazon RDS product page](#).

Billing starts for a DB instance as soon as the DB instance is available. Pricing is listed on a per-hour basis, but bills are calculated down to the second and show times in decimal form. Amazon RDS usage is billed in one-second increments, with a minimum of 10 minutes. In the case of billable configuration change, such as scaling compute or storage capacity, you're charged a 10-minute minimum. Billing continues until the DB instance terminates, which occurs when you delete the DB instance or if the DB instance fails.

If you no longer want to be charged for your DB instance, you must stop or delete it to avoid being billed for additional DB instance hours. For more information about the DB instance states for which you are billed, see [Viewing DB instance status in an Aurora cluster \(p. 555\)](#).

Stopped DB instances

While your DB instance is stopped, you're charged for provisioned storage, including Provisioned IOPS. You are also charged for backup storage, including storage for manual snapshots and automated backups within your specified retention window. You aren't charged for DB instance hours.

Multi-AZ DB instances

If you specify that your DB instance should be a Multi-AZ deployment, you're billed according to the Multi-AZ pricing posted on the Amazon RDS pricing page.

Reserved DB instances for Aurora

Using reserved DB instances, you can reserve a DB instance for a one- or three-year term. Reserved DB instances provide you with a significant discount compared to on-demand DB instance pricing. Reserved DB instances are not physical instances, but rather a billing discount applied to the use of certain on-demand DB instances in your account. Discounts for reserved DB instances are tied to instance type and AWS Region.

The general process for working with reserved DB instances is: First get information about available reserved DB instance offerings, then purchase a reserved DB instance offering, and finally get information about your existing reserved DB instances.

Overview of reserved DB instances

When you purchase a reserved DB instance in Amazon RDS, you purchase a commitment to getting a discounted rate, on a specific DB instance type, for the duration of the reserved DB instance. To use an Amazon RDS reserved DB instance, you create a new DB instance just like you do for an on-demand instance. The new DB instance that you create must match the specifications of the reserved DB instance. If the specifications of the new DB instance match an existing reserved DB instance for your account, you are billed at the discounted rate offered for the reserved DB instance. Otherwise, the DB instance is billed at an on-demand rate.

You can modify a reserved DB instance. If the modification is within the specifications of the reserved DB instance, part or all of the discount still applies to the modified DB instance. If the modification is outside the specifications, such as changing the instance class, the discount no longer applies. For more information, see [Size-flexible reserved DB instances \(p. 76\)](#).

For more information about reserved DB instances, including pricing, see [Amazon RDS reserved instances](#).

Offering types

Reserved DB instances are available in three varieties—No Upfront, Partial Upfront, and All Upfront—that let you optimize your Amazon RDS costs based on your expected usage.

No Upfront

This option provides access to a reserved DB instance without requiring an upfront payment. Your No Upfront reserved DB instance bills a discounted hourly rate for every hour within the term, regardless of usage, and no upfront payment is required. This option is only available as a one-year reservation.

Partial Upfront

This option requires a part of the reserved DB instance to be paid upfront. The remaining hours in the term are billed at a discounted hourly rate, regardless of usage. This option is the replacement for the previous Heavy Utilization option.

All Upfront

Full payment is made at the start of the term, with no other costs incurred for the remainder of the term regardless of the number of hours used.

If you are using consolidated billing, all the accounts in the organization are treated as one account. This means that all accounts in the organization can receive the hourly cost benefit of reserved DB instances that are purchased by any other account. For more information about consolidated billing, see [Amazon RDS reserved DB instances](#) in the *AWS Billing and Cost Management User Guide*.

Size-flexible reserved DB instances

When you purchase a reserved DB instance, one thing that you specify is the instance class, for example db.m4.large. For more information about instance classes, see [Aurora DB instance classes \(p. 54\)](#).

If you have a DB instance, and you need to scale it to larger capacity, your reserved DB instance is automatically applied to your scaled DB instance. That is, your reserved DB instances are automatically applied across all DB instance class sizes. Size-flexible reserved DB instances are available for DB instances with the same AWS Region and database engine. Size-flexible reserved DB instances can only scale in their instance class type. For example, a reserved DB instance for a db.m4.large can apply to a db.m4.xlarge, but not to a db.m5.large, because db.m4 and db.m5 are different instance class types.

Reserved DB instance benefits also apply for both Multi-AZ and Single-AZ configurations. Flexibility means that you can move freely between configurations within the same DB instance class type. For example, you can move from a Single-AZ deployment running on one large DB instance (four normalized units) to a Multi-AZ deployment running on two small DB instances ($2 \times 2 = 4$ normalized units).

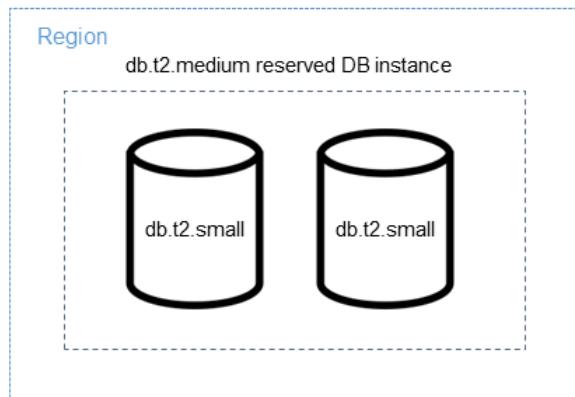
Size-flexible reserved DB instances are available for the following Aurora database engines:

- Aurora MySQL
- Aurora PostgreSQL

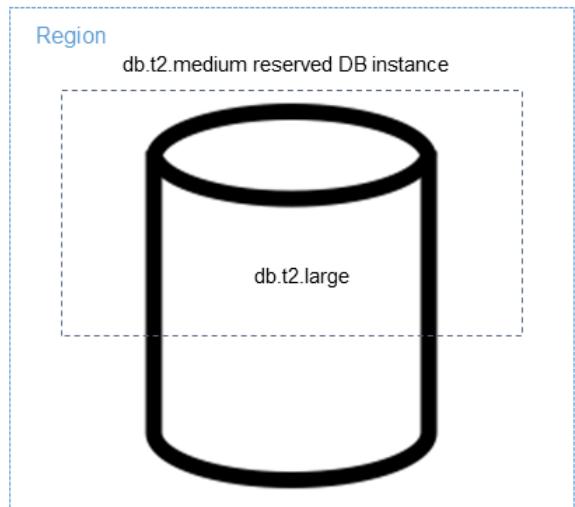
You can compare usage for different reserved DB instance sizes by using normalized units. For example, one unit of usage on two db.m3.large DB instances is equivalent to eight normalized units of usage on one db.m3.small. The following table shows the number of normalized units for each DB instance size.

Instance size	Single-AZ normalized units	Multi-AZ normalized units
micro	0.5	1
small	1	2
medium	2	4
large	4	8
xlarge	8	16
2xlarge	16	32
4xlarge	32	64
8xlarge	64	128
10xlarge	80	160
12xlarge	96	192
16xlarge	128	256
24xlarge	192	384

For example, suppose that you purchase a db.t2.medium reserved DB instance, and you have two running db.t2.small DB instances in your account in the same AWS Region. In this case, the billing benefit is applied in full to both instances.



Alternatively, if you have one db.t2.large instance running in your account in the same AWS Region, the billing benefit is applied to 50 percent of the usage of the DB instance.



Reserved DB instance billing example

The price for a reserved DB instance doesn't include regular costs associated with storage, backups, and I/O. The following example illustrates the total cost per month for a reserved DB instance:

- An Aurora MySQL reserved Single-AZ db.r4.large DB instance class in US East (N. Virginia) at a cost of \$0.19 per hour, or \$138.70 per month
- Aurora storage at a cost of \$0.10 per GiB per month (assume \$45.60 per month for this example)
- Aurora I/O at a cost of \$0.20 per 1 million requests (assume \$20 per month for this example)
- Aurora backup storage at \$0.021 per GiB per month (assume \$30 per month for this example)

Add all of these options ($\$138.70 + \$45.60 + \$20 + \30) with the reserved DB instance, and the total cost per month is \$234.30.

If you chose to use an on-demand DB instance instead of a reserved DB instance, an Aurora MySQL Single-AZ db.r4.large DB instance class in US East (N. Virginia) costs \$0.29 per hour, or \$217.50 per month. So, for an on-demand DB instance, add all of these options ($\$217.50 + \$45.60 + \$20 + \30), and the total cost per month is \$313.10. You save nearly \$79 per month by using the reserved DB instance.

Note

The prices in this example are sample prices and might not match actual prices.
For Aurora pricing information, see the [Aurora pricing page](#).

Deleting a reserved DB instance

The terms for a reserved DB instance involve a one-year or three-year commitment. You can't cancel a reserved DB instance. However, you can delete a DB instance that is covered by a reserved DB instance discount. The process for deleting a DB instance that is covered by a reserved DB instance discount is the same as for any other DB instance.

You're billed for the upfront costs regardless of whether you use the resources.

If you delete a DB instance that is covered by a reserved DB instance discount, you can launch another DB instance with compatible specifications. In this case, you continue to get the discounted rate during the reservation term (one or three years).

Working with reserved DB instances

You can use the AWS Management Console, the AWS CLI, and the RDS API to work with reserved DB instances.

Console

You can use the AWS Management Console to work with reserved DB instances as shown in the following procedures.

To get pricing and information about available reserved DB instance offerings

1. Sign in to the AWS Management Console and open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. In the navigation pane, choose **Reserved instances**.
3. Choose **Purchase Reserved DB Instance**.
4. For **Product description**, choose the DB engine and licensing type.
5. For **DB instance class**, choose the DB instance class.
6. For **Multi-AZ deployment**, choose whether you want a Multi-AZ deployment.

Note

Reserved Amazon Aurora instances always have the **Multi-AZ deployment** option set to **No**. When you create an Amazon Aurora DB cluster from your reserved DB instance, the DB cluster is automatically created as Multi-AZ. You must purchase a reserved DB instance for each DB instance you plan to use, including Aurora Replicas.

7. For **Term**, choose the length of time you want the DB instance reserved.
8. For **Offering type**, choose the offering type.

After you select the offering type, you can see the pricing information.

Important

Choose **Cancel** to avoid purchasing the reserved DB instance and incurring any charges.

After you have information about the available reserved DB instance offerings, you can use the information to purchase an offering as shown in the following procedure.

To purchase a reserved DB instance

1. Sign in to the AWS Management Console and open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.

2. In the navigation pane, choose **Reserved instances**.
3. Choose **Purchase Reserved DB Instance**.
4. For **Product description**, choose the DB engine and licensing type.
5. For **DB instance class**, choose the DB instance class.
6. For **Multi-AZ deployment**, choose whether you want a Multi-AZ deployment.

Note

Reserved Amazon Aurora instances always have the **Multi-AZ deployment** option set to **No**. When you create an Amazon Aurora DB cluster from your reserved DB instance, the DB cluster is automatically created as Multi-AZ. You must purchase a reserved DB instance for each DB instance you plan to use, including Aurora Replicas.

7. For **Term**, choose the length of time you want the DB instance reserved.
8. For **Offering type**, choose the offering type.

After you choose the offering type, you can see the pricing information.

The screenshot shows the 'Purchase Reserved DB Instances' wizard. The first step, 'Options', is displayed. It includes fields for Product description (set to 'aurora-mysql'), DB instance class (set to 'db.r5.xlarge — 4 vCPU, 32 GiB RAM'), Multi AZ deployment (radio button for 'No' selected), Term (set to '1 year'), Offering type (set to 'Partial Upfront'), and a Reserved Id field. The second step, 'Pricing details', is partially visible below it. It shows One-time payment (per instance) and Total one-time payment fields, both with redacted values. It also shows Usage charges (hourly) and a note about additional taxes. The 'Pricing details' step has a 'Cancel' and 'Continue' button at the bottom.

9. (Optional) You can assign your own identifier to the reserved DB instances that you purchase to help you track them. For **Reserved Id**, type an identifier for your reserved DB instance.
10. Choose **Continue**.

The **Purchase Reserved DB Instances** dialog box appears, with a summary of the reserved DB instance attributes that you've selected and the payment due.

The screenshot shows the 'Purchase Reserved DB Instances' dialog box. At the top, there's a breadcrumb navigation: RDS > Reserved instances > Purchase Reserved DB Instances. The main title is 'Purchase Reserved DB Instances'. Below it is a 'Summary of Purchase' section with the following details:

Region	US West (Oregon)
Product Description	aurora-mysql
DB Instance Class	db.r5.xlarge
Offering Type	Partial Upfront
Multi AZ Deployment	No
Term	1 year
Reserved DB Instance	default
Quantity	1
Price Per Instance	[Redacted]
Total Payment Due Now	[Redacted]

At the bottom of the summary section, there's a warning message: **⚠️ Purchasing this Reserved DB Instance will charge [Redacted] to the payment method associated with this Amazon Web Services account. Are you sure you would like to proceed?**

At the very bottom of the dialog box are three buttons: **Cancel**, **Back**, and **Order** (which is highlighted in orange).

11. On the confirmation page, review your reserved DB instance. If the information is correct, choose **Order** to purchase the reserved DB instance.

Alternatively, choose **Back** to edit your reserved DB instance.

After you have purchased reserved DB instances, you can get information about your reserved DB instances as shown in the following procedure.

To get information about reserved DB instances for your AWS account

1. Sign in to the AWS Management Console and open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. In the Navigation pane, choose **Reserved instances**.

The reserved DB instances for your account appear. To see detailed information about a particular reserved DB instance, choose that instance in the list. You can then see detailed information about that instance in the detail pane at the bottom of the console.

AWS CLI

You can use the AWS CLI to work with reserved DB instances as shown in the following examples.

Example of getting available reserved DB instance offerings

To get information about available reserved DB instance offerings, call the AWS CLI command [describe-reserved-db-instances-offerings](#).

```
aws rds describe-reserved-db-instances-offerings
```

This call returns output similar to the following:

OFFERING	OfferingId	Class	Multi-AZ	Duration	Fixed
Price	Usage Price	Description	Offering Type		
OFFERING	438012d3-4052-4cc7-b2e3-8d3372e0e706	db.m1.large	y	1y	1820.00
USD	0.368	mysql	Partial Upfront		
OFFERING	649fd0c8-cf6d-47a0-bfa6-060f8e75e95f	db.m1.small	n	1y	227.50
USD	0.046	mysql	Partial Upfront		
OFFERING	123456cd-ab1c-47a0-bfa6-12345667232f	db.m1.small	n	1y	162.00
USD	0.00	mysql	All Upfront		
Recurring Charges: Amount Currency Frequency					
Recurring Charges: 0.123 USD Hourly					
OFFERING	123456cd-ab1c-37a0-bfa6-12345667232d	db.m1.large	y	1y	700.00
USD	0.00	mysql	All Upfront		
Recurring Charges: Amount Currency Frequency					
Recurring Charges: 1.25 USD Hourly					
OFFERING	123456cd-ab1c-17d0-bfa6-12345667234e	db.m1.xlarge	n	1y	4242.00
USD	2.42	mysql	No Upfront		

After you have information about the available reserved DB instance offerings, you can use the information to purchase an offering.

To purchase a reserved DB instance, use the AWS CLI command [purchase-reserved-db-instances-offering](#) with the following parameters:

- **--reserved-db-instances-offering-id** – The ID of the offering that you want to purchase. See the preceding example to get the offering ID.
- **--reserved-db-instance-id** – You can assign your own identifier to the reserved DB instances that you purchase to help track them.

Example of purchasing a reserved DB instance

The following example purchases the reserved DB instance offering with ID ***649fd0c8-cf6d-47a0-bfa6-060f8e75e95f***, and assigns the identifier of ***MyReservation***.

For Linux, macOS, or Unix:

```
aws rds purchase-reserved-db-instances-offering \
--reserved-db-instances-offering-id 649fd0c8-cf6d-47a0-bfa6-060f8e75e95f \
--reserved-db-instance-id MyReservation
```

For Windows:

```
aws rds purchase-reserved-db-instances-offering ^
--reserved-db-instances-offering-id 649fd0c8-cf6d-47a0-bfa6-060f8e75e95f ^
--reserved-db-instance-id MyReservation
```

The command returns output similar to the following:

RESERVATION	ReservationId	Class	Multi-AZ	Start Time	Duration
Fixed Price	Usage Price	Count	State	Description	Offering Type
RESERVATION	MyReservation	db.m1.small	y	2011-12-19T00:30:23.247Z	1y
455.00	USD	0.092	1	payment-pending	mysql Partial Upfront

After you have purchased reserved DB instances, you can get information about your reserved DB instances.

To get information about reserved DB instances for your AWS account, call the AWS CLI command [describe-reserved-db-instances](#), as shown in the following example.

Example of getting your reserved DB instances

```
aws rds describe-reserved-db-instances
```

The command returns output similar to the following:

RESERVATION	ReservationId	Class	Multi-AZ	Start Time	Duration
Fixed Price	Usage Price	Count	State	Description	Offering Type
RESERVATION	MyReservation	db.m1.small	y	2011-12-09T23:37:44.720Z	1y
455.00	USD	0.092	1	retired	mysql Partial Upfront

RDS API

You can use the RDS API to work with reserved DB instances:

- To get information about available reserved DB instance offerings, call the Amazon RDS API operation [DescribeReservedDBInstancesOfferings](#).
- After you have information about the available reserved DB instance offerings, you can use the information to purchase an offering. Call the [PurchaseReservedDBInstancesOffering](#) RDS API operation with the following parameters:
 - `--reserved-db-instances-offering-id` – The ID of the offering that you want to purchase.
 - `--reserved-db-instance-id` – You can assign your own identifier to the reserved DB instances that you purchase to help track them.
- After you have purchased reserved DB instances, you can get information about your reserved DB instances. Call the [DescribeReservedDBInstances](#) RDS API operation.

Viewing the billing for your reserved DB instances

You can view the billing for your reserved DB instances in the Billing Dashboard in the AWS Management Console.

To view reserved DB instance billing

1. Sign in to the AWS Management Console.
2. From the **account menu** at the upper right, choose **Billing Dashboard**.

3. Choose **Bill Details** at the upper right of the dashboard.
4. Under **AWS Service Charges**, expand **Relational Database Service**.
5. Expand the AWS Region where your reserved DB instances are, for example **US West (Oregon)**.

Your reserved DB instances and their hourly charges for the current month are shown under **Amazon Relational Database Service for Database Engine Reserved Instances**.

Amazon Relational Database Service for MySQL Community Edition Reserved Instances	0.00
MySQL, db.t3.micro reserved instance applied, db.t3.micro instance used	0.00
USD 0.00 hourly fee per MySQL, db.t3.micro instance	0.00

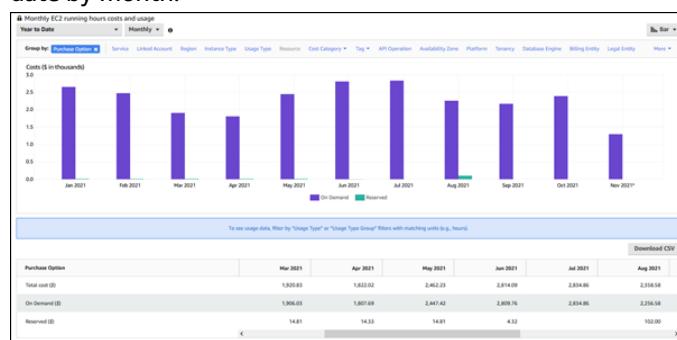
The reserved DB instance in this example was purchased All Upfront, so there are no hourly charges.

6. Choose the **Cost Explorer** (bar graph) icon next to the **Reserved Instances** heading.

The Cost Explorer displays the **Monthly EC2 running hours costs and usage** graph.

7. Clear the **Usage Type Group** filter to the right of the graph.
8. Choose the time period and time unit for which you want to examine usage costs.

The following example shows usage costs for on-demand and reserved DB instances for the year to date by month.



The reserved DB instance costs from January through June 2021 are monthly charges for a Partial Upfront instance, while the cost in August 2021 is a one-time charge for an All Upfront instance.

The reserved instance discount for the Partial Upfront instance expired in June 2021, but the DB instance wasn't deleted. After the expiration date, it was simply charged at the on-demand rate.

Setting up your environment for Amazon Aurora

Before you use Amazon Aurora for the first time, complete the following tasks:

1. [Sign up for AWS \(p. 84\)](#)
2. [Create an IAM user \(p. 84\)](#)
3. [Determine requirements \(p. 86\)](#)
4. [Provide access to the DB cluster in the VPC by creating a security group \(p. 87\)](#)

If you already have an AWS account, know your Aurora requirements, and prefer to use the defaults for IAM and VPC security groups, skip ahead to [Getting started with Amazon Aurora \(p. 89\)](#).

Sign up for AWS

When you sign up for AWS, your AWS account is automatically signed up for all services in AWS, including Amazon RDS. You are charged only for the services that you use.

With Amazon RDS, you pay only for the resources you use. The Amazon RDS DB clusters that you create are live (not running in a sandbox). You incur the standard Amazon RDS usage fees for each DB cluster until you terminate it. For more information about Amazon RDS usage rates, see the [Amazon RDS product page](#). If you are a new AWS customer, you can get started with Amazon RDS for free; for more information, see [AWS free tier](#).

If you have an AWS account already, skip to the next section, [Create an IAM user \(p. 84\)](#).

If you don't have an AWS account, you can use the following procedure to create one.

To create a new AWS account

1. Open <https://portal.aws.amazon.com/billing/signup>.
2. Follow the online instructions.

Part of the sign-up procedure involves receiving a phone call and entering a verification code on the phone keypad.

Note your AWS account number, because you'll need it for the next task.

Create an IAM user

After you create an AWS account and successfully connect to the AWS Management Console, you can create an AWS Identity and Access Management (IAM) user. Instead of signing in with your AWS root account, we recommend that you use an IAM administrative user with Amazon RDS.

One way to do this is to create a new IAM user and grant it administrator permissions. Alternatively, you can add an existing IAM user to an IAM group with Amazon RDS administrative permissions. You can then access AWS from a special URL using the credentials for the IAM user.

If you signed up for AWS but have not created an IAM user for yourself, you can create one using the IAM console.

To create an administrator user for yourself and add the user to an administrators group (console)

1. Sign in to the [IAM console](#) as the account owner by choosing **Root user** and entering your AWS account email address. On the next page, enter your password.

Note

We strongly recommend that you adhere to the best practice of using the **Administrator** IAM user that follows and securely lock away the root user credentials. Sign in as the root user only to perform a few [account and service management tasks](#).

2. In the navigation pane, choose **Users** and then choose **Add user**.
3. For **User name**, enter **Administrator**.
4. Select the check box next to **AWS Management Console access**. Then select **Custom password**, and then enter your new password in the text box.
5. (Optional) By default, AWS requires the new user to create a new password when first signing in. You can clear the check box next to **User must create a new password at next sign-in** to allow the new user to reset their password after they sign in.
6. Choose **Next: Permissions**.
7. Under **Set permissions**, choose **Add user to group**.
8. Choose **Create group**.
9. In the **Create group** dialog box, for **Group name** enter **Administrators**.
10. Choose **Filter policies**, and then select **AWS managed - job function** to filter the table contents.
11. In the policy list, select the check box for **AdministratorAccess**. Then choose **Create group**.

Note

You must activate IAM user and role access to Billing before you can use the **AdministratorAccess** permissions to access the AWS Billing and Cost Management console. To do this, follow the instructions in [step 1 of the tutorial about delegating access to the billing console](#).

12. Back in the list of groups, select the check box for your new group. Choose **Refresh** if necessary to see the group in the list.
13. Choose **Next: Tags**.
14. (Optional) Add metadata to the user by attaching tags as key-value pairs. For more information about using tags in IAM, see [Tagging IAM entities](#) in the *IAM User Guide*.
15. Choose **Next: Review** to see the list of group memberships to be added to the new user. When you are ready to proceed, choose **Create user**.

You can use this same process to create more groups and users and to give your users access to your AWS account resources. To learn about using policies that restrict user permissions to specific AWS resources, see [Access management](#) and [Example policies](#).

To sign in as this new IAM user, sign out of the AWS console, then use the following URL, where *your_aws_account_id* is your AWS account number without the hyphens (for example, if your AWS account number is 1234-5678-9012, your AWS account ID is 123456789012):

`https://your_aws_account_id.signin.aws.amazon.com/console/`

Enter the IAM user name and password that you just created. When you're signed in, the navigation bar displays "`your_user_name @ your_aws_account_id`".

If you don't want the URL for your sign-in page to contain your AWS account ID, you can create an account alias. From the IAM dashboard, choose **Customize** and enter an alias, such as your company name. To sign in after you create an account alias, use the following URL:

```
https://your_account_alias.signin.aws.amazon.com/console/
```

To verify the sign-in link for IAM users for your account, open the IAM console and check under **AWS Account Alias** on the dashboard.

You can also create access keys for your AWS account. These access keys can be used to access AWS through the AWS Command Line Interface (AWS CLI) or through the Amazon RDS API. For more information, see [Programmatic access, Installing, updating, and uninstalling the AWS CLI](#), and the [Amazon RDS API reference](#).

Determine requirements

The basic building block of Aurora is the DB cluster. One or more DB instances can belong to a DB cluster. A DB cluster provides a network address called the *cluster endpoint*. Your applications connect to the cluster endpoint exposed by the DB cluster whenever they need to access the databases created in that DB cluster. The information you specify when you create the DB cluster controls configuration elements such as memory, database engine and version, network configuration, security, and maintenance periods.

Before you create a DB cluster and a security group, you must know your DB cluster and network needs. Here are some important things to consider:

- **Resource requirements** – What are the memory and processor requirements for your application or service? You will use these settings when you determine what DB instance class you will use when you create your DB cluster. For specifications about DB instance classes, see [Aurora DB instance classes \(p. 54\)](#).
- **VPC, subnet, and security group** – Your DB cluster will be in a virtual private cloud (VPC). Security group rules must be configured to connect to a DB cluster. The following list describes the rules for each VPC option:
 - **Default VPC** — If your AWS account has a default VPC in the AWS Region, that VPC is configured to support DB clusters. If you specify the default VPC when you create the DB cluster:
 - Make sure to create a *VPC security group* that authorizes connections from the application or service to the Aurora DB cluster. Use the **Security Group** option on the VPC console or the AWS CLI to create VPC security groups. For information, see [Step 4: Create a VPC security group \(p. 1793\)](#).
 - You must specify the default DB subnet group. If this is the first DB cluster you have created in the AWS Region, Amazon RDS will create the default DB subnet group when it creates the DB cluster.
 - **User-defined VPC** — If you want to specify a user-defined VPC when you create a DB cluster:
 - Make sure to create a *VPC security group* that authorizes connections from the application or service to the Aurora DB cluster. Use the **Security Group** option on the VPC console or the AWS CLI to create VPC security groups. For information, see [Step 4: Create a VPC security group \(p. 1793\)](#).
 - The VPC must meet certain requirements in order to host DB clusters, such as having at least two subnets, each in a separate availability zone. For information, see [Amazon Virtual Private Cloud VPCs and Amazon Aurora \(p. 1787\)](#).
 - You must specify a DB subnet group that defines which subnets in that VPC can be used by the DB cluster. For information, see the DB Subnet Group section in [Working with a DB instance in a VPC \(p. 1788\)](#).

- **High availability:** Do you need failover support? On Aurora, a Multi-AZ deployment creates a primary instance and Aurora Replicas. You can configure the primary instance and Aurora Replicas to be in different Availability Zones for failover support. We recommend Multi-AZ deployments for production workloads to maintain high availability. For development and test purposes, you can use a non-Multi-AZ deployment. For more information, see [High availability for Amazon Aurora \(p. 68\)](#).
- **IAM policies:** Does your AWS account have policies that grant the permissions needed to perform Amazon RDS operations? If you are connecting to AWS using IAM credentials, your IAM account must have IAM policies that grant the permissions required to perform Amazon RDS operations. For more information, see [Identity and access management in Amazon Aurora \(p. 1724\)](#).
- **Open ports:** What TCP/IP port will your database be listening on? The firewall at some companies might block connections to the default port for your database engine. If your company firewall blocks the default port, choose another port for the new DB cluster. Note that once you create a DB cluster that listens on a port you specify, you can change the port by modifying the DB cluster.
- **AWS Region:** What AWS Region do you want your database in? Having the database close in proximity to the application or web service could reduce network latency. For more information, see [Regions and Availability Zones \(p. 11\)](#).

Once you have the information you need to create the security group and the DB cluster, continue to the next step.

Provide access to the DB cluster in the VPC by creating a security group

Your DB cluster will be created in a VPC. Security groups provide access to the DB cluster in the VPC. They act as a firewall for the associated DB cluster, controlling both inbound and outbound traffic at the cluster level. DB clusters are created by default with a firewall and a default security group that prevents access to the DB cluster. You must therefore add rules to a security group that enable you to connect to your DB cluster. Use the network and configuration information you determined in the previous step to create rules to allow access to your DB cluster.

For example, if you have an application that will access a database on your DB cluster in a VPC, you must add a custom TCP rule that specifies the port range and IP addresses that application will use to access the database. If you have an application on an Amazon EC2 cluster, you can use the VPC security group you set up for the Amazon EC2 cluster.

For more information about creating a VPC for use with Aurora, see [How to create a VPC for use with Amazon Aurora \(p. 1793\)](#). For information about common scenarios for accessing a DB instance, see [Scenarios for accessing a DB instance in a VPC \(p. 1800\)](#).

To create a VPC security group

1. Sign in to the AWS Management Console and open the Amazon VPC console at <https://console.aws.amazon.com/vpc>.

Note

Make sure you are in the VPC console, not the RDS console.

2. In the top right corner of the AWS Management Console, choose the AWS Region where you want to create your VPC security group and DB cluster. In the list of Amazon VPC resources for that AWS Region, you should see at least one VPC and several subnets. If you don't, you don't have a default VPC in that AWS Region.
3. In the navigation pane, choose **Security Groups**.
4. Choose **Create security group**.

The **Create security group** page appears.

5. In **Basic details**, enter the **Security group name** and **Description**. For **VPC**, choose the VPC that you want to create your DB cluster in.
6. In **Inbound rules**, choose **Add rule**.
 - a. For **Type**, choose **Custom TCP**.
 - b. For **Port range**, enter the port value to use for your DB cluster.
 - c. For **Source**, choose a security group name or type the IP address range (CIDR value) from where you access the DB cluster. If you choose **My IP**, this allows access to the DB cluster from the IP address detected in your browser.
7. If you need to add more IP addresses or different port ranges, choose **Add rule** and enter the information for the rule.
8. (Optional) In **Outbound rules**, add rules for outbound traffic. By default, all outbound traffic is allowed.
9. Choose **Create security group**.

You can use the VPC security group you just created as the security group for your DB cluster when you create it.

Note

If you use a default VPC, a default subnet group spanning all of the VPC's subnets is created for you. When you create a DB cluster, you can select the default VPC and use **default** for **DB Subnet Group**.

Once you have completed the setup requirements, you can create a DB cluster using your requirements and security group by following the instructions in [Creating an Amazon Aurora DB cluster \(p. 125\)](#). For information about getting started by creating a DB cluster that uses a specific DB engine, see [Getting started with Amazon Aurora \(p. 89\)](#).

Getting started with Amazon Aurora

In this section, you can find out how to create and connect to an Aurora DB cluster using Amazon RDS.

The following procedures are tutorials that demonstrate the basics of getting started with Aurora. Later sections introduce more advanced Aurora concepts and procedures, such as the different kinds of endpoints and how to scale Aurora clusters up and down.

Important

Before you can create or connect to a DB cluster, make sure to complete the tasks in [Setting up your environment for Amazon Aurora \(p. 84\)](#).

Topics

- [Creating a DB cluster and connecting to a database on an Aurora MySQL DB cluster \(p. 89\)](#)
- [Creating a DB cluster and connecting to a database on an Aurora PostgreSQL DB cluster \(p. 96\)](#)
- [Tutorial: Create a web server and an Amazon Aurora DB cluster \(p. 103\)](#)

Creating a DB cluster and connecting to a database on an Aurora MySQL DB cluster

The easiest way to create an Aurora MySQL DB cluster is to use the AWS Management Console. After you create the DB cluster, you can use standard MySQL utilities, such as MySQL Workbench, to connect to a database on the DB cluster.

Important

Before you can create or connect to a DB cluster, you must complete the tasks in [Setting up your environment for Amazon Aurora \(p. 84\)](#).

There's no charge for creating an AWS account. However, by completing this tutorial, you might incur costs for the AWS resources that you use. You can delete these resources after you complete the tutorial if they are no longer needed.

Topics

- [Create an Aurora MySQL DB cluster \(p. 89\)](#)
- [Connect to an instance in a DB cluster \(p. 94\)](#)
- [Delete the sample DB cluster, DB subnet group, and VPC \(p. 96\)](#)

Create an Aurora MySQL DB cluster

Before you create a DB cluster, you must first have a virtual private cloud (VPC) based on the Amazon VPC service and an Amazon RDS DB subnet group. Your VPC must have at least one subnet in each of at least two Availability Zones. You can use the default VPC for your AWS account, or you can create your own VPC. The Amazon RDS console is designed to make it easy for you to create your own VPC for use with Amazon Aurora or use an existing VPC with your Aurora DB cluster.

In some cases, you might want to create a VPC and DB subnet group for use with your Aurora DB cluster yourself, rather than having Amazon RDS create them. If so, follow the instructions in [How to create a](#)

[VPC for use with Amazon Aurora \(p. 1793\)](#). Otherwise, follow the instructions in this topic to create your DB cluster and have Amazon RDS create a VPC and DB subnet group for you.

You can use **Easy create** to create an Aurora MySQL-Compatible Edition DB cluster with the RDS console. With **Easy create**, you specify only the DB engine type, DB instance size, and DB instance identifier. **Easy create** uses the default settings for the other configuration options. When you use **Standard create** instead of **Easy create**, you specify more configuration options when you create a database, including ones for availability, security, backups, and maintenance.

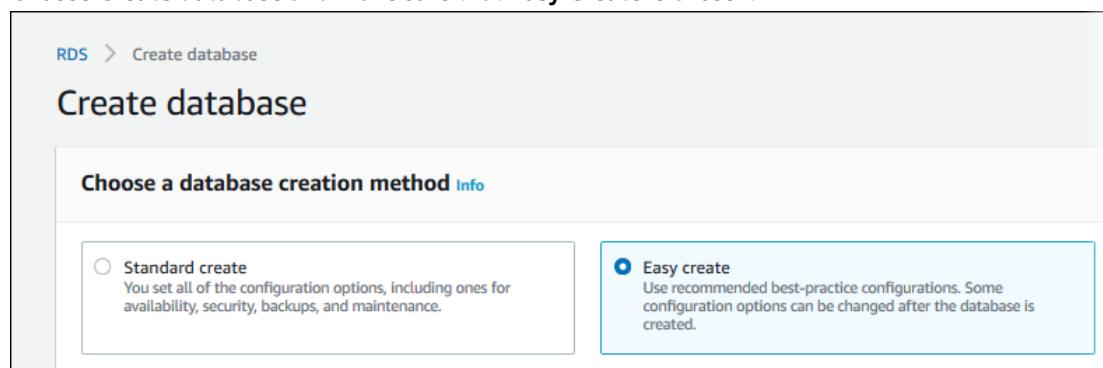
In this tutorial, you use **Easy create** to create an Aurora MySQL-Compatible Edition DB cluster.

Note

For information about creating DB clusters with **Standard create**, see [Creating an Amazon Aurora DB cluster \(p. 125\)](#).

To create an Aurora MySQL DB cluster with Easy create enabled

1. Sign in to the AWS Management Console and open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
 2. In the upper-right corner of the Amazon RDS console, choose the AWS Region in which you want to create the DB cluster.
- Aurora is not available in all AWS Regions. For a list of AWS Regions where Aurora is available, see [Region availability \(p. 12\)](#).
3. In the navigation pane, choose **Databases**.
 4. Choose **Create database** and make sure that **Easy Create** is chosen.



5. For **Engine type**, choose **Amazon Aurora**.
6. For **Edition**, choose **Amazon Aurora with MySQL compatibility**.
7. For **DB instance size**, choose **Dev/Test**.
8. For **DB cluster identifier**, enter a name for the DB cluster, or leave the default name.
9. For **Master username**, enter a name for the user, or leave the default name.

The **Create database** page should look similar to the following image.

Create database

Choose a database creation method [Info](#)

Standard create

You set all of the configuration options, including ones for availability, security, backups, and maintenance.

Easy create

Use recommended best-practice configurations. Some configuration options can be changed after the database is created.

Configuration

Engine type [Info](#)

Amazon Aurora



MySQL



MariaDB



PostgreSQL



Oracle



Microsoft SQL Server



Edition

Amazon Aurora with MySQL compatibility

Amazon Aurora with PostgreSQL compatibility

DB instance size

Production

db.r6g.2xlarge
8 vCPUs
64 GiB RAM

Dev/Test

db.r6g.large
2 vCPUs
16 GiB RAM

DB cluster identifier

Type a name for your DB cluster. The name must be unique across all DB clusters owned by your AWS account in the current AWS Region.

database-1

The DB cluster identifier is case-insensitive, but is stored as all lowercase (as in "mydbcluster"). Constraints: 1 to 60 alphanumeric characters or hyphens. First character must be a letter. Can't contain two consecutive hyphens. Can't end with a hyphen.

10. To use an automatically generated password for the DB cluster, make sure that the **Auto generate a password** box is selected.

To enter your password, clear the **Auto generate a password** box, and then enter the same password in **Master password** and **Confirm password**.

11. (Optional) Open **View default settings for Easy create**.

View default settings for Easy create		
Easy create sets the following configurations to their default values, some of which can be changed later. If you want to change any of these settings now, use Standard Create .		
Configuration	Value	Editable after database is created
Database Features	provisioned	No
Encryption	Enabled	No
VPC	Default VPC (vpc-6594f31c)	No
Option Group	default:aurora-mysql-5-7	No
Subnet Group	default	Yes
Automatic Backups	Enabled	Yes
VPC Security Group	sg-68184619	Yes
Publically Accessible	No	Yes
Database Port	3306	Yes
DB Cluster Identifier	database-1	Yes
DB Instance Identifier	database-1	Yes
DB Engine Version	5.7.mysql_aurora.2.09.2	Yes
DB Parameter Group	default.aurora-mysql5.7	Yes
DB Cluster Parameter Group	default.aurora-mysql5.7	Yes
Performance Insights	Enabled	Yes
Monitoring	Enabled	Yes
Maintenance	Auto Minor Version Upgrade Enabled	Yes
Delete Protection	Not Enabled	Yes

You can examine the default settings used with **Easy create**. The **Editable after database is created** column shows which options you can change after database creation.

- To change settings with **No** in that column, use **Standard create**.
- To change settings with **Yes** in that column, either use **Standard create**, or modify the DB cluster after it is created to change the settings.

The following are important considerations for changing the default settings:

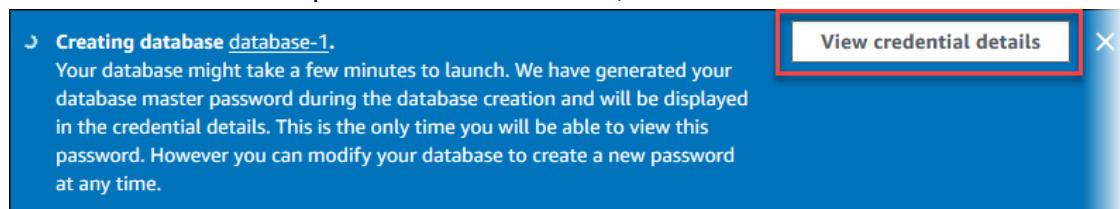
- If you want the DB cluster to use a specific VPC, subnet group, and security group, use **Standard create** to specify these resources. You might have created these resources when you were setting up for Amazon RDS. For more information, see [Setting up your environment for Amazon Aurora \(p. 84\)](#).
- If you want to be able to access the DB cluster from a client outside of its VPC, use **Standard create** to set **Public access** to **Yes**.

If the DB cluster should be private, leave **Public access** set to **No**.

12. Choose **Create database**.

If you chose to use an automatically generated password, the **View credential details** button appears on the **Databases** page.

To view the user name and password for the DB cluster, choose **View credential details**.



To connect to the DB cluster as the master user, use the user name and password that appear.

Important

You can't view the master user password again. If you don't record it, you might have to change it.

If you need to change the master user password after the DB cluster is available, you can modify the DB cluster to do so. For more information about modifying a DB cluster, see [Modifying an Amazon Aurora DB cluster \(p. 372\)](#).

13. For **Databases**, choose the name of the new Aurora MySQL DB cluster.

On the RDS console, the details for new DB cluster appear. The DB cluster and its DB instance have a status of **Creating** until the DB cluster is ready to use. When the state changes to **Available** for both, you can connect to the DB cluster. Depending on the DB instance class and the amount of storage, it can take up to 20 minutes before the new DB cluster is available.

Databases						
		Role	Engine	Region & AZ	Size	Status
<input type="checkbox"/>	<input type="checkbox"/>	DB identifier				Creating
<input type="checkbox"/>	<input type="checkbox"/>	database-1	Regional cluster	Aurora MySQL	us-east-2	1 instance
<input type="checkbox"/>	<input type="checkbox"/>	database-1-instance-1	Reader instance	Aurora MySQL	-	db.r6g.large

Connect to an instance in a DB cluster

After Amazon RDS provisions your DB cluster and creates the primary instance, you can use any standard SQL client application to connect to a database on the DB cluster. In the following procedure, you connect to a database on the Aurora MySQL DB cluster using MySQL monitor commands.

To connect to a database on an Aurora MySQL DB cluster using the MySQL monitor

1. Install a SQL client that you can use to connect to the DB instance.

You can connect to an Aurora MySQL DB cluster by using tools like the MySQL command line utility. For more information on using the MySQL client, see [mysql - the MySQL command-line client](#) in the MySQL documentation. One GUI-based application you can use to connect is MySQL Workbench. For more information, see the [Download MySQL Workbench](#) page.

For more information on using MySQL, see the [MySQL documentation](#). For information about installing MySQL (including the MySQL client), see [Installing and upgrading MySQL](#).

If your DB instance is publicly accessible, you can install the SQL client outside of the VPC. If your DB instance is private, you typically install the SQL client on a resource inside the VPC, such as an Amazon EC2 instance.

2. Make sure that your DB cluster is associated with a security group that provides access to it. For more information, see [Setting up your environment for Amazon Aurora \(p. 84\)](#).

If you didn't specify the appropriate security group when you created the DB cluster, you can modify the DB cluster to change its security group. For more information, see [Modifying an Amazon Aurora DB cluster \(p. 372\)](#).

3. Sign in to the AWS Management Console and open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
4. Choose **Databases** and then choose the DB cluster name to show its details. On the **Connectivity & security** tab, copy the value for the **Endpoint name** of the **Writer instance** endpoint. Also, note the port number for the endpoint.

The screenshot shows the AWS RDS console for a database named 'database-1'. At the top, there are 'Modify' and 'Actions' buttons. Below that is a 'Related' section with a search bar. The main area displays a table of database instances:

DB identifier	Role	Engine	Region & AZ	Size
database-1	Regional cluster	Aurora MySQL	us-east-2	1 instance
database-1-instance-1	Writer instance	Aurora MySQL	us-east-2b	db.r6g.large

Below the table are tabs for Connectivity & security, Monitoring, Logs & events, Configuration, Maintenance & backups, and Tags. The 'Connectivity & security' tab is selected.

In the 'Endpoints' section, there are two entries:

Endpoint name	Status	Type	Port
database-1.cluster-ro-...us-east-2.rds.amazonaws.com	Available	Reader instance	3306
database-1.cluster-...us-east-2.rds.amazonaws.com	Available	Writer instance	3306

The second endpoint is circled in red, and the 'Writer instance' label is also circled in red.

- Enter the following command at a command prompt on a client computer to connect to a database on an Aurora MySQL DB cluster using the MySQL monitor. Use the cluster endpoint to connect to the primary instance, and the master user name that you created previously. (You are prompted for a password.) If you supplied a port value other than 3306, use that for the `-P` parameter instead.

```
PROMPT> mysql -h <cluster endpoint> -P 3306 -u <myusername> -p
```

You should see output similar to the following.

```
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 350
Server version: 5.6.10-log MySQL Community Server (GPL)

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql>
```

For more information about connecting to the DB cluster, see [Connecting to an Amazon Aurora MySQL DB cluster \(p. 281\)](#). If you can't connect to your DB cluster, see [Can't connect to Amazon RDS DB instance \(p. 1814\)](#).

Delete the sample DB cluster, DB subnet group, and VPC

After you have connected to the sample DB cluster that you created, you can delete the DB cluster, DB subnet group, and VPC (if you created a VPC).

To delete a DB cluster

1. Sign in to the AWS Management Console and open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. Choose **Databases** and then choose the DB instance associated with the DB cluster.
3. For **Actions**, choose **Delete**.
4. Choose **Delete**.

After all of the DB instances associated with a DB cluster are deleted, the DB cluster is deleted automatically.

To delete a DB subnet group

1. Sign in to the AWS Management Console and open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. Choose **Subnet groups** and then choose the DB subnet group.
3. Choose **Delete**.
4. Choose **Delete**.

To delete a VPC

1. Sign in to the AWS Management Console and open the Amazon VPC console at <https://console.aws.amazon.com/vpc/>.
2. Choose **Your VPCs** and then choose the VPC that was created for this procedure.
3. For **Actions**, choose **Delete VPC**.
4. Choose **Delete**.

Creating a DB cluster and connecting to a database on an Aurora PostgreSQL DB cluster

The easiest way to create an Aurora PostgreSQL DB cluster is to use the Amazon RDS console. After you create the DB cluster, you can use standard PostgreSQL utilities, such as pgAdmin, to connect to a database on the DB cluster.

Important

Before you can create or connect to a DB cluster, you must complete the tasks in [Setting up your environment for Amazon Aurora \(p. 84\)](#).

There's no charge for creating an AWS account. However, by completing this tutorial, you might incur costs for the AWS resources that you use. You can delete these resources after you complete the tutorial if they are no longer needed.

Topics

- [Create an Aurora PostgreSQL DB cluster \(p. 97\)](#)
- [Connect to an instance in an Aurora PostgreSQL DB cluster \(p. 101\)](#)
- [Delete the sample DB cluster, DB subnet group, and VPC \(p. 102\)](#)

Create an Aurora PostgreSQL DB cluster

Before you create a DB cluster, make sure first to have a virtual private cloud (VPC) based on the Amazon VPC service and an Amazon RDS DB subnet group. Your VPC must have at least one subnet in each of at least two Availability Zones. You can use the default VPC for your AWS account, or you can create your own VPC. The Amazon RDS console is designed to make it easy for you to create your own VPC for use with Amazon Aurora or use an existing VPC with your Aurora DB cluster.

In some cases, you might want to create a VPC and DB subnet group for use with your Amazon Aurora DB cluster yourself, rather than having Amazon RDS create them. If so, follow the instructions in [How to create a VPC for use with Amazon Aurora \(p. 1793\)](#). Otherwise, follow the instructions in this topic to create your DB cluster and have Amazon RDS create a VPC and DB subnet group for you.

You can use **Easy create** to create an Aurora PostgreSQL DB cluster with the AWS Management Console. With **Easy create**, you specify only the DB engine type, DB instance size, and DB instance identifier. **Easy create** uses the default settings for the other configuration options. When you use **Standard create** instead of **Easy create**, you specify more configuration options when you create a database, including ones for availability, security, backups, and maintenance.

In this example, you use **Easy create** to create an Aurora PostgreSQL DB cluster.

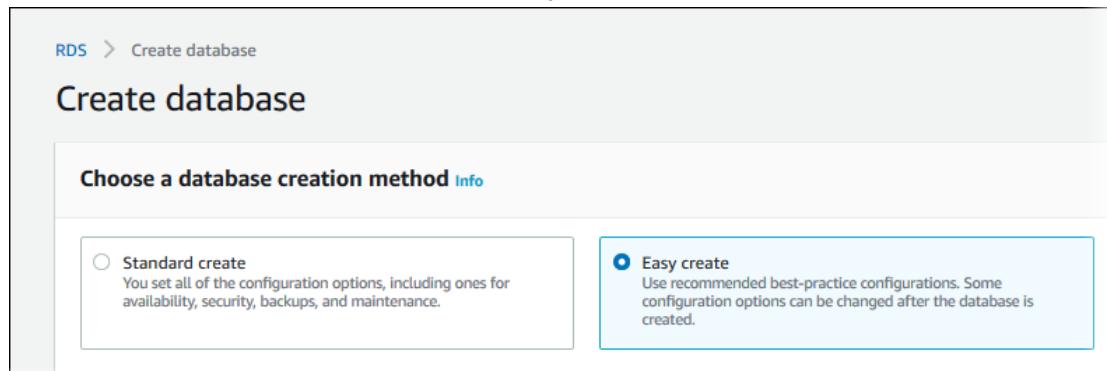
Note

For information about creating DB clusters with **Standard create**, see [Creating an Amazon Aurora DB cluster \(p. 125\)](#).

To create an Aurora PostgreSQL DB cluster with Easy create enabled

1. Sign in to the AWS Management Console and open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. In the upper-right corner of the Amazon RDS console, choose the AWS Region in which you want to create the DB cluster.

Aurora is not available in all AWS Regions. For a list of AWS Regions where Aurora is available, see [Region availability \(p. 12\)](#).
3. In the navigation pane, choose **Databases**.
4. Choose **Create database** and make sure that **Easy Create** is chosen.



5. For **Engine type**, choose **Amazon Aurora**.
6. For **Edition**, choose **Amazon Aurora with PostgreSQL compatibility**.

7. For **DB instance size**, choose **Dev/Test**.
8. For **DB cluster identifier**, enter a name for the DB cluster, or leave the default name.
9. For **Master username**, enter a name for the master user, or leave the default name.

The **Create database** page should look similar to the following image.

Create database

Choose a database creation method [Info](#)

Standard create
You set all of the configuration options, including ones for availability, security, backups, and maintenance.

Easy create
Use recommended best-practice configurations. Some configuration options can be changed after the database is created.

Configuration

Engine type [Info](#)

Amazon Aurora 

MySQL 

MariaDB 

PostgreSQL 

Microsoft SQL Server 

Edition

Amazon Aurora with MySQL compatibility

Amazon Aurora with PostgreSQL compatibility

DB instance size

Production
db.r6g.2xlarge
8 vCPUs
64 GiB RAM

Dev/Test
db.r6g.large
2 vCPUs
16 GiB RAM

DB cluster identifier

Type a name for your DB cluster. The name must be unique across all DB clusters owned by your AWS account in the current AWS Region.

database-1

The DB cluster identifier is case-insensitive, but is stored as all lowercase (as in "mydbcluster"). Constraints: 1 to 60 alphanumeric characters or hyphens. First character must be a letter. Can't contain two consecutive hyphens. Can't end with a hyphen.

10. To use an automatically generated master password for the DB cluster, make sure that the **Auto generate a password** box is selected.

To enter your master password, clear the **Auto generate a password** box, and then enter the same password in **Master password** and **Confirm password**.

11. (Optional) Open **View default settings for Easy create**.

View default settings for Easy create

Easy create sets the following configurations to their default values, some of which can be changed later. If you want to change any of these settings now, use [Standard Create](#).

Configuration	Value	Editable after database is created
Database Features	provisioned	No
Encryption	Enabled	No
VPC	tutorial-vpc (vpc-057ab62a5117ac9fb)	No
Option Group	default:aurora-postgresql-11	No
Subnet Group	tutorial-db-subnet-group	Yes
Automatic Backups	Enabled	Yes
VPC Security Group	sg-0d3bc58e3febca979	Yes
Publicly Accessible	No	Yes
Database Port	5432	Yes
DB Cluster Identifier	database-1	Yes
DB Instance Identifier	database-1	Yes
DB Engine Version	11.9	Yes
DB Parameter Group	default.aurora-postgresql11	Yes
DB Cluster Parameter Group	default.aurora-postgresql11	Yes
Performance Insights	Enabled	Yes
Monitoring	Enabled	Yes
Maintenance	Auto Minor Version Upgrade Enabled	Yes
Delete Protection	Not Enabled	Yes

You can examine the default settings used with **Easy create**. The **Editable after database is created** column shows which options you can change after database creation.

- To change settings with **No** in that column, use **Standard create**.
- To change settings with **Yes** in that column, either use **Standard create**, or modify the DB cluster after it is created to change the settings.

The following are important considerations for changing the default settings:

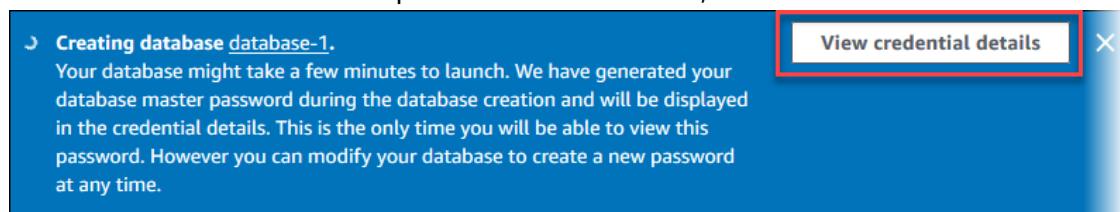
- If you want the DB cluster to use a specific VPC, subnet group, and security group, use **Standard create** to specify these resources. You might have created these resources when you were setting up for Amazon RDS. For more information, see [Setting up your environment for Amazon Aurora \(p. 84\)](#).
- If you want to be able to access the DB cluster from a client outside of its VPC, use **Standard create** to set **Public access** to **Yes**.

If the DB cluster should be private, leave **Public access** set to **No**.

12. Choose **Create database**.

If you chose to use an automatically generated password, the **View credential details** button appears on the **Databases** page.

To view the master user name and password for the DB cluster, choose **View credential details**.



To connect to the DB cluster as the master user, use the user name and password that appear.

Important

You can't view the master user password again. If you don't record it, you might have to change it. If you need to change the master user password after the DB cluster is available, you can modify the DB cluster to do so. For more information about modifying a DB cluster, see [Modifying an Amazon Aurora DB cluster \(p. 372\)](#).

13. For **Databases**, choose the name of the new Aurora PostgreSQL DB cluster.

On the RDS console, the details for new DB cluster appear. The DB cluster and its DB instance have a status of **Creating** until the DB cluster is ready to use. When the state changes to **Available** for both, you can connect to the DB cluster. Depending on the DB instance class and the amount of storage, it can take up to 20 minutes before the new DB cluster is available.

Databases						
	DB identifier	Role	Engine	Region & AZ	Size	Status
Filter databases						
<input type="checkbox"/>	database-1	Regional cluster	Aurora PostgreSQL	us-west-1	1 instance	Creating
<input type="checkbox"/>	database-1-instance-1	Reader instance	Aurora PostgreSQL	-	db.r6g.large	Creating

Connect to an instance in an Aurora PostgreSQL DB cluster

After Amazon RDS provisions your DB cluster and creates the primary instance, you can use any standard SQL client application to connect to a database on the DB cluster.

To connect to a database on an Aurora PostgreSQL DB cluster

1. Make sure that your DB cluster is associated with a security group that provides access to it. For more information, see [Setting up your environment for Amazon Aurora \(p. 84\)](#).

If you didn't specify the appropriate security group when you created the DB cluster, you can modify the DB cluster to change its security group. For more information, see [Modifying an Amazon Aurora DB cluster \(p. 372\)](#).

2. Sign in to the AWS Management Console and open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
3. Choose **Databases** and then choose the DB cluster name to show its details. On the **Connectivity & security** tab, copy the value for the **Endpoint name** of the **Writer instance** endpoint. Also, note the port number for the endpoint.

The screenshot shows the AWS RDS Databases page for a cluster named "database-1". The "Connectivity & security" tab is selected. In the "Endpoints" section, there are two entries:

Endpoint name	Status	Type	Port
database-1.cluster-ro-*.us-west-1.rds.amazonaws.com	Available	Reader instance	5432
database-1.cluster-*.us-west-1.rds.amazonaws.com	Available	Writer instance	5432

The "Writer instance" endpoint is circled in red.

4. If your client computer has PostgreSQL installed, you can use a local instance of psql to connect to a PostgreSQL DB instance. To connect to your PostgreSQL DB instance using psql, provide host information and access credentials.

The following format is used to connect to a PostgreSQL DB instance on Amazon RDS.

```
psql --host=DB_instance_endpoint --port=port --username=master_user_name --password --dbname=database_name
```

For example, the following command connects to a database called mypgdb on a PostgreSQL DB instance called mypostgresql using fictitious credentials.

```
psql --host=database-1.123456789012.us-west-1.rds.amazonaws.com --port=5432 --username=awsuser --password --dbname=postgres
```

For more information about connecting to the DB cluster using the endpoint and port, see [Connecting to an Amazon Aurora PostgreSQL DB cluster \(p. 285\)](#). If you can't connect to your DB cluster, see [Can't connect to Amazon RDS DB instance \(p. 1814\)](#).

Delete the sample DB cluster, DB subnet group, and VPC

After you have connected to the sample DB cluster that you created, you can delete the DB cluster, DB subnet group, and VPC (if you created a VPC).

To delete a DB cluster

1. Sign in to the AWS Management Console and open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. Choose **Databases** and then choose the DB instance associated with the DB cluster.
3. For **Actions**, choose **Delete**.
4. Choose **Delete**.

After all of the DB instances associated with a DB cluster are deleted, the DB cluster is deleted automatically.

To delete a DB subnet group

1. Sign in to the AWS Management Console and open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. Choose **Subnet groups** and then choose the DB subnet group.
3. Choose **Delete**.
4. Choose **Delete**.

To delete a VPC

1. Sign in to the AWS Management Console and open the Amazon VPC console at <https://console.aws.amazon.com/vpc/>.
2. Choose **Your VPCs** and then choose the VPC that was created for this procedure.
3. For **Actions**, choose **Delete VPC**.
4. Choose **Delete**.

Tutorial: Create a web server and an Amazon Aurora DB cluster

This tutorial helps you install an Apache web server with PHP and create a MySQL database. The web server runs on an Amazon EC2 instance using Amazon Linux, and the MySQL database is an Aurora MySQL DB cluster. Both the Amazon EC2 instance and the DB cluster run in a virtual private cloud (VPC) based on the Amazon VPC service.

Important

There's no charge for creating an AWS account. However, by completing this tutorial, you might incur costs for the AWS resources you use. You can delete these resources after you complete the tutorial if they are no longer needed.

Note

This tutorial works with Amazon Linux and might not work for other versions of Linux such as Ubuntu.

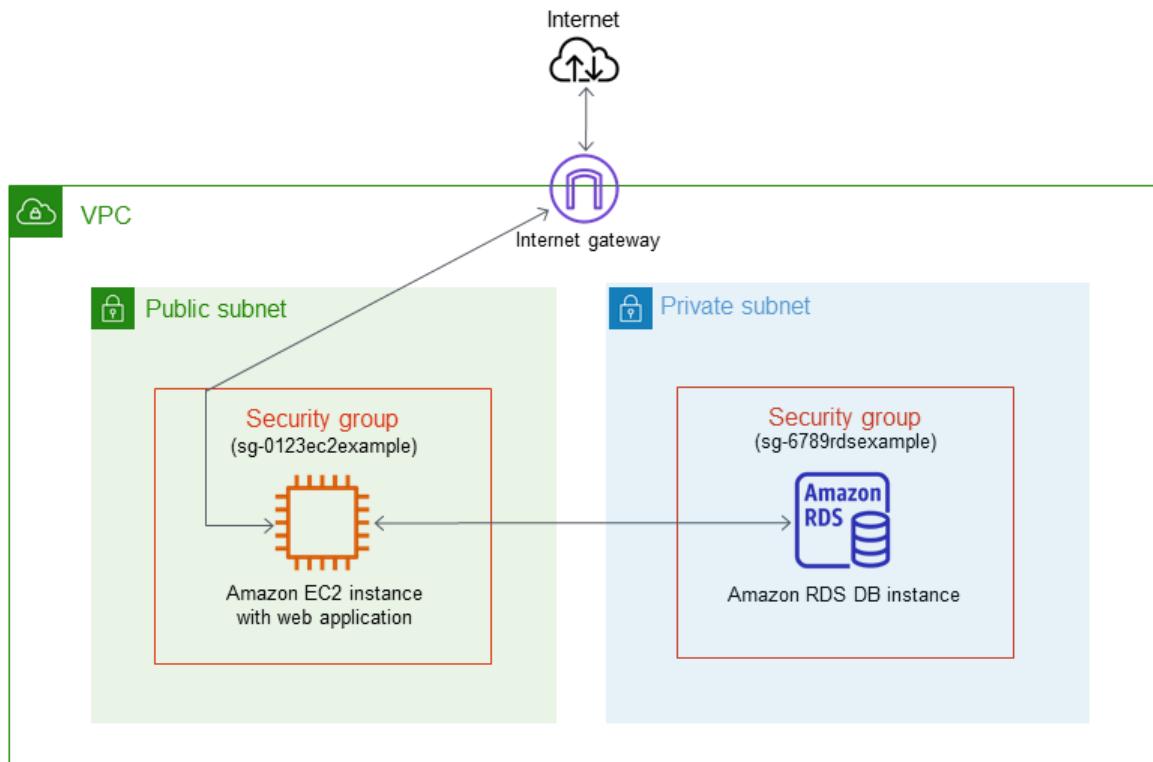
In the tutorial that follows, you specify the VPC, subnets, and security groups when you create the DB cluster. You also specify them when you create the EC2 instance to host your web server. The VPC, subnets, and security groups are required for the DB cluster and the web server to communicate. After the VPC is set up, this tutorial shows you how to create the DB cluster and install the web server. You connect your web server to your DB cluster in the VPC using the DB cluster writer endpoint.

1. Complete the tasks in [Tutorial: Create an Amazon VPC for use with a DB instance \(p. 1805\)](#).

Before you begin this tutorial, make sure that you have a VPC with both public and private subnets, and corresponding security groups. If you don't have these, complete the following tasks in the tutorial:

- a. [Create a VPC with private and public subnets \(p. 1805\)](#)
 - b. [Create additional subnets \(p. 1806\)](#)
 - c. [Create a VPC security group for a public web server \(p. 1807\)](#)
 - d. [Create a VPC security group for a private DB instance \(p. 1808\)](#)
 - e. [Create a DB subnet group \(p. 1808\)](#)
2. [Create an Amazon Aurora DB cluster \(p. 104\)](#)
 3. [Create an EC2 instance and install a web server \(p. 109\)](#)

The following diagram shows the configuration when the tutorial is complete.



Create an Amazon Aurora DB cluster

In this step, you create an Amazon Aurora MySQL DB cluster that maintains the data used by a web application.

Important

Before you begin this step, you must have a VPC with both public and private subnets, and corresponding security groups. If you don't have these, see [Tutorial: Create an Amazon VPC for use with a DB instance \(p. 1805\)](#). Complete the steps in [Create a VPC with private and public subnets \(p. 1805\)](#), [Create additional subnets \(p. 1806\)](#), [Create a VPC security group for a public web server \(p. 1807\)](#), and [Create a VPC security group for a private DB instance \(p. 1808\)](#).

To create an Aurora MySQL DB cluster

1. Sign in to the AWS Management Console and open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. In the upper-right corner of the AWS Management Console, choose the AWS Region where you want to create the DB cluster. This example uses the US West (Oregon) Region.
3. In the navigation pane, choose **Databases**.
4. Choose **Create database**.
5. On the **Create database** page, shown following, make sure that the **Standard create** option is chosen, and then choose **Amazon Aurora**. Keep the default values for **Version** and the other engine options.

RDS > Create database

Create database

Choose a database creation method Info

Standard create
You set all of the configuration options, including ones for availability, security, backups, and maintenance.

Easy create
Use recommended best-practice configurations. Some configuration options can be changed after the database is created.

Engine options

Engine type Info

Amazon Aurora 

MySQL 

MariaDB 

PostgreSQL 

Oracle 

Microsoft SQL Server 

Edition

Amazon Aurora with MySQL compatibility

Amazon Aurora with PostgreSQL compatibility

Capacity type Info

Provisioned
You provision and manage the server instance sizes.

6. In the **Templates** section, choose **Dev/Test**.
7. In the **Settings** section, set these values:
 - **DB cluster identifier** – `tutorial-db-cluster`
 - **Master username** – `tutorial_user`
 - **Auto generate a password** – Disable the option.
 - **Master password** – Choose a password.
 - **Confirm password** – Retype the password.

Settings

DB cluster identifier [Info](#)

Type a name for your DB cluster. The name must be unique across all DB clusters owned by your AWS account in the current AWS Region.

tutorial-db-cluster

The DB cluster identifier is case-insensitive, but is stored as all lowercase (as in "mydbcluster"). Constraints: 1 to 60 alphanumeric characters or hyphens. First character must be a letter. Can't contain two consecutive hyphens. Can't end with a hyphen.

▼ Credentials Settings

Master username [Info](#)

Type a login ID for the master user of your DB instance.

tutorial_user

1 to 16 alphanumeric characters. First character must be a letter

Auto generate a password

Amazon RDS can generate a password for you, or you can specify your own password

Master password [Info](#)

Constraints: At least 8 printable ASCII characters. Can't contain any of the following: / (slash), "(double quote) and @ (at sign).

Confirm password [Info](#)

8. In the **DB instance class** section, enable **Include previous generation classes**, and set these values:
 - **Burstable classes (includes t classes)**
 - **db.t2.small**

DB instance class

DB instance class [Info](#)

Choose a DB instance class that meets your processing power and memory requirements. The DB instance class options below are limited to those supported by the engine you selected above.

Memory optimized classes (includes r classes)

Burstable classes (includes t classes)

db.t2.small

1 vCPUs 2 GiB RAM Not EBS Optimized

i New instance classes are available for specific engine versions. [Info](#)

Include previous generation classes

9. In the **Availability & durability** section, use the default values.
10. In the **Connectivity** section, set these values:

- **Virtual private cloud (VPC)** – Choose an existing VPC with both public and private subnets, such as the `tutorial-vpc` (`vpc-identifier`) created in [Create a VPC with private and public subnets \(p. 1805\)](#)

Note

The VPC must have subnets in different Availability Zones.

- **Subnet group** – The DB subnet group for the VPC, such as the `tutorial-db-subnet-group` created in [Create a DB subnet group \(p. 1808\)](#)

- **Public access** – No

- **VPC security group** – Choose existing

- **Existing VPC security groups** – Choose an existing VPC security group that is configured for private access, such as the `tutorial-db-securitygroup` created in [Create a VPC security group for a private DB instance \(p. 1808\)](#).

Remove other security groups, such as the default security group, by choosing the **X** associated with each.

- **Availability Zone – No preference**

- Open **Additional configuration**, and make sure **Database port** uses the default value **3306**.

Connectivity

Virtual private cloud (VPC) [Info](#)
VPC that defines the virtual networking environment for this DB instance.

tutorial-vpc (vpc-08bf0876fa2e229cf)

Only VPCs with a corresponding DB subnet group are listed.

i After a database is created, you can't change the VPC selection.

Subnet group [Info](#)
DB subnet group that defines which subnets and IP ranges the DB instance can use in the VPC you selected.

tutorial-db-subnet-group

Public access [Info](#)

Yes
Amazon EC2 instances and devices outside the VPC can connect to your database. Choose one or more VPC security groups that specify which EC2 instances and devices inside the VPC can connect to the database.

No
RDS will not assign a public IP address to the database. Only Amazon EC2 instances and devices inside the VPC can connect to your database.

VPC security group
Choose a VPC security group to allow access to your database. Ensure that the security group rules allow the appropriate incoming traffic.

Choose existing
Choose existing VPC security groups

Create new
Create new VPC security group

Existing VPC security groups

Choose VPC security groups

tutorial-db-securitygroup X

Availability Zone [Info](#)

No preference

▼ Additional configuration

Database port [Info](#)
TCP/IP port that the database will use for application connections.

3306

11. Open the **Additional configuration** section, and enter **sample** for **Initial database name**. Keep the default settings for the other options.
 12. To create your Aurora MySQL DB cluster, choose **Create database**.
- Your new DB cluster appears in the **Databases** list with the status **Creating**.
13. Wait for the **Status** of your new DB cluster to show as **Available**. Then choose the DB cluster name to show its details.
 14. In the **Connectivity & security** section, view the **Endpoint** and **Port** of the writer DB instance.

The screenshot shows the AWS RDS console with the following details:

- Databases > tutorial-db-cluster**
- tutorial-db-cluster** (DB identifier)
- Related** section: A search bar labeled "Filter databases".
- DB identifier** table:

DB identifier	Role	Engine	Region & AZ	Size
tutorial-db-cluster	Regional	Aurora MySQL	us-west-2	1 instance
tutorial-db-cluster-instance-1	Writer	Aurora MySQL	-	db.t2.small
- Endpoints (2)** table:

Endpoint name	Status	Type	Port
tutorial-db-cluster.cluster-ro-*.us-west-2.rds.amazonaws.com	Creating	Reader	3306
tutorial-db-cluster.cluster-*.us-west-2.rds.amazonaws.com	Creating	Writer	3306

 The "Writer" endpoint and its port (3306) are circled in red.

Note the endpoint and port for your writer DB instance. You use this information to connect your web server to your DB cluster.

15. Complete [Create an EC2 instance and install a web server \(p. 109\)](#).

Create an EC2 instance and install a web server

In this step, you create a web server to connect to the Amazon Aurora DB cluster that you created in [Create an Amazon Aurora DB cluster \(p. 104\)](#).

Launch an EC2 instance

First, you create an Amazon EC2 instance in the public subnet of your VPC.

To launch an EC2 instance

1. Sign in to the AWS Management Console and open the Amazon EC2 console at <https://console.aws.amazon.com/ec2/>.
2. Choose **EC2 Dashboard**, and then choose **Launch instance**, as shown following.

The screenshot shows the AWS EC2 Resources page. At the top, it displays resource counts: 1 Running instances, 0 Snapshots, 2 Key pairs, 1 Elastic IPs, 0 Volumes, and 0 Security groups. Below this, a callout box says: "Easily size, configure, and deploy Microsoft SQL Server Always On availability groups on AWS". The main section is titled "Launch instance" with the sub-instruction: "To get started, launch an Amazon EC2 instance, which is a virtual server in the cloud." A large red circle highlights the "Launch instance" button. A note below it states: "Note: Your instances will launch in the US West (Oregon) Region". To the right, there are service links for "Server Migration", "Region Selector", "US West (Oregon)", and "Available Regions".

3. Choose the **Amazon Linux 2 AMI**.

The screenshot shows the "Step 1: Choose an Amazon Machine Image (AMI)" screen. It lists three AMIs: "Amazon Linux 2 AMI (HVM), SSD Volume Type", "Red Hat Enterprise Linux 8 (HVM), SSD Volume Type", and "SUSE Linux Enterprise Server 15 SP2 (HVM), SSD Volume Type". The first item is selected and highlighted with a red box. The "Select" button next to it is also highlighted. The "Amazon Linux" row includes a note: "Amazon Linux 2 comes with five years support. It provides Linux kernel 4.14 tuned for optimal performance on Amazon EC2, systemd 219, GCC 7.3, Glibc 2.26, Binutils 2.29.1, and the latest software packages through extras. This AMI is the successor of the Amazon Linux AMI that is approaching end of life on December 31, 2020 and has been removed from this wizard." There are also checkboxes for "Root device type: ebs", "Virtualization type: hvm", and "ENAs Enabled: Yes".

4. Choose the **t2.micro** instance type, as shown following, and then choose **Next: Configure Instance Details**.

Step 2: Choose an Instance Type

Amazon EC2 provides a wide selection of instance types optimized to fit different use cases. Instances are virtual servers that can run applications. They have varying combinations of CPU, memory, storage, and networking capacity, and give you the flexibility to choose the appropriate mix of resources for your applications. [Learn more](#) about instance types and how they can meet your computing needs.

Filter by: All instance families Current generation Show/Hide Columns								
Currently selected: t2.micro (- ECUs, 1 vCPUs, 2.5 GHz, -, 1 GiB memory, EBS only)								
	Family	Type	vCPUs	Memory (GiB)	Instance Storage (GB)	EBS-Optimized Available	Network Performance	IPv6 Support
<input type="checkbox"/>	t2	t2.nano	1	0.5	EBS only	-	Low to Moderate	Yes
<input checked="" type="checkbox"/>	t2	t2.micro <small>Free tier eligible</small>	1	1	EBS only	-	Low to Moderate	Yes
<input type="checkbox"/>	t2	t2.small	1	2	EBS only	-	Low to Moderate	Yes
<input type="checkbox"/>	t2	t2.medium	2	4	EBS only	-	Low to Moderate	Yes
<input type="checkbox"/>	t2	t2.large	2	8	EBS only	-	Low to Moderate	Yes

[Cancel](#) [Previous](#) [Review and Launch](#) [Next: Configure Instance Details](#)

5. On the **Configure Instance Details** page, shown following, set these values and keep the other values as their defaults:

- **Network:** Choose the VPC with both public and private subnets that you chose for the DB cluster, such as the `vpc-identifier` | `tutorial-vpc` created in [Create a VPC with private and public subnets \(p. 1805\)](#).
- **Subnet:** Choose an existing public subnet, such as `subnet-identifier` | `Tutorial public` | `us-west-2a` created in [Create a VPC security group for a public web server \(p. 1807\)](#).
- **Auto-assign Public IP:** Choose **Enable**.

Step 3: Configure Instance Details

Configure the instance to suit your requirements. You can launch multiple instances from the same AMI, request Spot instances to take advantage of the lower pricing, assign an access management role to the instance, and more.

Number of instances	<input type="text" value="1"/>	Launch into Auto Scaling Group
Purchasing option	<input type="checkbox"/> Request Spot instances	
Network	vpc-<i>identifier</i> tutorial-vpc Create new VPC Subnet Create new subnet 249 IP Addresses available	
Auto-assign Public IP	Enable	
Placement group	<input type="checkbox"/> Add instance to placement group	
Capacity Reservation	Open	
Domain join directory	No directory Create new directory	
IAM role	None Create new IAM role	
CPU options	<input type="checkbox"/> Specify CPU options	
Shutdown behavior	Stop	
Stop - Hibernate behavior	<input type="checkbox"/> Enable hibernation as an additional stop behavior	
Enable termination protection	<input type="checkbox"/> Protect against accidental termination	
Monitoring	<input type="checkbox"/> Enable CloudWatch detailed monitoring <small>Additional charges apply.</small>	
Tenancy	Shared - Run a shared hardware instance <small>Additional charges will apply for dedicated tenancy.</small>	
Elastic Inference	<input type="checkbox"/> Add an Elastic Inference accelerator	

[Cancel](#) [Previous](#) [Review and Launch](#) [Next: Add Storage](#)

6. Choose **Next: Add Storage**.
7. On the **Add Storage** page, keep the default values and choose **Next: Add Tags**.
8. On the **Add Tags** page, shown following, choose **Add Tag**, then enter **Name** for **Key** and enter **tutorial-web-server** for **Value**.

Step 5: Add Tags

A tag consists of a case-sensitive key-value pair. For example, you could define a tag with key = Name and value = Webserver. A copy of a tag can be applied to volumes, instances or both. Tags will be applied to all instances and volumes. [Learn more](#) about tagging your Amazon EC2 resources.

Key (127 characters maximum)	Value (255 characters maximum)	Instances <small>(1)</small>	Volumes <small>(1)</small>
<input type="text" value="Name"/>	<input type="text" value="tutorial-web-server"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
Add another tag (Up to 50 tags maximum)		Cancel Previous Review and Launch Next: Configure Security Group	

9. Choose **Next: Configure Security Group**.
10. On the **Configure Security Group** page, shown following, choose **Select an existing security group**. Then choose an existing security group, such as the **tutorial-securitygroup** created in [Create a VPC security group for a public web server \(p. 1807\)](#). Make sure that the security group that you choose includes inbound rules for Secure Shell (SSH) and HTTP access.

Step 6: Configure Security Group

A security group is a set of firewall rules that control the traffic for your instance. On this page, you can add rules to allow specific traffic to reach your instance. For example, if you want to set up a web server and allow Internet traffic to reach your instance, add rules that allow unrestricted access to the HTTP and HTTPS ports. You can create a new security group or select from an existing one below. [Learn more](#) about Amazon EC2 security groups.

Assign a security group: Create a new security group Select an existing security group

Security Group ID	Name	Description	Actions
sg-0ef508f81f84a5764	default	default VPC security group	Copy to new
sg-0ef508f81f84a5764	tutorial-db-securitygroup	Tutorial DB Instance Security Group	Copy to new
sg-0ef508f81f84a5764	tutorial-securitygroup	Tutorial Security Group	Copy to new

Inbound rules for sg-0ef508f81f84a5764 (Selected security groups: sg-0ef508f81f84a5764)

Type	Protocol	Port Range	Source	Description
HTTP	TCP	80	<input type="text" value="0.0.0.0/0"/>	
SSH	TCP	22	<input type="text" value="0.0.0.0/0"/>	

Cancel **Previous** **Review and Launch**

11. Choose **Review and Launch**.
12. On the **Review Instance Launch** page, shown following, verify your settings and then choose **Launch**.

Step 7: Review Instance Launch

The screenshot shows the 'Step 7: Review Instance Launch' page for creating a Lambda function. At the top, it displays the selected AMI: 'Amazon Linux 2 AMI (HVM), SSD Volume Type - ami-0cf6f5c8a62fa5da6'. A note indicates it's a 'Free tier eligible' instance. Below this, there's a brief description of the Amazon Linux 2 AMI and its compatibility with EC2. The 'Root Device Type: ebs' and 'Virtualization type: hvm' are also listed.

Instance Type:

Instance Type	ECUs	vCPUs	Memory (GiB)	Instance Storage (GB)	EBS-Optimized Available	Network Performance
t2.micro	-	1	1	EBS only	-	Low to Moderate

Security Groups:

Security Group ID	Name	Description
sg-...	tutorial-securitygroup	Tutorial Security Group

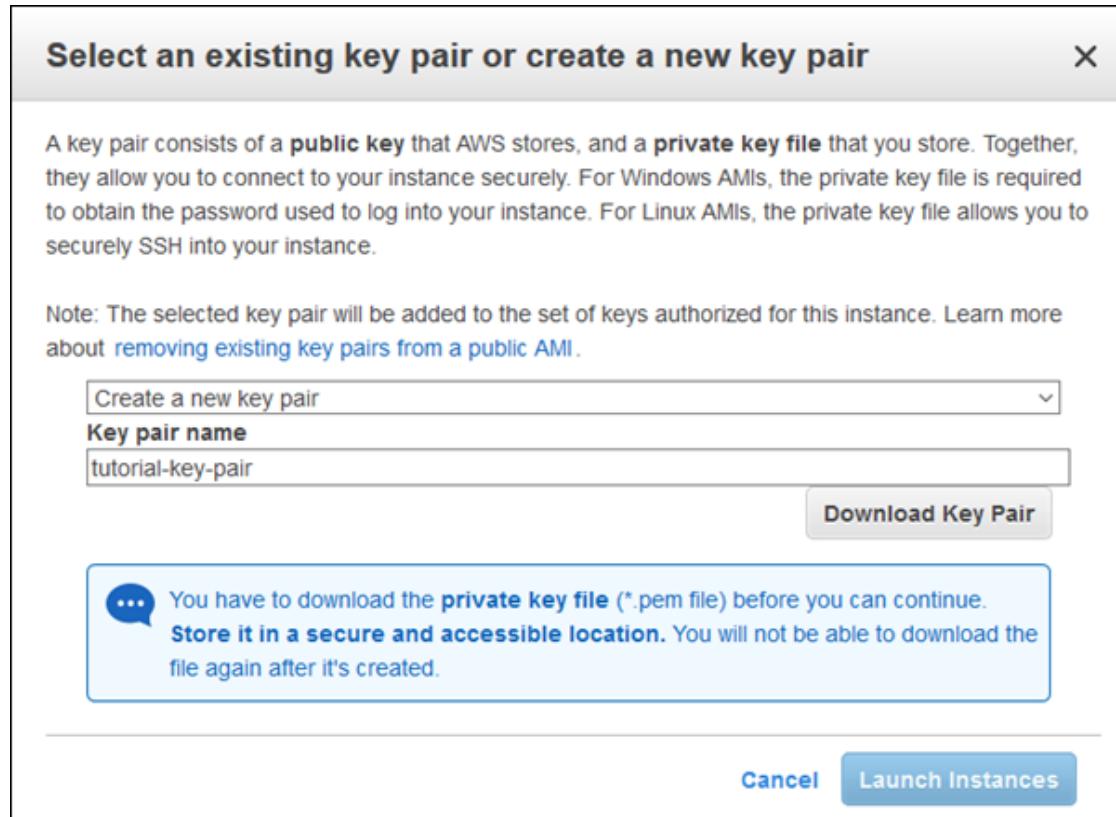
All selected security groups inbound rules:

Type	Protocol	Port Range	Source	Description
HTTP	TCP	80	...	
SSH	TCP	22	...	

Instance Details:

Cancel Previous Launch

- On the **Select an existing key pair or create a new key pair** page, shown following, choose **Create a new key pair** and set **Key pair name** to `tutorial-key-pair`. Choose **Download Key Pair**, and then save the key pair file on your local machine. You use this key pair file to connect to your EC2 instance.



14. To launch your EC2 instance, choose **Launch Instances**. On the **Launch Status** page, shown following, note the identifier for your new EC2 instance, for example: i-0288d65fd4470b6a9.

Launch Status

Your instances are now launching
The following instance launches have been initiated: [i-0288d65fd4470b6a9](#) [View launch log](#)

Get notified of estimated charges
Create [billing alerts](#) to get an email notification when estimated charges on your AWS bill exceed an amount you define (for example, if you exceed the free usage tier).

How to connect to your instances

Your instances are launching, and it may take a few minutes until they are in the **running** state, when they will be ready for you to use. Usage hours on your new instances will start immediately and continue to accrue until you stop or terminate your instances.

Click [View Instances](#) to monitor your instances' status. Once your instances are in the **running** state, you can [connect](#) to them from the Instances screen. [Find out](#) how to connect to your instances.

▼ Here are some helpful resources to get you started

- [How to connect to your Linux instance](#)
- [Learn about AWS Free Usage Tier](#)
- [Amazon EC2: User Guide](#)
- [Amazon EC2: Discussion Forum](#)

While your instances are launching you can also

- [Create status check alarms](#) to be notified when these instances fail status checks. (Additional charges may apply)
- [Create and attach additional EBS volumes](#) (Additional charges may apply)
- [Manage security groups](#)

[View Instances](#)

15. Choose [View Instances](#) to find your instance.
16. Wait until **Instance Status** for your instance reads as **Running** before continuing.

Install an Apache web server with PHP

Next, you connect to your EC2 instance and install the web server.

To connect to your EC2 instance and install the Apache web server with PHP

1. Connect to the EC2 instance that you created earlier by following the steps in [Connect to your Linux instance](#).
2. Get the latest bug fixes and security updates by updating the software on your EC2 instance. To do this, use the following command.

Note

The `-y` option installs the updates without asking for confirmation. To examine updates before installing, omit this option.

```
sudo yum update -y
```

3. After the updates complete, install the PHP software using the `amazon-linux-extras install` command. This command installs multiple software packages and related dependencies at the same time.

```
sudo amazon-linux-extras install -y lamp-mariadb10.2-php7.2 php7.2
```

If you receive an error stating `sudo: amazon-linux-extras: command not found`, then your instance was not launched with an Amazon Linux 2 AMI (perhaps you are using the Amazon Linux AMI instead). You can view your version of Amazon Linux using the following command.

```
cat /etc/system-release
```

For more information, see [Updating instance software](#).

4. Install the Apache web server.

```
sudo yum install -y httpd
```

5. Start the web server with the command shown following.

```
sudo systemctl start httpd
```

You can test that your web server is properly installed and started. To do this, enter the public Domain Name System (DNS) name of your EC2 instance in the address bar of a web browser, for example: `http://ec2-42-8-168-21.us-west-1.compute.amazonaws.com`. If your web server is running, then you see the Apache test page.

If you don't see the Apache test page, check your inbound rules for the VPC security group that you created in [Tutorial: Create an Amazon VPC for use with a DB instance \(p. 1805\)](#). Make sure that your inbound rules include a rule allowing HTTP (port 80) access for the IP address you use to connect to the web server.

Note

The Apache test page appears only when there is no content in the document root directory, `/var/www/html`. After you add content to the document root directory, your content appears at the public DNS address of your EC2 instance instead of the Apache test page.

6. Configure the web server to start with each system boot using the `systemctl` command.

```
sudo systemctl enable httpd
```

To allow `ec2-user` to manage files in the default root directory for your Apache web server, modify the ownership and permissions of the `/var/www` directory. There are many ways to accomplish this task. In this tutorial, you add `ec2-user` to the `apache` group, to give the `apache` group ownership of the `/var/www` directory and assign write permissions to the group.

To set file permissions for the Apache web server

1. Add the `ec2-user` user to the `apache` group.

```
sudo usermod -a -G apache ec2-user
```

2. Log out to refresh your permissions and include the new `apache` group.

```
exit
```

3. Log back in again and verify that the `apache` group exists with the `groups` command.

```
groups
```

Your output looks similar to the following:

```
ec2-user adm wheel apache systemd-journal
```

4. Change the group ownership of the /var/www directory and its contents to the apache group.

```
sudo chown -R ec2-user:apache /var/www
```

5. Change the directory permissions of /var/www and its subdirectories to add group write permissions and set the group ID on subdirectories created in the future.

```
sudo chmod 2775 /var/www
find /var/www -type d -exec sudo chmod 2775 {} \;
```

6. Recursively change the permissions for files in the /var/www directory and its subdirectories to add group write permissions.

```
find /var/www -type f -exec sudo chmod 0664 {} \;
```

Now, `ec2-user` (and any future members of the `apache` group) can add, delete, and edit files in the Apache document root, enabling you to add content, such as a static website or a PHP application.

Note

A web server running the HTTP protocol provides no transport security for the data that it sends or receives. When you connect to an HTTP server using a web browser, the URLs that you visit, the content of web pages that you receive, and the contents (including passwords) of any HTML forms that you submit are all visible to eavesdroppers anywhere along the network pathway. The best practice for securing your web server is to install support for HTTPS (HTTP Secure), which protects your data with SSL/TLS encryption. For more information, see [Tutorial: Configure SSL/TLS with the Amazon Linux AMI](#) in the *Amazon EC2 User Guide*.

Connect your Apache web server to your DB instance

Next, you add content to your Apache web server that connects to your Amazon Aurora DB cluster.

To add content to the Apache web server that connects to your DB cluster

1. While still connected to your EC2 instance, change the directory to /var/www and create a new subdirectory named `inc`.

```
cd /var/www
mkdir inc
cd inc
```

2. Create a new file in the `inc` directory named `dbinfo.inc`, and then edit the file by calling `nano` (or the editor of your choice).

```
>dbinfo.inc
nano dbinfo.inc
```

3. Add the following contents to the `dbinfo.inc` file. Here, `db_instance_endpoint` is DB cluster writer endpoint, without the port, and `master_password` is the master password for your DB cluster.

Note

We recommend placing the user name and password information in a folder that isn't part of the document root for your web server. Doing this reduces the possibility of your security information being exposed.

```
<?php

define('DB_SERVER', 'db_cluster_writer_endpoint');
define('DB_USERNAME', 'tutorial_user');
define('DB_PASSWORD', 'master password');
define('DB_DATABASE', 'sample');

?>
```

4. Save and close the dbinfo.inc file.
5. Change the directory to /var/www/html.

```
cd /var/www/html
```

6. Create a new file in the html directory named SamplePage.php, and then edit the file by calling nano (or the editor of your choice).

```
>SamplePage.php
nano SamplePage.php
```

7. Add the following contents to the SamplePage.php file:

Note

We recommend placing the user name and password information in a folder that isn't part of the document root for your web server. Doing this reduces the possibility of your security information being exposed.

```
<?php include "../inc/dbinfo.inc"; ?>
<html>
<body>
<h1>Sample page</h1>
<?php

/* Connect to MySQL and select the database. */
$connection = mysqli_connect(DB_SERVER, DB_USERNAME, DB_PASSWORD);

if (mysqli_connect_errno()) echo "Failed to connect to MySQL: " .
mysqli_connect_error();

$database = mysqli_select_db($connection, DB_DATABASE);

/* Ensure that the EMPLOYEES table exists. */
VerifyEmployeesTable($connection, DB_DATABASE);

/* If input fields are populated, add a row to the EMPLOYEES table. */
$employee_name = htmlentities($_POST['NAME']);
$employee_address = htmlentities($_POST['ADDRESS']);

if (strlen($employee_name) || strlen($employee_address)) {
    AddEmployee($connection, $employee_name, $employee_address);
}
?>
```

```

<!-- Input form -->
<form action="<?PHP echo $_SERVER['SCRIPT_NAME'] ?>" method="POST">
  <table border="0">
    <tr>
      <td>NAME</td>
      <td>ADDRESS</td>
    </tr>
    <tr>
      <td>
        <input type="text" name="NAME" maxlength="45" size="30" />
      </td>
      <td>
        <input type="text" name="ADDRESS" maxlength="90" size="60" />
      </td>
      <td>
        <input type="submit" value="Add Data" />
      </td>
    </tr>
  </table>
</form>

<!-- Display table data. -->
<table border="1" cellpadding="2" cellspacing="2">
  <tr>
    <td>ID</td>
    <td>NAME</td>
    <td>ADDRESS</td>
  </tr>
<?php

$result = mysqli_query($connection, "SELECT * FROM EMPLOYEES");

while($query_data = mysqli_fetch_row($result)) {
  echo "<tr>";
  echo "<td>",$query_data[0], "</td>",
        "<td>",$query_data[1], "</td>",
        "<td>",$query_data[2], "</td>";
  echo "</tr>";
}
?>

</table>

<!-- Clean up. -->
<?php

mysqli_free_result($result);
mysqli_close($connection);

?>

</body>
</html>

<?php

/* Add an employee to the table. */
function AddEmployee($connection, $name, $address) {
  $n = mysqli_real_escape_string($connection, $name);
  $a = mysqli_real_escape_string($connection, $address);

  $query = "INSERT INTO EMPLOYEES (NAME, ADDRESS) VALUES ('$n', '$a');";

  if(!mysqli_query($connection, $query)) echo("<p>Error adding employee data.</p>");
```

```
}

/* Check whether the table exists and, if not, create it. */
function VerifyEmployeesTable($connection, $dbName) {
    if(!TableExists("EMPLOYEES", $connection, $dbName))
    {
        $query = "CREATE TABLE EMPLOYEES (
            ID int(11) UNSIGNED AUTO_INCREMENT PRIMARY KEY,
            NAME VARCHAR(45),
            ADDRESS VARCHAR(90)
        )";

        if(!mysqli_query($connection, $query)) echo("<p>Error creating table.</p>");
    }
}

/* Check for the existence of a table. */
function TableExists($tableName, $connection, $dbName) {
    $t = mysqli_real_escape_string($connection, $tableName);
    $d = mysqli_real_escape_string($connection, $dbName);

    $checktable = mysqli_query($connection,
        "SELECT TABLE_NAME FROM information_schema.TABLES WHERE TABLE_NAME = '$t' AND
        TABLE_SCHEMA = '$d'");
    if(mysqli_num_rows($checktable) > 0) return true;
    return false;
}
?>
```

8. Save and close the SamplePage.php file.
9. Verify that your web server successfully connects to your DB cluster by opening a web browser and browsing to <http://EC2 instance endpoint/SamplePage.php>, for example: <http://ec2-55-122-41-31.us-west-2.compute.amazonaws.com/SamplePage.php>.

You can use SamplePage.php to add data to your DB cluster. The data that you add is then displayed on the page. To verify that the data was inserted into the table, you can install MySQL on the Amazon EC2 instance, connect to the DB instance, and query the table.

For information about connecting to a DB cluster, see [Connecting to an Amazon Aurora DB cluster \(p. 281\)](#).

To make sure that your DB cluster is as secure as possible, verify that sources outside of the VPC can't connect to your DB cluster.

After you have finished testing your web server and your database, you should delete your DB cluster and your Amazon EC2 instance.

- To delete a DB cluster, follow the instructions in [Deleting Aurora DB clusters and DB instances \(p. 467\)](#). You don't need to create a final snapshot.
- To terminate an Amazon EC2 instance, follow the instruction in [Terminate your instance](#) in the *Amazon EC2 User Guide*.

Amazon Aurora tutorials and sample code

The AWS documentation includes several tutorials that guide you through common Amazon Aurora use cases. Many of these tutorials show you how to use Amazon Aurora with other AWS services. In addition, you can access sample code in GitHub.

Note

You can find more tutorials at the [AWS Database Blog](#). For information about training, see [AWS Training and Certification](#).

Topics

- [Tutorials in this guide \(p. 121\)](#)
- [Tutorials in other AWS guides \(p. 121\)](#)
- [Tutorials and sample code in GitHub \(p. 122\)](#)

Tutorials in this guide

The following tutorials in this guide show you how to perform common tasks with Amazon Aurora:

- [Tutorial: Create an Amazon VPC for use with a DB instance \(p. 1805\)](#)

Learn how to include a DB cluster in an Amazon virtual private cloud (VPC) that shares data with a web server that is running on an Amazon EC2 instance in the same VPC.

- [Tutorial: Create a web server and an Amazon Aurora DB cluster \(p. 103\)](#)

Learn how to install an Apache web server with PHP and create a MySQL database. The web server runs on an Amazon EC2 instance using Amazon Linux, and the MySQL database is an Aurora MySQL DB cluster. Both the Amazon EC2 instance and the DB cluster run in an Amazon VPC.

- [Tutorial: Use tags to specify which Aurora DB clusters to stop \(p. 479\)](#)

Learn how to use tags to specify which Aurora DB clusters to stop.

- [Tutorial: Log the state of an instance using EventBridge \(p. 692\)](#)

Learn how to log a DB instance state change using Amazon EventBridge and AWS Lambda.

Tutorials in other AWS guides

The following tutorials in other AWS guides show you how to perform common tasks with Amazon Aurora:

Note

Some of the tutorials use Amazon RDS DB instances, but they can be adapted to use Aurora DB clusters.

- [Tutorial: Aurora Serverless in the AWS AppSync Developer Guide](#)

Learn how to use AWS AppSync to provide a data source for executing SQL commands against Aurora Serverless v1 DB clusters with the Data API enabled. You can use AWS AppSync resolvers to execute SQL statements against the Data API with GraphQL queries, mutations, and subscriptions.

- [Tutorial: Rotating a Secret for an AWS Database](#) in the [AWS Secrets Manager User Guide](#)

Learn how to create a secret for an AWS database and configure the secret to rotate on a schedule. You trigger one rotation manually, and then confirm that the new version of the secret continues to provide access.

- [Tutorial: Configuring a Lambda function to access Amazon RDS in an Amazon VPC](#) in the [AWS Lambda Developer Guide](#)

Learn how to create a Lambda function to access a database, create a table, add a few records, and retrieve the records from the table. You also learn how to invoke the Lambda function and verify the query results.

- [Tutorials and samples](#) in the [AWS Elastic Beanstalk Developer Guide](#)

Learn how to deploy applications that use Amazon RDS databases with AWS Elastic Beanstalk.

- [Using Data from an Amazon RDS Database to Create an Amazon ML Datasource](#) in the [Amazon Machine Learning Developer Guide](#)

Learn how to create an Amazon Machine Learning (Amazon ML) datasource object from data stored in a MySQL DB instance.

- [Manually Enabling Access to an Amazon RDS Instance in a VPC](#) in the [Amazon QuickSight User Guide](#)

Learn how to enable Amazon QuickSight access to an Amazon RDS DB instance in a VPC.

Tutorials and sample code in GitHub

The following tutorials and sample code in GitHub show you how to perform common tasks with Amazon Aurora:

Note

Some of the tutorials use Amazon RDS DB instances, but they can be adapted to use Aurora DB clusters.

- [Creating a Job Posting Site using Amazon Aurora and Amazon Translation Services](#)

Learn how to create a web application that stores and queries data by using Amazon Aurora, Elastic Beanstalk, and SDK for Java 2.x. The application created in this AWS tutorial is a job posting web application that lets an employer, an administrator, or human resources staff alert employees or the public about a job opening within a company.

- [Creating the Amazon Relational Database Service item tracker](#)

Learn how to create an application that tracks and reports on work items using Amazon RDS, Amazon Simple Email Service, Elastic Beanstalk, and SDK for Java 2.x.

- [SDK for Go code samples for Amazon RDS](#)

View a collection of SDK for Go code samples for Amazon RDS and Aurora.

- [SDK for Java 2.x code samples for Amazon RDS](#)

View a collection of SDK for Java 2.x code samples for Amazon RDS and Aurora.

- [SDK for PHP code samples for Amazon RDS](#)

View a collection of SDK for PHP code samples for Amazon RDS and Aurora.

- [SDK for Ruby code samples for Amazon RDS](#)

View a collection of SDK for Ruby code samples for Amazon RDS and Aurora.

Configuring your Amazon Aurora DB cluster

This section shows how to set up your Aurora DB cluster. Before creating an Aurora DB cluster, decide on the DB instance class that will run the DB cluster. Also, decide where the DB cluster will run by choosing an AWS Region. Next, create the DB cluster. If you have data outside of Aurora, you can migrate the data into an Aurora DB cluster.

Topics

- [Creating an Amazon Aurora DB cluster \(p. 125\)](#)
- [Creating Amazon Aurora resources with AWS CloudFormation \(p. 146\)](#)
- [Using Amazon Aurora Serverless v1 \(p. 147\)](#)
- [Using Amazon Aurora Serverless v2 \(preview\) \(p. 212\)](#)
- [Using Amazon Aurora global databases \(p. 225\)](#)
- [Connecting to an Amazon Aurora DB cluster \(p. 281\)](#)
- [Using Amazon RDS Proxy \(p. 288\)](#)
- [Working with DB parameter groups and DB cluster parameter groups \(p. 339\)](#)
- [Migrating data to an Amazon Aurora DB cluster \(p. 366\)](#)

Creating an Amazon Aurora DB cluster

An Amazon Aurora DB cluster consists of a DB instance, compatible with either MySQL or PostgreSQL, and a cluster volume that holds the data for the DB cluster, copied across three Availability Zones as a single, virtual volume. By default, an Aurora DB cluster contains a primary DB instance that performs reads and writes, and, optionally, up to 15 Aurora Replicas (reader DB instances). For more information about Aurora DB clusters, see [Amazon Aurora DB clusters \(p. 3\)](#).

Following, you can find out how to create an Aurora DB cluster. To get started, first see [DB cluster prerequisites \(p. 125\)](#).

For simple instructions on connecting to your Aurora DB cluster, see [Connecting to an Amazon Aurora DB cluster \(p. 281\)](#).

DB cluster prerequisites

Important

Before you can create an Aurora DB cluster, you must complete the tasks in [Setting up your environment for Amazon Aurora \(p. 84\)](#).

The following are prerequisites to create a DB cluster.

VPC, subnets, and AZs

You can create an Amazon Aurora DB cluster only in a virtual private cloud (VPC) based on the Amazon VPC service, in an AWS Region that has at least two Availability Zones. The DB subnet group that you choose for the DB cluster must cover at least two Availability Zones. This configuration ensures that your DB cluster always has at least one DB instance available for failover, in the unlikely event of an Availability Zone failure.

If you use the AWS Management Console to create your Aurora DB cluster, you can have Amazon RDS automatically create a VPC for you. Or you can use an existing VPC or create a new VPC for your Aurora DB cluster. Whichever approach you take, your VPC must have at least one subnet in each of at least two Availability Zones for you to use it with an Amazon Aurora DB cluster.

By default, Amazon RDS creates the primary DB instance and the Aurora Replica in the AZs automatically for you. To choose a specific AZ, you need to change the **Availability & durability** Multi-AZ deployment setting to **Don't create an Aurora Replica**. Doing so exposes a drop-down selector that lets you choose from among the AZs in your VPC. However, we strongly recommend that you keep the default setting and let Amazon RDS create a multi-AZ deployment and choose AZs for you. By doing so, your Aurora DB cluster is created with the fast failover and high availability features that are two of Aurora's key benefits.

For more information, see [How to create a VPC for use with Amazon Aurora \(p. 1793\)](#). For information on VPCs, see [Amazon Virtual Private Cloud VPCs](#) and [Amazon Aurora \(p. 1787\)](#).

Note

You can communicate with an EC2 instance that is not in a VPC and an Amazon Aurora DB cluster using ClassicLink. For more information, see [A DB instance in a VPC accessed by an EC2 instance not in a VPC \(p. 1803\)](#).

If you don't have a default VPC or you haven't created a VPC, you can have Amazon RDS automatically create a VPC for you when you create an Aurora DB cluster using the console. Otherwise, you must do the following:

- Create a VPC with at least one subnet in each of at least two of the Availability Zones in the AWS Region where you want to deploy your DB cluster. For more information, see [How to create a VPC for use with Amazon Aurora \(p. 1793\)](#).

- Specify a VPC security group that authorizes connections to your Aurora DB cluster. For more information, see [Working with a DB instance in a VPC \(p. 1788\)](#).
- Specify an RDS DB subnet group that defines at least two subnets in the VPC that can be used by the Aurora DB cluster. For more information, see [Working with DB subnet groups \(p. 1788\)](#).

Additional prerequisites

If you are connecting to AWS using AWS Identity and Access Management (IAM) credentials, your AWS account must have IAM policies that grant the permissions required to perform Amazon RDS operations. For more information, see [Identity and access management in Amazon Aurora \(p. 1724\)](#).

If you are using IAM to access the Amazon RDS console, you must first sign on to the AWS Management Console with your IAM user credentials. Then go to the Amazon RDS console at <https://console.aws.amazon.com/rds/>.

If you want to tailor the configuration parameters for your DB cluster, you must specify a DB cluster parameter group and DB parameter group with the required parameter settings. For information about creating or modifying a DB cluster parameter group or DB parameter group, see [Working with DB parameter groups and DB cluster parameter groups \(p. 339\)](#).

You must determine the TCP/IP port number to specify for your DB cluster. The firewalls at some companies block connections to the default ports (3306 for MySQL, 5432 for PostgreSQL) for Aurora. If your company firewall blocks the default port, choose another port for your DB cluster. All instances in a DB cluster use the same port.

Creating a DB cluster

You can create an Aurora DB cluster using the AWS Management Console, the AWS CLI, or the RDS API.

Note

If you are using the console, a new console interface is available for database creation. Choose either the **New Console** or the **Original Console** instructions based on the console that you are using. The **New Console** instructions are open by default.

New console

You can create a DB instance running MySQL with the AWS Management Console with **Easy create** enabled or not enabled. With **Easy create** enabled, you specify only the DB engine type, DB instance size, and DB instance identifier. **Easy create** uses the default setting for other configuration options. With **Easy create** not enabled, you specify more configuration options when you create a database, including ones for availability, security, backups, and maintenance.

Note

For this example, **Standard create** is enabled, and **Easy create** isn't enabled. For information about creating an Aurora MySQL DB cluster with **Easy create** enabled, see [Getting started with Amazon Aurora \(p. 89\)](#).

To create an Aurora DB cluster using the console

1. Sign in to the AWS Management Console and open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. In the upper-right corner of the AWS Management Console, choose the AWS Region in which you want to create the DB cluster.
Aurora is not available in all AWS Regions. For a list of AWS Regions where Aurora is available, see [Region availability \(p. 12\)](#).
3. In the navigation pane, choose **Databases**.
4. Choose **Create database**.

5. In **Choose a database creation method**, choose **Standard create**.
6. In **Engine options**, choose **Amazon Aurora**.

RDS > Create database

Create database

Choose a database creation method Info

Standard create
You set all of the configuration options, including ones for availability, security, backups, and maintenance.

Easy create
Use recommended best-practice configurations. Some configuration options can be changed after the database is created.

Engine options

Engine type Info

Amazon Aurora 

MySQL 

MariaDB 

PostgreSQL 

Oracle 

Microsoft SQL Server 

Edition

Amazon Aurora with MySQL compatibility

Amazon Aurora with PostgreSQL compatibility

Capacity type Info

Provisioned
You provision and manage the server instance sizes.

7. In **Edition**, choose one of the following:

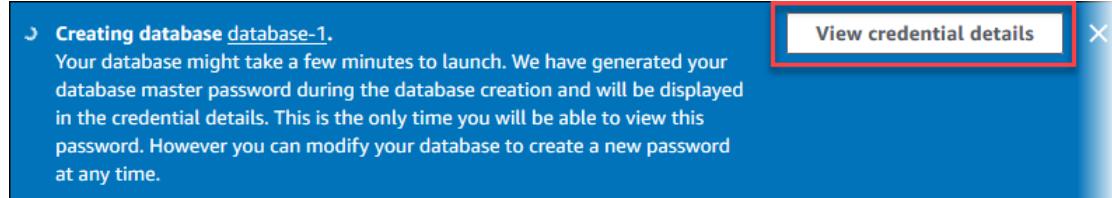
- **Amazon Aurora with MySQL compatibility**
 - **Amazon Aurora with PostgreSQL compatibility**
8. Choose one of the following in **Capacity type**:
- **Provisioned**
For more information, see [Amazon Aurora DB clusters \(p. 3\)](#).
 - **Serverless**
For more information, see [Using Amazon Aurora Serverless v1 \(p. 147\)](#).
9. For **Version**, choose the engine version.
10. In **Templates**, choose the template that matches your use case.
11. To enter your master password, do the following:
- a. In the **Settings** section, open **Credential Settings**.
 - b. Clear the **Auto generate a password** check box.
 - c. (Optional) Change the **Master username** value and enter the same password in **Master password** and **Confirm password**.

By default, the new DB instance uses an automatically generated password for the master user.

12. For the remaining sections, specify your DB cluster settings. For information about each setting, see [Settings for Aurora DB clusters \(p. 137\)](#).
13. Choose **Create database**.

If you chose to use an automatically generated password, the **View credential details** button appears on the **Databases** page.

To view the master user name and password for the DB cluster, choose **View credential details**.



To connect to the DB instance as the master user, use the user name and password that appear.

Important

You can't view the master user password again. If you don't record it, you might have to change it. If you need to change the master user password after the DB instance is available, you can modify the DB instance to do so. For more information about modifying a DB instance, see [Modifying an Amazon Aurora DB cluster \(p. 372\)](#).

14. For **Databases**, choose the name of the new Aurora DB cluster.

On the RDS console, the details for new DB cluster appear. The DB cluster and its DB instance have a status of **creating** until the DB cluster is ready to use.

The screenshot shows the 'Databases' page in the Amazon RDS console. At the top, there are two tabs: 'All' (selected) and 'By database group'. Below the tabs is a search bar labeled 'Filter databases'. The main area displays a table titled 'Databases' with the following data:

DB identifier	Role	Engine	Region & AZ
database-1	Regional cluster	Aurora MySQL	us-east-2
database-1-instance-1	Reader instance	Aurora MySQL	-

When the state changes to **available** for both, you can connect to the DB cluster. Depending on the DB instance class and the amount of storage, it can take up to 20 minutes before the new DB cluster is available.

To view the newly created cluster, choose **Databases** from the navigation pane in the Amazon RDS console. Then choose the DB cluster to show the DB cluster details. For more information, see [Viewing an Amazon Aurora DB cluster \(p. 547\)](#).

The screenshot shows the AWS RDS console interface for a database cluster named "database-1". The "Connectivity & security" tab is active. In the "Endpoints (2)" section, there are two entries:

Endpoint name	Status	Type	Port
database-1.cluster-ro-*.us-east-2.rds.amazonaws.com	Available	Reader instance	3306
database-1.cluster-*.us-east-2.rds.amazonaws.com	Available	Writer instance	3306

The "Writer instance" entry is highlighted with a red circle.

On the **Connectivity & security** tab, note the port and the endpoint of the writer DB instance. Use the endpoint and port of the cluster in your JDBC and ODBC connection strings for any application that performs write or read operations.

Original console

To create an Aurora DB cluster using the AWS Management Console

1. Sign in to the AWS Management Console and open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
 2. In the top-right corner of the AWS Management Console, choose the AWS Region in which you want to create the Aurora DB cluster.
 3. In the navigation pane, choose **Databases**.
- If the navigation pane is closed, choose the menu icon at the top left to open it.
4. Choose **Create database** to open the **Select engine** page.
 5. On the **Select engine** page, choose an edition of Aurora. Choose either MySQL 5.6-compatible, MySQL 5.7-compatible, MySQL 8.0-compatible, or PostgreSQL-compatible.

Select engine

Engine options

Amazon Aurora

Amazon
Aurora

MySQL



MariaDB



PostgreSQL



Oracle

ORACLE®

Microsoft SQL Server



Amazon Aurora

Amazon Aurora is a MySQL- and PostgreSQL-compatible enterprise-class database, starting at <\$1/day.

- Up to 5 times the throughput of MySQL and 3 times the throughput of PostgreSQL
- Up to 64TB of auto-scaling SSD storage
- 6-way replication across three Availability Zones
- Up to 15 Read Replicas with sub-10ms replica lag
- Automatic monitoring and failover in less than 30 seconds

Edition

- MySQL 5.6-compatible
- MySQL 5.7-compatible
- PostgreSQL-compatible

Only enable options eligible for RDS Free Usage Tier [info](#)

Cancel

Next

6. Choose **Next**.
7. On the **Specify DB details** page, specify your DB instance information. For information about each setting, see [Settings for Aurora DB clusters \(p. 137\)](#).

A typical **Specify DB details** page looks like the following.

Specify DB details

Instance specifications

Estimate your monthly costs for the DB Instance using the [AWS Simple Monthly Calculator](#).

DB engine
Aurora - compatible with MySQL 5.7.12

DB instance class [info](#)

Multi-AZ deployment [info](#)
 Create Replica in Different Zone
 No

Settings

DB instance identifier [info](#)
Specify a name that is unique for all DB instances owned by your AWS account in the current region.

DB instance identifier is case insensitive, but stored as all lower-case, as in "mydbinstance".

Master username [info](#)
Specify an alphanumeric string that defines the login ID for the master user.

Master Username must start with a letter.

Master password [info](#) Confirm password [info](#)

Master Password must be at least eight characters long, as in "mypassword".

[Cancel](#) [Previous](#) [Next](#)

8. Confirm your master password and choose **Next**.
9. On the **Configure advanced settings** page, you can customize additional settings for your Aurora DB cluster. For information about each setting, see [Settings for Aurora DB clusters \(p. 137\)](#).
10. Choose **Create database** to create your Aurora DB cluster, and then choose **Close**.

On the Amazon RDS console, the new DB cluster appears in the list of DB clusters. The DB cluster will have a status of **creating** until the DB cluster is created and ready for use. When the state changes to available, you can connect to the writer instance for your DB cluster. Depending on the DB cluster class and store allocated, it can take several minutes for the new cluster to be available.

To view the newly created cluster, choose **Databases** from the navigation pane in the Amazon RDS console and choose the DB cluster to show the DB cluster details. For more information, see [Viewing an Amazon Aurora DB cluster \(p. 547\)](#).

The screenshot shows the AWS RDS console interface for managing a database cluster named "gs-db-cluster1".

Related Databases:

DB identifier	Role	Engine
gs-db-cluster1	Regional	Aurora MySQL
gs-db-instance1	Writer	Aurora MySQL
gs-db-instance1-us-east-2b	Reader	Aurora MySQL

Endpoints (2):

Endpoint name	Status	Type
gs-db-cluster1. [REDACTED] .rds.amazonaws.com	Available	Writer
gs-db-cluster1. [REDACTED] .rds.amazonaws.com	Available	Reader

A red oval highlights the endpoint "gs-db-cluster1.[REDACTED].rds.amazonaws.com" under the "Writer" status.

Note the ports and the endpoints of the cluster. Use the endpoint and port of the writer DB cluster in your JDBC and ODBC connection strings for any application that performs write or read operations.

AWS CLI

Note

Before you can create an Aurora DB cluster using the AWS CLI, you must fulfill the required prerequisites, such as creating a VPC and an RDS DB subnet group. For more information, see [DB cluster prerequisites \(p. 125\)](#).

You can use the AWS CLI to create an Aurora MySQL DB cluster or an Aurora PostgreSQL DB cluster.

To create an Aurora MySQL DB cluster using the AWS CLI

When you create an Aurora MySQL DB cluster or DB instance, ensure that you specify the correct value for the `--engine` option value based on the MySQL compatibility of the DB cluster or DB instance.

- When you create an Aurora MySQL 8.0-compatible or 5.7-compatible DB cluster or DB instance, you specify `aurora-mysql` for the `--engine` option.
- When you create an Aurora MySQL 5.6-compatible DB cluster or DB instance, you specify `aurora` for the `--engine` option.

Complete the following steps:

1. Identify the DB subnet group and VPC security group ID for your new DB cluster, and then call the `create-db-cluster` AWS CLI command to create the Aurora MySQL DB cluster.

For example, the following command creates a new MySQL 8.0-compatible DB cluster named `sample-cluster`.

For Linux, macOS, or Unix:

```
aws rds create-db-cluster --db-cluster-identifier sample-cluster --engine aurora-mysql \
  \
  --engine-version 8.0 --master-username user-name --master-user-password password \
  --db-subnet-group-name mysubnetgroup --vpc-security-group-ids sg-c7e5b0d2
```

For Windows:

```
aws rds create-db-cluster --db-cluster-identifier sample-cluster --engine aurora-mysql \
  ^
  --engine-version 8.0 --master-username user-name --master-user-password password ^
  --db-subnet-group-name mysubnetgroup --vpc-security-group-ids sg-c7e5b0d2
```

The following command creates a new MySQL 5.7-compatible DB cluster named `sample-cluster`.

For Linux, macOS, or Unix:

```
aws rds create-db-cluster --db-cluster-identifier sample-cluster --engine aurora-mysql \
  \
  --engine-version 5.7.12 --master-username user-name --master-user-
  password password \
  --db-subnet-group-name mysubnetgroup --vpc-security-group-ids sg-c7e5b0d2
```

For Windows:

```
aws rds create-db-cluster --db-cluster-identifier sample-cluster --engine aurora-mysql \
  ^
  --engine-version 5.7.12 --master-username user-name --master-user-password password
```

```
--db-subnet-group-name mysubnetgroup --vpc-security-group-ids sg-c7e5b0d2
```

The following command creates a new MySQL 5.6-compatible DB cluster named `sample-cluster`.

For Linux, macOS, or Unix:

```
aws rds create-db-cluster --db-cluster-identifier sample-cluster --engine aurora \  
  --engine-version 5.6.10a --master-username user-name --master-user-  
  password password \  
  --db-subnet-group-name mysubnetgroup --vpc-security-group-ids sg-c7e5b0d2
```

For Windows:

```
aws rds create-db-cluster --db-cluster-identifier sample-cluster --engine aurora ^  
  --engine-version 5.6.10a --master-username user-name --master-user-  
  password password ^  
  --db-subnet-group-name mysubnetgroup --vpc-security-group-ids sg-c7e5b0d2
```

2. If you use the console to create a DB cluster, then Amazon RDS automatically creates the primary instance (writer) for your DB cluster. If you use the AWS CLI to create a DB cluster, you must explicitly create the primary instance for your DB cluster. The primary instance is the first instance that is created in a DB cluster.

Call the [create-db-instance](#) AWS CLI command to create the primary instance for your DB cluster. Include the name of the DB cluster as the `--db-cluster-identifier` option value.

For example, the following command creates a new MySQL 5.7-compatible or MySQL 8.0-compatible DB instance named `sample-instance`.

For Linux, macOS, or Unix:

```
aws rds create-db-instance --db-instance-identifier sample-instance \  
  --db-cluster-identifier sample-cluster --engine aurora-mysql --db-instance-class  
  db.r5.large
```

For Windows:

```
aws rds create-db-instance --db-instance-identifier sample-instance ^  
  --db-cluster-identifier sample-cluster --engine aurora-mysql --db-instance-class  
  db.r5.large
```

The following command creates a new MySQL 5.6-compatible DB instance named `sample-instance`.

For Linux, macOS, or Unix:

```
aws rds create-db-instance --db-instance-identifier sample-instance \  
  --db-cluster-identifier sample-cluster --engine aurora --db-instance-class  
  db.r5.large
```

For Windows:

```
aws rds create-db-instance --db-instance-identifier sample-instance ^  
  --db-cluster-identifier sample-cluster --engine aurora --db-instance-class  
  db.r5.large
```

To create an Aurora PostgreSQL DB cluster using the AWS CLI

1. Identify the DB subnet group and VPC security group ID for your new DB cluster, and then call the [create-db-cluster](#) AWS CLI command to create the Aurora PostgreSQL DB cluster.

For example, the following command creates a new DB cluster named `sample-cluster`.

For Linux, macOS, or Unix:

```
aws rds create-db-cluster --db-cluster-identifier sample-cluster --engine aurora-postgresql \
    --master-username user-name --master-user-password password \
    --db-subnet-group-name mysubnetgroup --vpc-security-group-ids sg-c7e5b0d2
```

For Windows:

```
aws rds create-db-cluster --db-cluster-identifier sample-cluster --engine aurora-postgresql ^
    --master-username user-name --master-user-password password ^
    --db-subnet-group-name mysubnetgroup --vpc-security-group-ids sg-c7e5b0d2
```

2. If you use the console to create a DB cluster, then Amazon RDS automatically creates the primary instance (writer) for your DB cluster. If you use the AWS CLI to create a DB cluster, you must explicitly create the primary instance for your DB cluster. The primary instance is the first instance that is created in a DB cluster.

Call the [create-db-instance](#) AWS CLI command to create the primary instance for your DB cluster. Include the name of the DB cluster as the `--db-cluster-identifier` option value.

For Linux, macOS, or Unix:

```
aws rds create-db-instance --db-instance-identifier sample-instance \
    --db-cluster-identifier sample-cluster --engine aurora-postgresql --db-instance-class db.r4.large
```

For Windows:

```
aws rds create-db-instance --db-instance-identifier sample-instance ^
    --db-cluster-identifier sample-cluster --engine aurora-postgresql --db-instance-class db.r4.large
```

RDS API

Note

Before you can create an Aurora DB cluster using the AWS CLI, you must fulfill the required prerequisites, such as creating a VPC and an RDS DB subnet group. For more information, see [DB cluster prerequisites \(p. 125\)](#).

Identify the DB subnet group and VPC security group ID for your new DB cluster, and then call the [CreateDBCluster](#) operation to create the DB cluster.

When you create an Aurora MySQL DB cluster or DB instance, ensure that you specify the correct value for the `Engine` parameter value based on the MySQL compatibility of the DB cluster or DB instance.

- When you create an Aurora MySQL 5.7 DB cluster or DB instance, you must specify `aurora-mysql` for the `Engine` parameter.

- When you create an Aurora MySQL 5.6 DB cluster or DB instance, you must specify `aurora` for the `Engine` parameter.

When you create an Aurora PostgreSQL DB cluster or DB instance, specify `aurora-postgresql` for the `Engine` parameter.

Settings for Aurora DB clusters

The following table contains details about settings that you choose when you create an Aurora DB cluster.

Note

Additional settings are available if you are creating an Aurora Serverless DB cluster. For information about these settings, see [Creating an Aurora Serverless v1 DB cluster \(p. 161\)](#). Also, some settings aren't available for Aurora Serverless because of Aurora Serverless limitations. For more information, see [Limitations of Aurora Serverless v1 \(p. 148\)](#).

Console setting	Setting description	CLI option and RDS API parameter
Auto minor version upgrade	<p>Choose Enable auto minor version upgrade if you want to enable your Aurora DB cluster to receive preferred minor version upgrades to the DB engine automatically when they become available.</p> <p>The Auto minor version upgrade setting applies to both Aurora PostgreSQL and Aurora MySQL DB clusters. For Aurora MySQL version 1 and version 2 clusters, this setting upgrades the clusters to a maximum version of 1.22.2 and 2.07.2, respectively.</p> <p>For more information about engine updates for Aurora PostgreSQL, see Amazon Aurora PostgreSQL updates (p. 1597).</p> <p>For more information about engine updates for Aurora MySQL, see Database engine updates for Amazon Aurora MySQL (p. 1082).</p>	<p>Set this value for every DB instance in your Aurora cluster. If any DB instance in your cluster has this setting turned off, the cluster isn't automatically upgraded.</p> <p>Using the AWS CLI, run <code>create-db-instance</code> and set the <code>--auto-minor-version-upgrade --no-auto-minor-version-upgrade</code> option.</p> <p>Using the RDS API, call <code>CreateDBInstance</code> and set the <code>AutoMinorVersionUpgrade</code> parameter.</p>
AWS KMS key	<p>Only available if Encryption is set to Enable encryption. Choose the AWS KMS key to use for encrypting this DB cluster. For more information, see Encrypting Amazon Aurora resources (p. 1709).</p>	<p>Using the AWS CLI, run <code>create-db-cluster</code> and set the <code>--kms-key-id</code> option.</p> <p>Using the RDS API, call <code>CreateDBCluster</code> and set the <code>KmsKeyId</code> parameter.</p>
Backtrack	<p>Applies only to Aurora MySQL. Choose Enable Backtrack to enable backtracking or Disable Backtrack to disable backtracking. Using</p>	<p>Using the AWS CLI, run <code>create-db-cluster</code> and set the <code>--backtrack-window</code> option.</p>

Console setting	Setting description	CLI option and RDS API parameter
	backtracking, you can rewind a DB cluster to a specific time, without creating a new DB cluster. It is disabled by default. If you enable backtracking, also specify the amount of time that you want to be able to backtrack your DB cluster (the target backtrack window). For more information, see Backtracking an Aurora DB cluster (p. 816) .	Using the RDS API, call CreateDBCluster and set the BacktrackWindow parameter.
Copy tags to snapshots	Choose this option to copy any DB instance tags to a DB snapshot when you create a snapshot. For more information, see Tagging Amazon RDS resources (p. 474) .	Using the AWS CLI, run create-db-cluster and set the --copy-tags-to-snapshot --no-copy-tags-to-snapshot option. Using the RDS API, call CreateDBCluster and set the CopyTagsToSnapshot parameter.
Database authentication	The database authentication you want to use. For MySQL: <ul style="list-style-type: none">• Choose Password authentication to authenticate database users with database passwords only.• Choose Password and IAM database authentication to authenticate database users with database passwords and user credentials through IAM users and roles. For more information, see IAM database authentication (p. 1743). For PostgreSQL: <ul style="list-style-type: none">• Choose IAM database authentication to authenticate database users with database passwords and user credentials through IAM users and roles. For more information, see IAM database authentication (p. 1743).• Choose Kerberos authentication to authenticate database passwords and user credentials using Kerberos authentication. For more information, see Using Kerberos authentication with Aurora PostgreSQL (p. 1534).	To use IAM database authentication with the AWS CLI, run create-db-cluster and set the --enable-iam-database-authentication --no-enable-iam-database-authentication option. To use IAM database authentication with the RDS API, call CreateDBCluster and set the EnableIAMDatabaseAuthentication parameter. To use Kerberos authentication with the AWS CLI, run create-db-cluster and set the --domain and --domain-iam-role-name options. To use Kerberos authentication with the RDS API, call CreateDBCluster and set the Domain and DomainIAMRoleName parameters.

Console setting	Setting description	CLI option and RDS API parameter
Database port	<p>Specify the port for applications and utilities to use to access the database. Aurora MySQL DB clusters default to the default MySQL port, 3306, and Aurora PostgreSQL DB clusters default to the default PostgreSQL port, 5432. The firewalls at some companies block connections to these default ports. If your company firewall blocks the default port, choose another port for the new DB cluster.</p>	<p>Using the AWS CLI, run create-db-cluster and set the --port option.</p> <p>Using the RDS API, call CreateDBCluster and set the Port parameter.</p>
DB cluster identifier	<p>Enter a name for your DB cluster that is unique for your account in the AWS Region that you chose. This identifier is used in the cluster endpoint address for your DB cluster. For information on the cluster endpoint, see Amazon Aurora connection management (p. 32).</p> <p>The DB cluster identifier has the following constraints:</p> <ul style="list-style-type: none"> • It must contain from 1 to 63 alphanumeric characters or hyphens. • Its first character must be a letter. • It cannot end with a hyphen or contain two consecutive hyphens. • It must be unique for all DB clusters per AWS account, per AWS Region. 	<p>Using the AWS CLI, run create-db-cluster and set the --db-cluster-identifier option.</p> <p>Using the RDS API, call CreateDBCluster and set the DBClusterIdentifier parameter.</p>
DB cluster parameter group	<p>Choose a DB cluster parameter group. Aurora has a default DB cluster parameter group you can use, or you can create your own DB cluster parameter group. For more information about DB cluster parameter groups, see Working with DB parameter groups and DB cluster parameter groups (p. 339).</p>	<p>Using the AWS CLI, run create-db-cluster and set the --db-cluster-parameter-group-name option.</p> <p>Using the RDS API, call CreateDBCluster and set the DBClusterParameterGroupName parameter.</p>
DB instance class	<p>Applies only to the provisioned capacity type. Choose a DB instance class that defines the processing and memory requirements for each instance in the DB cluster. For more information about DB instance classes, see Aurora DB instance classes (p. 54).</p>	<p>Set this value for every DB instance in your Aurora cluster.</p> <p>Using the AWS CLI, run create-db-instance and set the --db-instance-class option.</p> <p>Using the RDS API, call CreateDBInstance and set the DBInstanceClass parameter.</p>

Console setting	Setting description	CLI option and RDS API parameter
DB parameter group	Choose a parameter group. Aurora has a default parameter group you can use, or you can create your own parameter group. For more information about parameter groups, see Working with DB parameter groups and DB cluster parameter groups (p. 339) .	<p>Set this value for every DB instance in your Aurora cluster.</p> <p>Using the AWS CLI, run create-db-instance and set the --db-parameter-group-name option.</p> <p>Using the RDS API, call CreateDBInstance and set the DBParameterGroupName parameter.</p>
Enable deletion protection	Choose Enable deletion protection to prevent your DB cluster from being deleted. If you create a production DB cluster with the console, deletion protection is enabled by default.	<p>Using the AWS CLI, run create-db-cluster and set the --deletion-protection --no-deletion-protection option.</p> <p>Using the RDS API, call CreateDBCluster and set the DeletionProtection parameter.</p>
Enable encryption	Choose Enable encryption to enable encryption at rest for this DB cluster. For more information, see Encrypting Amazon Aurora resources (p. 1709) .	<p>Using the AWS CLI, run create-db-cluster and set the --storage-encrypted --no-storage-encrypted option.</p> <p>Using the RDS API, call CreateDBCluster and set the StorageEncrypted parameter.</p>
Enable Enhanced Monitoring	Choose Enable enhanced monitoring to enable gathering metrics in real time for the operating system that your DB cluster runs on. For more information, see Monitoring OS metrics with Enhanced Monitoring (p. 626) .	<p>Set these values for every DB instance in your Aurora cluster.</p> <p>Using the AWS CLI, run create-db-instance and set the --monitoring-interval and --monitoring-role-arn options.</p> <p>Using the RDS API, call CreateDBInstance and set the MonitoringInterval and MonitoringRoleArn parameters.</p>

Console setting	Setting description	CLI option and RDS API parameter
Enable Performance Insights	Choose Enable Performance Insights to enable Amazon RDS Performance Insights. For more information, see Monitoring DB load with Performance Insights on Amazon Aurora (p. 573) .	<p>Set these values for every DB instance in your Aurora cluster.</p> <p>Using the AWS CLI, run <code>create-db-instance</code> and set the <code>--enable-performance-insights</code> <code>--no-enable-performance-insights</code>, <code>--performance-insights-kms-key-id</code>, and <code>--performance-insights-retention-period</code> options.</p> <p>Using the RDS API, call <code>CreateDBInstance</code> and set the <code>EnablePerformanceInsights</code>, <code>PerformanceInsightsKMSKeyId</code>, and <code>PerformanceInsightsRetentionPeriod</code> parameters.</p>
Engine type	Choose the database engine to be used for this DB cluster.	<p>Using the AWS CLI, run <code>create-db-cluster</code> and set the <code>--engine</code> option.</p> <p>Using the RDS API, call <code>CreateDBCluster</code> and set the <code>Engine</code> parameter.</p>
Engine version	Applies only to the provisioned capacity type. Choose the version number of your DB engine.	<p>Using the AWS CLI, run <code>create-db-cluster</code> and set the <code>--engine-version</code> option.</p> <p>Using the RDS API, call <code>CreateDBCluster</code> and set the <code>EngineVersion</code> parameter.</p>
Failover priority	Choose a failover priority for the instance. If you don't choose a value, the default is tier-1 . This priority determines the order in which Aurora Replicas are promoted when recovering from a primary instance failure. For more information, see Fault tolerance for an Aurora DB cluster (p. 69) .	<p>Set this value for every DB instance in your Aurora cluster.</p> <p>Using the AWS CLI, run <code>create-db-instance</code> and set the <code>--promotion-tier</code> option.</p> <p>Using the RDS API, call <code>CreateDBInstance</code> and set the <code>PromotionTier</code> parameter.</p>

Console setting	Setting description	CLI option and RDS API parameter
Initial database name	<p>Enter a name for your default database. If you don't provide a name for an Aurora MySQL DB cluster, Amazon RDS doesn't create a database on the DB cluster you are creating. If you don't provide a name for an Aurora PostgreSQL DB cluster, Amazon RDS creates a database named <code>postgres</code>.</p> <p>For Aurora MySQL, the default database name has these constraints:</p> <ul style="list-style-type: none"> • It must contain 1–64 alphanumeric characters. • It can't be a word reserved by the database engine. <p>For Aurora PostgreSQL, the default database name has these constraints:</p> <ul style="list-style-type: none"> • It must contain 1–63 alphanumeric characters. • It must begin with a letter or an underscore. Subsequent characters can be letters, underscores, or digits (0–9). • It can't be a word reserved by the database engine. <p>To create additional databases, connect to the DB cluster and use the SQL command <code>CREATE DATABASE</code>. For more information about connecting to the DB cluster, see Connecting to an Amazon Aurora DB cluster (p. 281).</p>	<p>Using the AWS CLI, run <code>create-db-cluster</code> and set the <code>--database-name</code> option.</p> <p>Using the RDS API, call <code>CreateDBCluster</code> and set the <code>DatabaseName</code> parameter.</p>
Log exports	<p>In the Log exports section, choose the logs that you want to start publishing to Amazon CloudWatch Logs. For more information about publishing Aurora MySQL logs to CloudWatch Logs, see Publishing Amazon Aurora MySQL logs to Amazon CloudWatch Logs (p. 1017). For more information about publishing Aurora PostgreSQL logs to CloudWatch Logs, see Publishing Aurora PostgreSQL logs to Amazon CloudWatch Logs (p. 1487).</p>	<p>Using the AWS CLI, run <code>create-db-cluster</code> and set the <code>--enable-cloudwatch-logs-exports</code> option.</p> <p>Using the RDS API, call <code>CreateDBCluster</code> and set the <code>EnableCloudwatchLogsExports</code> parameter.</p>

Console setting	Setting description	CLI option and RDS API parameter
Maintenance window	Choose Select window and specify the weekly time range during which system maintenance can occur. Or choose No preference for Amazon RDS to assign a period randomly.	Using the AWS CLI, run create-db-cluster and set the --preferred-maintenance-window option. Using the RDS API, call CreateDBCluster and set the PreferredMaintenanceWindow parameter.
Master password	Enter a password to log on to your DB cluster: <ul style="list-style-type: none">• For Aurora MySQL, the password must contain 8–41 printable ASCII characters.• For Aurora PostgreSQL, it must contain 8–128 printable ASCII characters.• It can't contain /, ", @, or a space.	Using the AWS CLI, run create-db-cluster and set the --master-user-password option. Using the RDS API, call CreateDBCluster and set the MasterUserPassword parameter.
Master username	Enter a name to use as the master user name to log on to your DB cluster: <ul style="list-style-type: none">• For Aurora MySQL, the name must contain 1–16 alphanumeric characters.• For Aurora PostgreSQL, it must contain 1–63 alphanumeric characters.• The first character must be a letter.• The name can't be a word reserved by the database engine.	Using the AWS CLI, run create-db-cluster and set the --master-username option. Using the RDS API, call CreateDBCluster and set the MasterUsername parameter.
Multi-AZ deployment	Applies only to the provisioned capacity type. Determine if you want to create Aurora Replicas in other Availability Zones for failover support. If you choose Create Replica in Different Zone , then Amazon RDS creates an Aurora Replica for you in your DB cluster in a different Availability Zone than the primary instance for your DB cluster. For more information about multiple Availability Zones, see Regions and Availability Zones (p. 11) .	Using the AWS CLI, run create-db-cluster and set the --availability-zones option. Using the RDS API, call CreateDBCluster and set the AvailabilityZones parameter.

Console setting	Setting description	CLI option and RDS API parameter
Option group	Aurora has a default option group.	Using the AWS CLI, run create-db-cluster and set the --option-group-name option. Using the RDS API, call CreateDBCluster and set the OptionGroupName parameter.
Public access	<p>Choose Publicly accessible to give the DB cluster a public IP address, or choose Not publicly accessible. The instances in your DB cluster can be a mix of both public and private DB instances. For more information about hiding instances from public access, see Hiding a DB instance in a VPC from the internet (p. 1789).</p> <p>To connect to a DB instance from outside of its Amazon VPC, the DB instance must be publicly accessible, access must be granted using the inbound rules of the DB instance's security group, and other requirements must be met. For more information, see Can't connect to Amazon RDS DB instance (p. 1814).</p> <p>If your DB instance is isn't publicly accessible, you can also use an AWS Site-to-Site VPN connection or an AWS Direct Connect connection to access it from a private network. For more information, see Internetwork traffic privacy (p. 1723).</p>	Set this value for every DB instance in your Aurora cluster. Using the AWS CLI, run create-db-instance and set the --publicly-accessible --no-publicly-accessible option. Using the RDS API, call CreateDBInstance and set the PubliclyAccessible parameter.
Retention period	Choose the length of time, from 1 to 35 days, that Aurora retains backup copies of the database. Backup copies can be used for point-in-time restores (PITR) of your database down to the second.	Using the AWS CLI, run create-db-cluster and set the --backup-retention-period option. Using the RDS API, call CreateDBCluster and set the BackupRetentionPeriod parameter.
Subnet group	Choose the DB subnet group to use for the DB cluster. For more information, see DB cluster prerequisites (p. 125) .	Using the AWS CLI, run create-db-cluster and set the --db-subnet-group-name option. Using the RDS API, call CreateDBCluster and set the DBSubnetGroupName parameter.

Console setting	Setting description	CLI option and RDS API parameter
Virtual Private Cloud (VPC)	Choose the VPC to host the DB cluster. Choose Create a New VPC to have Amazon RDS create a VPC for you. For more information, see DB cluster prerequisites (p. 125) .	For the AWS CLI and API, you specify the VPC security group IDs.
VPC security group	<p>Choose Create new to have Amazon RDS create a VPC security group for you. Or choose Choose existing and specify one or more VPC security groups to secure network access to the DB cluster.</p> <p>When you choose Create new in the RDS console, a new security group is created with an inbound rule that allows access to the DB instance from the IP address detected in your browser.</p> <p>For more information, see DB cluster prerequisites (p. 125).</p>	<p>Using the AWS CLI, run <code>create-db-cluster</code> and set the <code>--vpc-security-group-ids</code> option.</p> <p>Using the RDS API, call <code>CreateDBCluster</code> and set the <code>VpcSecurityGroupIds</code> parameter.</p>

Creating Amazon Aurora resources with AWS CloudFormation

Amazon Aurora is integrated with AWS CloudFormation, a service that helps you to model and set up your AWS resources so that you can spend less time creating and managing your resources and infrastructure. You create a template that describes all the AWS resources that you want (such as DB clusters and DB cluster parameter groups), and AWS CloudFormation provisions and configures those resources for you.

When you use AWS CloudFormation, you can reuse your template to set up your Aurora resources consistently and repeatedly. Describe your resources once, and then provision the same resources over and over in multiple AWS accounts and Regions.

Aurora and AWS CloudFormation templates

To provision and configure resources for Aurora and related services, you must understand [AWS CloudFormation templates](#). Templates are formatted text files in JSON or YAML. These templates describe the resources that you want to provision in your AWS CloudFormation stacks. If you're unfamiliar with JSON or YAML, you can use AWS CloudFormation Designer to help you get started with AWS CloudFormation templates. For more information, see [What is AWS CloudFormation Designer?](#) in the [AWS CloudFormation User Guide](#).

Aurora supports creating resources in AWS CloudFormation. For more information, including examples of JSON and YAML templates for these resources, see the [RDS resource type reference](#) in the [AWS CloudFormation User Guide](#).

Learn more about AWS CloudFormation

To learn more about AWS CloudFormation, see the following resources:

- [AWS CloudFormation](#)
- [AWS CloudFormation User Guide](#)
- [AWS CloudFormation API Reference](#)
- [AWS CloudFormation Command Line Interface User Guide](#)

Using Amazon Aurora Serverless v1

Amazon Aurora Serverless v1 (Amazon Aurora Serverless version 1) is an on-demand autoscaling configuration for Amazon Aurora. An *Aurora Serverless DB cluster* is a DB cluster that scales compute capacity up and down based on your application's needs. This contrasts with Aurora *provisioned DB clusters*, for which you manually manage capacity. Aurora Serverless v1 provides a relatively simple, cost-effective option for infrequent, intermittent, or unpredictable workloads. It is cost-effective because it automatically starts up, scales compute capacity to match your application's usage, and shuts down when it's not in use.

To learn more about pricing, see [Serverless Pricing](#) under **MySQL-Compatible Edition** or **PostgreSQL-Compatible Edition** on the Amazon Aurora pricing page.

Aurora Serverless v1 clusters have the same kind of high-capacity, distributed, and highly available storage volume that is used by provisioned DB clusters. The cluster volume for an Aurora Serverless v1 cluster is always encrypted. You can choose the encryption key, but you can't disable encryption. That means that you can perform the same operations on an Aurora Serverless v1 that you can on encrypted snapshots. For more information, see [Aurora Serverless v1 and snapshots \(p. 161\)](#).

Topics

- [Advantages of Aurora Serverless v1 \(p. 147\)](#)
- [Use cases for Aurora Serverless v1 \(p. 148\)](#)
- [Limitations of Aurora Serverless v1 \(p. 148\)](#)
- [Configuration requirements for Aurora Serverless v1 \(p. 149\)](#)
- [Using TLS/SSL with Aurora Serverless v1 \(p. 150\)](#)
- [How Aurora Serverless v1 works \(p. 151\)](#)
- [Creating an Aurora Serverless v1 DB cluster \(p. 161\)](#)
- [Restoring an Aurora Serverless v1 DB cluster \(p. 166\)](#)
- [Modifying an Aurora Serverless v1 DB cluster \(p. 170\)](#)
- [Scaling Aurora Serverless v1 DB cluster capacity manually \(p. 172\)](#)
- [Viewing Aurora Serverless v1 DB clusters \(p. 174\)](#)
- [Deleting an Aurora Serverless v1 DB cluster \(p. 175\)](#)
- [Aurora Serverless v1 and Aurora database engine versions \(p. 177\)](#)
- [Using the Data API for Aurora Serverless \(p. 178\)](#)
- [Logging Data API calls with AWS CloudTrail \(p. 202\)](#)
- [Using the query editor for Aurora Serverless \(p. 204\)](#)

Advantages of Aurora Serverless v1

Aurora Serverless v1 provides the following advantages:

- **Simpler than provisioned** – Aurora Serverless v1 removes much of the complexity of managing DB instances and capacity.
- **Scalable** – Aurora Serverless v1 seamlessly scales compute and memory capacity as needed, with no disruption to client connections.
- **Cost-effective** – When you use Aurora Serverless v1, you pay only for the database resources that you consume, on a per-second basis.
- **Highly available storage** – Aurora Serverless v1 uses the same fault-tolerant, distributed storage system with six-way replication as Aurora to protect against data loss.

Use cases for Aurora Serverless v1

Aurora Serverless v1 is designed for the following use cases:

- **Infrequently used applications** – You have an application that is only used for a few minutes several times per day or week, such as a low-volume blog site. With Aurora Serverless v1, you pay for only the database resources that you consume on a per-second basis.
- **New applications** – You're deploying a new application and you're unsure about the instance size you need. By using Aurora Serverless v1, you can create a database endpoint and have the database autoscale to the capacity requirements of your application.
- **Variable workloads** – You're running a lightly used application, with peaks of 30 minutes to several hours a few times each day, or several times per year. Examples are applications for human resources, budgeting, and operational reporting applications. With Aurora Serverless v1, you no longer need to provision for peak or average capacity.
- **Unpredictable workloads** – You're running daily workloads that have sudden and unpredictable increases in activity. An example is a traffic site that sees a surge of activity when it starts raining. With Aurora Serverless v1, your database autoscales capacity to meet the needs of the application's peak load and scales back down when the surge of activity is over.
- **Development and test databases** – Your developers use databases during work hours but don't need them on nights or weekends. With Aurora Serverless v1, your database automatically shuts down when it's not in use.
- **Multi-tenant applications** – With Aurora Serverless v1, you don't have to individually manage database capacity for each application in your fleet. Aurora Serverless v1 manages individual database capacity for you.

Limitations of Aurora Serverless v1

The following limitations apply to Aurora Serverless v1:

- You can't use Aurora MySQL version 3 for Aurora Serverless v1 clusters. Aurora MySQL version 3 works with Aurora Serverless v2, which is currently in preview.
- Aurora Serverless v1 is available in certain AWS Regions and for specific Aurora MySQL and Aurora PostgreSQL versions only. For more information, see [Aurora Serverless v1 \(p. 29\)](#).
- Aurora Serverless v1 doesn't support the following features:
 - Aurora global databases
 - Aurora multi-master clusters
 - Aurora Replicas
 - AWS Identity and Access Management (IAM) database authentication
 - Backtracking in Aurora
 - Database activity streams
 - Performance Insights
- Connections to an Aurora Serverless v1 DB cluster are closed automatically if held open for longer than one day.
- All Aurora Serverless v1 DB clusters have the following limitations:
 - You can't export Aurora Serverless v1 snapshots to Amazon S3 buckets.
 - You can't save data to text files in Amazon S3.
 - You can't use AWS Database Migration Service and Change Data Capture (CDC) with Aurora Serverless DB clusters. Only provisioned Aurora DB clusters support CDC with AWS DMS as a source.
 - You can't load text file data to Aurora MySQL Serverless from Amazon S3. However, you can load data to Aurora PostgreSQL Serverless from Amazon S3 by using the `aws_s3` extension with

the `aws_s3.table_import_from_s3` function and the `credentials` parameter. For more information, see [Importing Amazon S3 data into an Aurora PostgreSQL DB cluster \(p. 1438\)](#).

- Aurora MySQL-based DB clusters running Aurora Serverless v1 don't support the following:
 - Invoking AWS Lambda functions from within your Aurora MySQL DB cluster. However, AWS Lambda functions can make calls to your Aurora MySQL Serverless DB cluster.
 - Restoring a snapshot from a DB instance that isn't Aurora MySQL or RDS for MySQL.
 - Replicating data using replication based on binary logs (binlogs). This limitation is true regardless of whether your Aurora MySQL-based DB cluster Aurora Serverless v1 is the source or the target of the replication. To replicate data into an Aurora Serverless v1 DB cluster from a MySQL DB instance outside Aurora, such as one running on Amazon EC2, consider using AWS Database Migration Service. For more information, see the [AWS Database Migration Service User Guide](#).
- Aurora PostgreSQL-based DB clusters running Aurora Serverless v1 have the following limitations:
 - Aurora PostgreSQL query plan management (`apg_plan_management` extension) isn't supported.
 - The logical replication feature available in Amazon RDS PostgreSQL and Aurora PostgreSQL isn't supported.
 - Outbound communications such as those enabled by Amazon RDS for PostgreSQL extensions aren't supported. For example, you can't access external data with the `postgres_fdw/dblink` extension. For more information about RDS PostgreSQL extensions, see [PostgreSQL on Amazon RDS](#) in the *RDS User Guide*.
 - Currently, certain SQL queries and commands aren't recommended. These include session-level advisory locks, temporary relations, asynchronous notifications (`LISTEN`), and cursors with hold (`DECLARE name ... CURSOR WITH HOLD FOR query`). Also, `NOTIFY` and `COPY` commands prevent scaling and aren't recommended.

For more information, see [Autoscaling for Aurora Serverless v1 \(p. 153\)](#).

- You can't set the preferred backup window for an Aurora Serverless v1 DB cluster.

Configuration requirements for Aurora Serverless v1

When you create an Aurora Serverless v1 DB cluster, pay attention to the following requirements:

- Use these specific port numbers for each DB engine:
 - Aurora MySQL – 3306
 - Aurora PostgreSQL – 5432
- Create your Aurora Serverless v1 DB cluster in a virtual private cloud (VPC) based on the Amazon VPC service. When you create an Aurora Serverless v1 DB cluster in your VPC, you consume two (2) of the fifty (50) Interface and Gateway Load Balancer endpoints allotted to your VPC. These endpoints are created automatically for you. To increase your quota, you can contact AWS Support. For more information, see [Amazon VPC quotas](#).
- You can't give an Aurora Serverless v1 DB cluster a public IP address. You can access an Aurora Serverless v1 DB cluster only from within a VPC.
- Create subnets in different Availability Zones for the DB subnet group that you use for your Aurora Serverless v1 DB cluster. In other words, you can't have more than one subnet in the same Availability Zone.
- Changes to a subnet group used by an Aurora Serverless v1 DB cluster aren't applied to the cluster.
- You can access an Aurora Serverless v1 DB cluster from AWS Lambda. To do so, you must configure your Lambda function to run in the same VPC as your Aurora Serverless v1 DB cluster. For more information about working with AWS Lambda, see [Configuring a Lambda function to access resources in an Amazon VPC](#) in the *AWS Lambda Developer Guide*.

Using TLS/SSL with Aurora Serverless v1

By default, Aurora Serverless v1 uses the Transport Layer Security/Secure Sockets Layer (TLS/SSL) protocol to encrypt communications between clients and your Aurora Serverless v1 DB cluster. It supports TLS/SSL versions 1.0, 1.1, and 1.2. You don't need to configure your Aurora Serverless v1 DB cluster to use TLS/SSL.

However, the following limitations apply:

- TLS/SSL support for Aurora Serverless v1 DB clusters isn't currently available in the China (Beijing) AWS Region.
- When you create database users for an Aurora MySQL-based Aurora Serverless v1 DB cluster, don't use the `REQUIRE` clause for SSL permissions. Doing so prevents users from connecting to the Aurora DB instance.
- For both MySQL Client and PostgreSQL Client utilities, session variables that you might use in other environments have no effect when using TLS/SSL between client and Aurora Serverless v1.
- For the MySQL Client, when connecting with TLS/SSL's `VERIFY_IDENTITY` mode, currently you need to use the MySQL 8.0-compatible `mysql` command. For more information, see [Connecting to a DB instance running the MySQL database engine](#).

Depending on the client that you use to connect to Aurora Serverless v1 DB cluster, you might not need to specify TLS/SSL to get an encrypted connection. For example, to use the PostgreSQL Client to connect to an Aurora Serverless v1 DB cluster running Aurora PostgreSQL-Compatible Edition, connect as you normally do.

```
psql -h endpoint -U user
```

After you enter your password, the PostgreSQL Client shows you see the connection details, including the TLS/SSL version and cipher.

```
psql (12.5 (Ubuntu 12.5-0ubuntu0.20.04.1), server 10.12)
SSL connection (protocol: TLSv1.2, cipher: ECDHE-RSA-AES256-GCM-SHA384, bits: 256,
compression: off)
Type "help" for help.
```

Important

Aurora Serverless v1 uses the Transport Layer Security/Secure Sockets Layer (TLS/SSL) protocol to encrypt connections by default unless SSL/TLS is disabled by the client application. The TLS/SSL connection terminates at the router fleet. Communication between the router fleet and your Aurora Serverless v1 DB cluster occurs within the service's internal network boundary.

You can check the status of the client connection to examine whether the connection to Aurora Serverless v1 is TLS/SSL encrypted. The PostgreSQL `pg_stat_ssl` and `pg_stat_activity` tables and its `ssl_is_used` function don't show the TLS/SSL state for the communication between the client application and Aurora Serverless v1. Similarly, the TLS/SSL state can't be derived from the MySQL `status` statement.

The Aurora cluster parameters `force_ssl` for PostgreSQL and `require_secure_transport` for MySQL aren't supported for Aurora Serverless v1. For a complete list of parameters supported by Aurora Serverless v1, call the [DescribeEngineDefaultClusterParameters](#) API. For more information on parameter groups and Aurora Serverless v1, see [Parameter groups and Aurora Serverless v1 \(p. 156\)](#).

To use the MySQL Client to connect to an Aurora Serverless v1 DB cluster running Aurora MySQL-Compatible Edition, you specify TLS/SSL in your request. The following example includes the [Amazon root CA 1 trust store](#) downloaded from Amazon Trust Services, which is necessary for this connection to succeed.

```
mysql -h endpoint -P 3306 -u user -p --ssl-ca=amazon-root-CA-1.pem --ssl-mode=REQUIRED
```

When prompted, enter your password. Soon, the MySQL monitor opens. You can confirm that the session is encrypted by using the `status` command.

```
mysql> status
-----
mysql Ver 14.14 Distrib 5.5.62, for Linux (x86_64) using readline 5.1
Connection id:          19
Current database:
Current user:          ***@*****
SSL:                  Cipher in use is ECDHE-RSA-AES256-SHA
...
```

To learn more about connecting to Aurora MySQL database with the MySQL Client, see [Connecting to a DB instance running the MySQL database engine](#).

Aurora Serverless v1 supports all TLS/SSL modes available to the MySQL Client (`mysql`) and PostgreSQL Client (`psql`), including those listed in the following table.

Description of TLS/SSL mode	<code>mysql</code>	<code>psql</code>
Connect without using TLS/SSL.	DISABLED	disable
Try the connection using TLS/SSL first, but fall back to non-SSL if necessary.	PREFERRED	prefer (default)
Enforce using TLS/SSL.	REQUIRED	require
Enforce TLS/SSL and verify the CA.	VERIFY_CA	verify-ca
Enforce TLS/SSL, verify the CA, and verify the CA hostname.	VERIFY_IDENTITY	verify-full

Aurora Serverless v1 uses wildcard certificates. If you specify the "verify CA" or the "verify CA and CA hostname" option when using TLS/SSL, first download the [Amazon root CA 1 trust store](#) from Amazon Trust Services. After doing so, you can identify this PEM-formatted file in your client command. To do so using the PostgreSQL Client:

For Linux, macOS, or Unix:

```
psql 'host=endpoint user=user sslmode=require sslrootcert=amazon-root-CA-1.pem dbname=db-name'
```

To learn more about working with the Aurora PostgreSQL database using the Postgres Client, see [Connecting to a DB instance running the PostgreSQL database engine](#).

For more information about connecting to Aurora DB clusters in general, see [Connecting to an Amazon Aurora DB cluster \(p. 281\)](#).

How Aurora Serverless v1 works

Amazon Aurora offers two different DB engine modes aimed at two broadly different usage models.

The *provisioned* DB engine mode is designed for predictable workloads. When you work with Aurora provisioned DB clusters, you choose your DB instance class size and several other configuration options. For example, you can create one or more Aurora Replicas to increase read throughput. If your workload changes, you can modify the DB instance class size and change the number of Aurora Replicas. The provisioned model works well when you can adjust capacity in advance of expected consumption patterns.

The *serverless* DB engine mode is designed for a different usage pattern entirely. For example, your database usage might be heavy for a short period of time, followed by long periods of light activity or no activity at all. Some examples are retail websites with intermittent sales events, databases that produce reports when needed, development and testing environments, and new applications with uncertain requirements. For cases such as these and many others, configuring capacity correctly in advance isn't always possible with the provisioned model. It can also result in higher costs if you overprovision and have capacity that you don't use.

By using Aurora Serverless v1, you can create a database endpoint without specifying the DB instance class size. You specify only the minimum and maximum range for the Aurora Serverless v1 DB cluster's capacity. The Aurora Serverless v1 database endpoint makes up a *router fleet* that supports continuous connections and distributes the workload among resources. Aurora Serverless v1 scales the resources automatically based on your minimum and maximum capacity specifications.

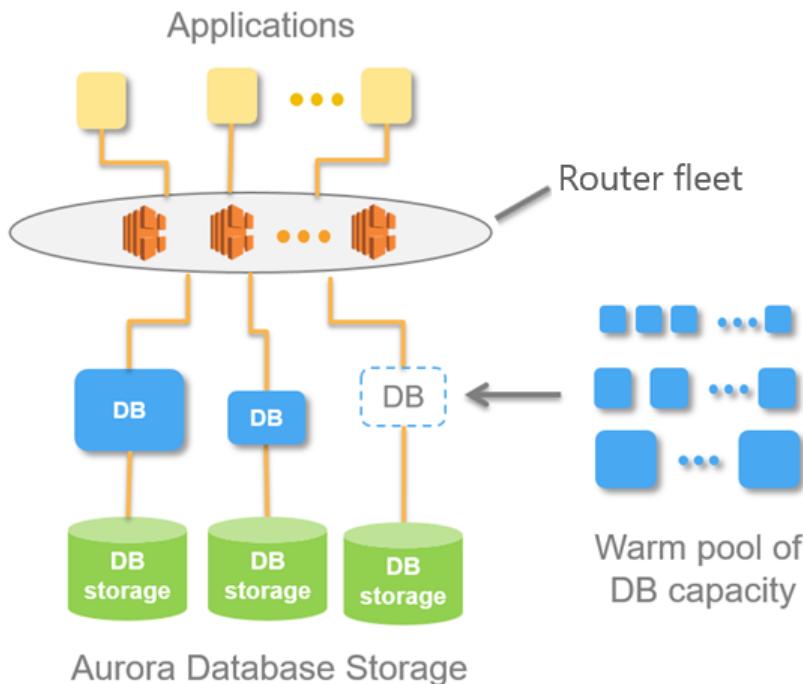
You don't need to change your database client application code to use the router fleet. Aurora Serverless v1 manages the connections automatically. Scaling is fast thanks to a "warm" resources pool that's always ready to service requests. Storage and processing are separate, so your Aurora Serverless v1 DB cluster can scale down to zero when it's finished processing workloads. When your Aurora Serverless v1 DB cluster scales to zero, you're charged only for storage.

Topics

- [Aurora Serverless v1 architecture \(p. 152\)](#)
- [Autoscaling for Aurora Serverless v1 \(p. 153\)](#)
- [Timeout action for capacity changes \(p. 154\)](#)
- [Pause and resume for Aurora Serverless v1 \(p. 155\)](#)
- [Parameter groups and Aurora Serverless v1 \(p. 156\)](#)
- [Logging for Aurora Serverless v1 \(p. 158\)](#)
- [Aurora Serverless v1 and maintenance \(p. 160\)](#)
- [Aurora Serverless v1 and failover \(p. 160\)](#)
- [Aurora Serverless v1 and snapshots \(p. 161\)](#)

Aurora Serverless v1 architecture

The following image shows an overview the Aurora Serverless v1 architecture.



Instead of provisioning and managing database servers, you specify Aurora capacity units (ACUs). Each ACU is a combination of approximately 2 gigabytes (GB) of memory, corresponding CPU, and networking. Database storage automatically scales from 10 gibibytes (GiB) to 128 tebibytes (TiB), the same as storage in a standard Aurora DB cluster.

You can specify the minimum and maximum ACU. The *minimum Aurora capacity unit* is the lowest ACU to which the DB cluster can scale down. The *maximum Aurora capacity unit* is the highest ACU to which the DB cluster can scale up. Based on your settings, Aurora Serverless v1 automatically creates scaling rules for thresholds for CPU utilization, connections, and available memory.

Aurora Serverless v1 manages the warm pool of resources in an AWS Region to minimize scaling time. When Aurora Serverless v1 adds new resources to the Aurora DB cluster, it uses the router fleet to switch active client connections to the new resources. At any specific time, you are only charged for the ACUs that are being actively used in your Aurora DB cluster.

Autoscaling for Aurora Serverless v1

The capacity allocated to your Aurora Serverless v1 DB cluster seamlessly scales up and down based on the load generated by your client application. Here, load is CPU utilization and the number of connections. When capacity is constrained by either of these, Aurora Serverless v1 scales up. Aurora Serverless also scales up when it detects performance issues that can be resolved by doing so.

You can view scaling events for your Aurora Serverless cluster in the AWS Management Console. During autoscaling, Aurora Serverless v1 resets the `EngineUptime` metric. The value of the reset metric value doesn't mean that seamless scaling had problems, nor does it mean Aurora Serverless dropped connections. It's simply the starting point for uptime at the new capacity. To learn more about metrics, see [Monitoring metrics in an Amazon Aurora cluster \(p. 541\)](#).

When your Aurora Serverless v1 DB cluster has no active connections, it can scale down to zero capacity (0 ACUs). To learn more, see [Pause and resume for Aurora Serverless v1 \(p. 155\)](#).

When it does need to perform a scaling operation, Aurora Serverless v1 first tries to identify a *scaling point*, a moment when no queries are being processed. Aurora Serverless might not be able to find a scaling point for the following reasons:

- Long-running queries
- In-progress transactions
- Temporary tables or table locks

To increase your Aurora Serverless DB cluster's success rate when finding a scaling point, we recommend that you avoid long-running queries and long-running transactions. To learn more about scale-blocking operations and how to avoid them, see [Best practices for working with Amazon Aurora Serverless](#).

By default, Aurora Serverless v1 tries to find a scaling point for 5 minutes (300 seconds). You can specify a different timeout period when you create or modify the cluster. The timeout period can be between 60 seconds and 10 minutes (600 seconds). If Aurora Serverless can't find a scaling point within the specified period, the autoscaling operation times out.

By default, if autoscaling doesn't find a scaling point before timing out, Aurora Serverless v1 keeps the cluster at the current capacity. You can change this default behavior when you create or modify your Aurora Serverless DB cluster by selecting the **Force the capacity change** option. For more information, see [Timeout action for capacity changes \(p. 154\)](#).

Timeout action for capacity changes

If autoscaling times out without finding a scaling point, by default Aurora keeps the current capacity. You can choose to have Aurora force the change by enabling the **Force the capacity change** option. This option is available in the **Autoscaling timeout and action** section of the Create database page, when you create the cluster.

- **Force the capacity change** – By default, this option is deselected. Leave this option unchecked to have your Aurora Serverless DB cluster's capacity to remain unchanged if the scaling operation times out without finding a scaling point.
- **Force the capacity change** – Choosing this option causes your Aurora Serverless DB cluster to enforce the capacity change, even without a scaling point. Before enabling this option, be aware of the consequences of this choice.
 - Any in-process transactions are interrupted, and the following error message appears.

Aurora MySQL 5.6 – ERROR 1105 (HY000): The last transaction was aborted due to an unknown error. Please retry.

Aurora MySQL 5.7 – ERROR 1105 (HY000): The last transaction was aborted due to Seamless Scaling. Please retry.

You can resubmit the transactions as soon as your Aurora Serverless v1 DB cluster is available.

- Connections to temporary tables and locks are dropped.

Note

We recommend that you choose the "force" option only if your application can recover from dropped connections or incomplete transactions.

The choices you make in the AWS Management Console when you create an Aurora Serverless DB cluster are stored in the `ScalingConfigurationInfo` object, in the `SecondsBeforeTimeout` and `TimeoutAction` properties. The value of the `TimeoutAction` property is set to one of the following values when you create your cluster:

- `RollbackCapacityChange` – This value is set when you choose the **Roll back the capacity change** option. This is the default behavior.
- `ForceApplyCapacityChange` – This value is set when you choose the **Force the capacity change** option.

You can get the value of this property on an existing Aurora Serverless DB cluster by using the [describe-db-clusters](#) AWS CLI command, as shown following.

For Linux, macOS, or Unix:

```
aws rds describe-db-clusters --region region \  
  --db-cluster-identifier your-cluster-name \  
  --query '*[].{ScalingConfigurationInfo:ScalingConfigurationInfo}'
```

For Windows:

```
aws rds describe-db-clusters --region region ^  
  --db-cluster-identifier your-cluster-name ^  
  --query "*[].{ScalingConfigurationInfo:ScalingConfigurationInfo}"
```

As an example, the following shows the query and response for an Aurora Serverless v1 DB cluster named `west-coast-sles` in the US West (N. California) Region.

```
$ aws rds describe-db-clusters --region us-west-1 --db-cluster-identifier west-coast-sles  
--query '*[].{ScalingConfigurationInfo:ScalingConfigurationInfo}'  
  
[  
  {  
    "ScalingConfigurationInfo": {  
      "MinCapacity": 1,  
      "MaxCapacity": 64,  
      "AutoPause": false,  
      "SecondsBeforeTimeout": 300,  
      "SecondsUntilAutoPause": 300,  
      "TimeoutAction": "RollbackCapacityChange"  
    }  
  }  
]
```

As the response shows, this Aurora Serverless v1 DB cluster uses the default setting.

For more information, see [Creating an Aurora Serverless v1 DB cluster \(p. 161\)](#). After creating your Aurora Serverless v1, you can modify the timeout action and other capacity settings at any time. To learn how, see [Modifying an Aurora Serverless v1 DB cluster \(p. 170\)](#).

Pause and resume for Aurora Serverless v1

You can choose to pause your Aurora Serverless v1 DB cluster after a given amount of time with no activity. You specify the amount of time with no activity before the DB cluster is paused. When you select this option, the default inactivity time is five minutes, but you can change this value. This is an optional setting.

When the DB cluster is paused, no compute or memory activity occurs, and you are charged only for storage. If database connections are requested when an Aurora Serverless DB cluster is paused, the DB cluster automatically resumes and services the connection requests.

When the DB cluster resumes activity, it has the same capacity as it had when Aurora paused the cluster. The number of ACUs depends on how much Aurora scaled the cluster up or down before pausing it.

Note

If a DB cluster is paused for more than seven days, the DB cluster might be backed up with a snapshot. In this case, Aurora restores the DB cluster from the snapshot when there is a request to connect to it.

Parameter groups and Aurora Serverless v1

When you create your Aurora Serverless v1 DB cluster, you choose a specific Aurora DB engine and an associated DB cluster parameter group. Unlike provisioned Aurora DB clusters, an Aurora Serverless DB cluster has a single read/write DB instance that's configured with a DB cluster parameter group only—it doesn't have a separate DB parameter group. During autoscaling, Aurora Serverless needs to be able to change parameters for the cluster to work best for the increased or decreased capacity. Thus, with an Aurora Serverless DB cluster, some of the changes you might make to parameters for a particular DB engine type might not apply.

For example, an Aurora PostgreSQL-based Aurora Serverless DB cluster can't use `apg_plan_mgmt.capture_plan_baselines` and other parameters that might be used on provisioned Aurora PostgreSQL DB clusters for query plan management.

You can get a list of default values for the default parameter groups for the various Aurora DB engines by using the `describe-engine-default-cluster-parameters` CLI command and querying the AWS Region. The following are values you can use for the `--db-parameter-group-family` option.

Aurora MySQL 5.6	<code>aurora5.6</code>
Aurora MySQL 5.7	<code>aurora-mysql5.7</code>
Aurora PostgreSQL 10.12 (and later)	<code>aurora-postgresql10</code>

We recommend that you configure your AWS CLI with your AWS Access Key ID and AWS Secret Access Key, and that you set your AWS Region before using AWS CLI commands. Providing the Region to your CLI configuration saves you from entering the `--region` parameter when running commands. To learn more about configuring AWS CLI, see [Configuration basics](#) in the AWS Command Line Interface User Guide.

The following example gets a list of parameters from the default DB cluster group for Aurora MySQL 5.6.

For Linux, macOS, or Unix:

```
aws rds describe-engine-default-cluster-parameters \
--db-parameter-group-family aurora5.6 --query \
'EngineDefaults.Parameters[*]' \
{ParameterName:ParameterName,SupportedEngineModes:SupportedEngineModes} | [?
contains(SupportedEngineModes, `serverless`) == `true`] | [*].{param:ParameterName}' \
--output text
```

For Windows:

```
aws rds describe-engine-default-cluster-parameters ^
--db-parameter-group-family aurora5.6 --query ^
"EngineDefaults.Parameters[*]" \
{ParameterName:ParameterName,SupportedEngineModes:SupportedEngineModes} | [?
contains(SupportedEngineModes, 'serverless') == 'true'] | [*].{param:ParameterName}" ^
--output text
```

Modifying parameter values for Aurora Serverless v1

As explained in [Working with DB parameter groups and DB cluster parameter groups \(p. 339\)](#), you can't directly change values in a default parameter group, regardless of its type (DB cluster parameter group, DB parameter group). Instead, you create a custom parameter group based on the default DB cluster

parameter group for your Aurora DB engine and change settings as needed on that parameter group. For example, you might want to change some of the settings for your Aurora Serverless DB cluster to [log queries or to upload DB engine specific logs \(p. 158\)](#) to Amazon CloudWatch.

To create a custom DB cluster parameter group

1. Sign in to the AWS Management Console and then open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. Choose **Parameter groups**.
3. Choose **Create parameter group** to open the Parameter group details pane.
4. Choose the appropriate default DB cluster group for the DB engine you want to use for your Aurora Serverless v1 DB cluster. Be sure you choose the following options:
 - a. For **Parameter group family**, choose the appropriate family for your chosen DB engine. Be sure your selection has the prefix `aurora-` in its name.
 - b. For **Type**, choose **DB Cluster Parameter Group**.
 - c. For **Group name** and **Description**, enter meaningful names for you or others who might need to work with your Aurora Serverless v1 DB cluster and its parameters.
 - d. Choose **Create**.

Your custom DB cluster parameter group is added to the list of parameter groups available in your AWS Region. You can use your custom DB cluster parameter group when you create new Aurora Serverless DB clusters, and you can modify an existing Aurora Serverless DB cluster to use your custom DB cluster parameter group. Once your Aurora Serverless DB cluster starts using your custom DB cluster parameter group, you can change values for dynamic parameters using either the AWS Management Console or the AWS CLI. You can also use the Console to view a side-by-side comparison of the values in your custom DB cluster parameter group compared to the default DB cluster parameter group, as shown in the following screenshot.

Parameter	my-db-cluster-param-group-for-mysql-logs	default.aurora-mysql5.7
general_log	1	<engine-default>
log_queries_not_using_indexes	1	<engine-default>
long_query_time	60	<engine-default>
server_audit_events	CONNECT	<engine-default>
server_audit_logging	1	0
server_audit_logs_upload	1	0
slow_query_log	1	<engine-default>

Close

When you change parameter values on an active DB cluster, Aurora Serverless starts a seamless scale in order to apply the parameter changes. If your Aurora Serverless DB cluster is in a "paused" state, it

resumes and starts scaling so that it can make the change. The scaling operation for a parameter group change always [forces the capacity change \(p. 154\)](#), so be aware that modifying parameters might result in dropped connections if a scaling point can't be found during the scaling period.

Logging for Aurora Serverless v1

By default, error logs for Aurora Serverless are enabled and automatically uploaded to Amazon CloudWatch. You can also have your Aurora Serverless DB cluster upload Aurora database-engine specific logs to CloudWatch by enabling configuration parameters in your custom DB cluster parameter group. Your Aurora Serverless DB cluster then uploads all available logs to Amazon CloudWatch, and you can use CloudWatch to analyze log data, create alarms, and view metrics.

For Aurora MySQL, you can enable the following logs have them automatically uploaded from your Aurora Serverless DB cluster to Amazon CloudWatch.

Aurora MySQL	Description
general_log	Creates the general log. Set to 1 to turn on. Default is off (0).
log_queries_not_using_indexes	Logs any queries to the slow query log that don't use an index. Default is off (0). Set to 1 to turn on this log.
long_query_time	Prevents fast-running queries from being logged in the slow query log. Can be set to a float between 0 and 31536000. Default is 0 (not active).
server_audit_events	The list of events to capture in the logs. Supported values are CONNECT, QUERY, QUERY_DCL, QUERY_DDL, QUERY_DML, and TABLE.
server_audit_logging	Set to 1 to turn on server audit logging. If you turn this on, you can specify the audit events to send to CloudWatch by listing them in the server_audit_events parameter.
slow_query_log	Creates a slow query log. Set to 1 to turn on the slow query log. Default is off (0).

For more information, see [Using Advanced Auditing with an Amazon Aurora MySQL DB cluster \(p. 914\)](#).

For Aurora PostgreSQL, you can enable the following logs on your Aurora Serverless DB cluster and have them automatically uploaded to Amazon CloudWatch along with the regular error logs.

Aurora PostgreSQL	Description
log_connections	Enabled by default, and can't be changed. It logs details for all new client connections.
log_disconnections	Enabled by default, and can't be changed. Logs all client disconnections.
log_lock_waits	Default is 0 (off). Set to 1 to log lock waits.

Aurora PostgreSQL	Description
<code>log_min_duration_statement</code>	The minimum duration (in milliseconds) for a statement to run before it's logged.
<code>log_min_messages</code>	Sets the message levels that are logged. Supported values are <code>debug5</code> , <code>debug4</code> , <code>debug3</code> , <code>debug2</code> , <code>debug1</code> , <code>info</code> , <code>notice</code> , <code>warning</code> , <code>error</code> , <code>log</code> , <code>fatal</code> , <code>panic</code> . To log performance data to the <code>postgres</code> log, set the value to <code>debug1</code> .
<code>log_temp_files</code>	Logs the use of temporary files that are above the specified kilobytes (kB).
<code>log_statement</code>	Controls the specific SQL statements that get logged. Supported values are <code>none</code> , <code>ddl</code> , <code>mod</code> , and <code>all</code> . Default is <code>none</code> .

After you enable logs for Aurora MySQL 5.6, Aurora MySQL 5.7, or Aurora PostgreSQL for your Aurora Serverless DB cluster, you can view the logs in CloudWatch.

Viewing Aurora Serverless v1 logs with Amazon CloudWatch

Aurora Serverless v1 automatically uploads ("publishes") to Amazon CloudWatch all logs that are enabled in your custom DB cluster parameter group. You don't need to choose or specify the log types. Uploading logs starts as soon as you enable the log configuration parameter. If you later disable the log parameter, further uploads stop. However, all the logs that have already been published to CloudWatch remain until you delete them.

For more information on using CloudWatch with Aurora MySQL logs, see [Monitoring log events in Amazon CloudWatch \(p. 1020\)](#).

For more information about CloudWatch and Aurora PostgreSQL, see [Publishing Aurora PostgreSQL logs to Amazon CloudWatch Logs \(p. 1487\)](#).

To view logs for your Aurora Serverless DB cluster

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. Choose your AWS Region.
3. Choose **Log groups**.
4. Choose your Aurora Serverless DB cluster log from the list. For error logs, the naming pattern is as follows.

```
/aws/rds/cluster/cluster-name/error
```

For example, in the following screenshot you can find listings for logs published for an Aurora PostgreSQL Aurora Serverless DB cluster named "western-sles." You can also find several listings for Aurora MySQL Aurora Serverless DB cluster, "west-coast-sles." Choose the log of interest to start exploring its content.

The screenshot shows the AWS CloudWatch Log Groups interface. At the top, there are navigation links: CloudWatch > CloudWatch Logs > Log groups. Below this, a header bar includes 'Log groups (5)', a search bar, and buttons for Actions, View in Logs Insights, and Create log group. A note says 'By default, we only load up to 10000 log groups.' A search bar with a placeholder 'Filter log groups or try prefix search' is followed by an 'Exact match' checkbox and navigation arrows. The main table lists five log groups:

Log group	Retention	Metric filters	Contributor Insights
/aws/rds/cluster/west-coast-sles/audit	Never expire	-	-
/aws/rds/cluster/west-coast-sles/error	Never expire	-	-
/aws/rds/cluster/west-coast-sles/general	Never expire	-	-
/aws/rds/cluster/western-sles/postgresql	Never expire	-	-

Aurora Serverless v1 and maintenance

Maintenance for Aurora Serverless v1 DB cluster, such as applying the latest features, fixes, and security updates, is performed automatically for you. Unlike provisioned Aurora DB clusters, Aurora Serverless doesn't have user-settable maintenance windows. However, it does have a maintenance window that you can view in the AWS Management Console in **Maintenance & backups** for your Aurora Serverless DB cluster. You can find the date and time that maintenance might be performed and if any maintenance is pending for your Aurora Serverless DB cluster, as shown following.

The screenshot shows the AWS Management Console with the 'Maintenance & backups' tab selected. The interface includes tabs for Connectivity & security, Monitoring, Logs & events, Configuration, Maintenance & backups (which is highlighted in red), and Tags. Under the Maintenance section, it shows:

Maintenance window	Pending maintenance
tue:08:41-tue:09:11 UTC (GMT)	none

Whenever possible, Aurora Serverless performs maintenance in a non-disruptive manner. When maintenance is required, your Aurora Serverless DB cluster scales its capacity to handle the necessary operations. Before scaling, Aurora Serverless looks for a scaling point and it does so for up to seven days if necessary.

At the end of each day that Aurora Serverless can't find a scaling point, it creates a cluster event. This event notifies you of the pending maintenance and the need to scale to perform maintenance. The notification includes the date when the Aurora Serverless can force the DB cluster to scale.

Until that time, your Aurora Serverless DB cluster continues looking for a scaling point and behaves according to its `TimeoutAction` setting. That is, if it can't find a scaling point before timing out, it abandons the capacity change if it's configured to `RollbackCapacityChange`. Or it forces the change if it's set to `ForceApplyCapacityChange`. As with any change that's forced without an appropriate scaling point, this might interrupt your workload.

For more information, see [Timeout action for capacity changes \(p. 154\)](#).

Aurora Serverless v1 and failover

If the DB instance for an Aurora Serverless v1 DB cluster becomes unavailable or the Availability Zone (AZ) it is in fails, Aurora recreates the DB instance in a different AZ. We refer to this capability as automatic multi-AZ failover.

This failover mechanism takes longer than for an Aurora provisioned cluster. The Aurora Serverless v1 failover time is currently undefined because it depends on demand and capacity availability in other AZs within the given AWS Region.

Because Aurora separates computation capacity and storage, the storage volume for the cluster is spread across multiple AZs. Your data remains available even if outages affect the DB instance or the associated AZ.

Aurora Serverless v1 and snapshots

The cluster volume for an Aurora Serverless v1 cluster is always encrypted. You can choose the encryption key, but you can't disable encryption. To copy or share a snapshot of an Aurora Serverless v1 cluster, you encrypt the snapshot using your own AWS KMS key. For more information, see [Copying a DB cluster snapshot](#). To learn more about encryption and Amazon Aurora, see [Encrypting Amazon Aurora resources](#).

Creating an Aurora Serverless v1 DB cluster

When you create an Aurora Serverless v1 DB cluster, you can set the minimum and maximum capacity for the cluster. A capacity unit is equivalent to a specific compute and memory configuration. Aurora Serverless creates scaling rules for thresholds for CPU utilization, connections, and available memory and seamlessly scales to a range of capacity units as needed for your applications. For more information see [Aurora Serverless v1 architecture \(p. 152\)](#).

You can set the following specific values for your Aurora Serverless v1 DB cluster:

- **Minimum Aurora capacity unit** – Aurora Serverless v1 can reduce capacity down to this capacity unit.
- **Maximum Aurora capacity unit** – Aurora Serverless v1 can increase capacity up to this capacity unit.

You can also choose the following optional scaling configuration options:

- **Roll back the capacity change** – To cancel capacity changes if Aurora Serverless v1 can't find a scaling point, choose this setting.

Force the capacity change – To force Aurora Serverless v1 to scale even if it can't find a scaling point before it times out, choose this setting.

For more information, see [Timeout action for capacity changes \(p. 154\)](#).

- **Pause compute capacity after consecutive minutes of inactivity** – You can choose this setting if you want Aurora Serverless v1 to scale to zero when there's no activity on your DB cluster for an amount of time you specify. With this setting enabled, your Aurora Serverless v1 DB cluster automatically resumes processing and scales to the necessary capacity to handle the workload when database traffic resumes. To learn more, see [Pause and resume for Aurora Serverless v1 \(p. 155\)](#).

Before you can create an Aurora Serverless v1 DB cluster, you need an AWS account. You also need to have completed the setup tasks for working with Amazon Aurora. For more information, see [Setting up your environment for Amazon Aurora \(p. 84\)](#). You also need to complete other preliminary steps for creating any Aurora DB cluster. To learn more, see [Creating an Amazon Aurora DB cluster \(p. 125\)](#).

Aurora Serverless v1 is available in certain AWS Regions and for specific Aurora MySQL and Aurora PostgreSQL versions only. For more information, see [Aurora Serverless v1 \(p. 29\)](#).

Note

The cluster volume for an Aurora Serverless v1 cluster is always encrypted. When you create your Aurora Serverless v1 DB cluster, you can't turn off encryption, but you can choose to use your own encryption key.

You can create an Aurora Serverless v1 DB cluster with the AWS Management Console, the AWS CLI, or the RDS API by following the steps below.

Console

To create a new Aurora Serverless v1 DB cluster, you sign in to the AWS Management Console and choose an AWS Region that supports Aurora Serverless v1. Choose Amazon RDS from the AWS Services list, and then choose **Create database**.

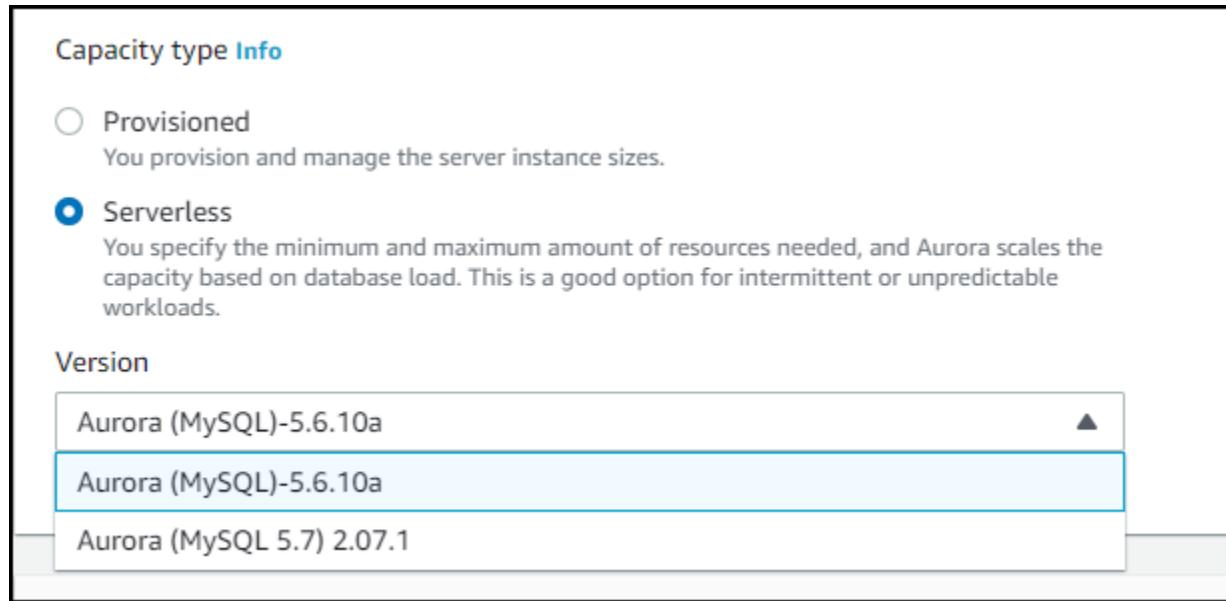
On the **Create database** page:

- Choose **Standard Create** for the database creation method.
- Choose **Amazon Aurora** for the Engine type in the **Engine options** section.

You then choose **Amazon Aurora with MySQL compatibility** or **Amazon Aurora with PostgreSQL compatibility** and continue creating the Aurora Serverless v1 DB cluster by using the steps from the following examples. If you choose a version of the DB engine that doesn't support Aurora Serverless v1, the **Serverless** option doesn't display.

Example for Aurora MySQL

Choose **Amazon Aurora with MySQL Compatibility** for the Edition. Choose the Aurora MySQL engine you want for your cluster from the **Version** selector. The following image shows an example.



Choose **Serverless** for the Capacity type.

You can configure the scaling configuration of the Aurora Serverless v1 DB cluster by adjusting values in the **Capacity settings** section of the page. To learn more about capacity settings, see [Autoscaling for Aurora Serverless v1 \(p. 153\)](#). The following image shows the **Capacity settings** you can adjust for an Aurora MySQL Serverless DB cluster.

Capacity settings

This billing estimate is based on published prices. [Learn more](#)

Minimum Aurora capacity unit Info	Maximum Aurora capacity unit Info
<input type="text" value="1"/> 2GB RAM	<input type="text" value="256"/> 488GB RAM

▼ Additional scaling configuration

Force scaling the capacity to the specified values when the timeout is reached [Info](#)
Enable to force capacity scaling as soon as possible. Disable to cancel the capacity changes when a timeout is reached

Pause compute capacity after consecutive minutes of inactivity [Info](#)
You are only charged for database storage while the compute capacity is paused

<input type="text" value="0"/> hours	<input type="text" value="5"/> minutes	<input type="text" value="0"/> seconds
--------------------------------------	--	--

Max: 24 hours

You can also enable the Data API for your Aurora MySQL Serverless DB cluster. Select the **Data API** checkbox in the **Connectivity** section of the Create database page. To learn more about the Data API, see [Using the Data API for Aurora Serverless \(p. 178\)](#).

Example for Aurora PostgreSQL

Choose **Amazon Aurora with Postgres; Compatibility** for the Edition and select the **Version** of Aurora PostgreSQL available for Aurora Serverless v1. For more information, see [Aurora Serverless v1 \(p. 29\)](#).

Capacity type [Info](#)

Provisioned
You provision and manage the server instance sizes.

Serverless
You specify the minimum and maximum amount of resources needed, and Aurora scales the capacity based on database load. This is a good option for intermittent or unpredictable workloads.

Version

Aura PostgreSQL (compatible with PostgreSQL 10.7) ▼

To see more versions, modify the capacity types. [Info](#)

You can configure the scaling configuration of the Aurora Serverless v1 DB cluster by adjusting values in the **Capacity settings** section of the page. The following image shows the **Capacity settings** you can adjust for an Aurora PostgreSQL Serverless DB cluster. To learn more about capacity settings, see [Autoscaling for Aurora Serverless v1 \(p. 153\)](#).

Capacity settings

This billing estimate is based on published prices. [Learn more](#)

Minimum Aurora capacity unit [Info](#)

2
4GB RAM

Maximum Aurora capacity unit [Info](#)

384
768GB RAM

▼ Additional scaling configuration

Force scaling the capacity to the specified values when the timeout is reached [Info](#)
Enable to force capacity scaling as soon as possible. Disable to cancel the capacity changes when a timeout is reached

Pause compute capacity after consecutive minutes of inactivity [Info](#)
You are only charged for database storage while the compute capacity is paused

0 ▼ hours 5 ▼ minutes 0 ▼ seconds

Max: 24 hours

You can also enable the Data API for your Aurora PostgreSQL Serverless DB cluster. Select the **Data API** checkbox in the **Connectivity** section of the Create database page. See [Using the Data API for Aurora Serverless \(p. 178\)](#) for more information about the Data API.

For more information on creating an Aurora DB cluster using the AWS Management Console, see [Creating an Amazon Aurora DB cluster \(p. 125\)](#).

Note

If you receive the following error message when trying to create your cluster, your account needs additional permissions.

Unable to create the resource. Verify that you have permission to create service linked role. Otherwise wait and try again later.

See [Using service-linked roles for Amazon Aurora \(p. 1783\)](#) for more information.

You can't directly connect to the DB instance on your Aurora Serverless v1 DB cluster. To connect to your Aurora Serverless v1 DB cluster, you use the database endpoint. You can find the endpoint for your Aurora Serverless v1 DB cluster on the **Connectivity & security** tab for your cluster in the AWS Management Console. For more information, see [Connecting to an Amazon Aurora DB cluster \(p. 281\)](#).

AWS CLI

To create a new Aurora Serverless v1 DB cluster with the AWS CLI, run the `create-db-cluster` command and specify `serverless` for the `--engine-mode` option.

You can optionally specify the `--scaling-configuration` option to configure the minimum capacity, maximum capacity, and automatic pause when there are no connections.

The following command examples create a new Serverless DB cluster by setting the `--engine-mode` option to `serverless`. The examples also specify values for the `--scaling-configuration` option.

Example for Aurora MySQL

The following commands create new MySQL-compatible Serverless DB clusters. Valid capacity values for Aurora MySQL are 1, 2, 4, 8, 16, 32, 64, 128, and 256.

For Linux, macOS, or Unix:

```
aws rds create-db-cluster --db-cluster-identifier sample-cluster --engine aurora --engine-version 5.6.10a \
--engine-mode serverless --scaling-configuration
MinCapacity=4,MaxCapacity=32,SecondsUntilAutoPause=1000,AutoPause=true \
--master-username username --master-user-password password

aws rds create-db-cluster --db-cluster-identifier sample-cluster --engine aurora-mysql --engine-version 5.7.mysql_aurora.2.07.1 \
--engine-mode serverless --scaling-configuration
MinCapacity=4,MaxCapacity=32,SecondsUntilAutoPause=1000,AutoPause=true \
--master-username username --master-user-password password
```

For Windows:

```
aws rds create-db-cluster --db-cluster-identifier sample-cluster --engine aurora --engine-version 5.6.10a ^
--engine-mode serverless --scaling-configuration
MinCapacity=4,MaxCapacity=32,SecondsUntilAutoPause=1000,AutoPause=true ^
--master-username username --master-user-password password
```

```
aws rds create-db-cluster --db-cluster-identifier sample-cluster --engine aurora-mysql --  
engine-version 5.7.mysql_aurora.2.07.1 ^  
--engine-mode serverless --scaling-configuration  
MinCapacity=4,MaxCapacity=32,SecondsUntilAutoPause=1000,AutoPause=true ^  
--master-username username --master-user-password password
```

Example for Aurora PostgreSQL

The following command creates a new PostgreSQL 10.12–compatible Serverless DB cluster. Valid capacity values for Aurora PostgreSQL are 2, 4, 8, 16, 32, 64, 192, and 384.

For Linux, macOS, or Unix:

```
aws rds create-db-cluster --db-cluster-identifier sample-cluster --engine aurora-postgresql  
--engine-version 10.12 ^  
--engine-mode serverless --scaling-configuration  
MinCapacity=8,MaxCapacity=64,SecondsUntilAutoPause=1000,AutoPause=true ^  
--master-username username --master-user-password password
```

For Windows:

```
aws rds create-db-cluster --db-cluster-identifier sample-cluster --engine aurora-postgresql  
--engine-version 10.12 ^  
--engine-mode serverless --scaling-configuration  
MinCapacity=8,MaxCapacity=64,SecondsUntilAutoPause=1000,AutoPause=true ^  
--master-username username --master-user-password password
```

RDS API

To create a new Aurora Serverless v1 DB cluster with the RDS API, run the [CreateDBCluster](#) operation and specify `serverless` for the `EngineMode` parameter.

You can optionally specify the `ScalingConfiguration` parameter to configure the minimum capacity, maximum capacity, and automatic pause when there are no connections. Valid capacity values include the following:

- Aurora MySQL: 1, 2, 4, 8, 16, 32, 64, 128, and 256.
- Aurora PostgreSQL: 2, 4, 8, 16, 32, 64, 192, and 384.

Restoring an Aurora Serverless v1 DB cluster

You can configure an Aurora Serverless v1 DB cluster when you restore a provisioned DB cluster snapshot with the AWS Management Console, the AWS CLI, or the RDS API.

When you restore a snapshot to an Aurora Serverless v1 DB cluster, you can set the following specific values:

- **Minimum Aurora capacity unit** – Aurora Serverless v1 can reduce capacity down to this capacity unit.
- **Maximum Aurora capacity unit** – Aurora Serverless v1 can increase capacity up to this capacity unit.
- **Timeout action** – The action to take when a capacity modification times out because it can't find a scaling point. Aurora Serverless v1 DB cluster can force your DB cluster to the new capacity settings if

set the **Force scaling the capacity to the specified values...** option. Or, it can roll back the capacity change to cancel it if you don't choose the option. For more information, see [Timeout action for capacity changes \(p. 154\)](#).

- **Pause after inactivity** – The amount of time with no database traffic to scale to zero processing capacity. When database traffic resumes, Aurora automatically resumes processing capacity and scales to handle the traffic.

For general information about restoring a DB cluster from a snapshot, see [Restoring from a DB cluster snapshot \(p. 497\)](#).

Console

You can restore a DB cluster snapshot to an Aurora DB cluster with the AWS Management Console.

To restore a DB cluster snapshot to an Aurora DB cluster

1. Sign in to the AWS Management Console and open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. In the upper-right corner of the AWS Management Console, choose the AWS Region that hosts your source DB cluster.
3. In the navigation pane, choose **Snapshots**, and choose the DB cluster snapshot that you want to restore.
4. For **Actions**, choose **Restore Snapshot**.
5. On the **Restore DB Cluster** page, choose **Serverless** for **Capacity type**.

The screenshot shows the 'Restore DB Cluster' wizard with the following details:

- Instance specifications**
- DB Engine**: Aurora MySQL
- DB Engine Version**: Aurora (MySQL)-5.6.10a (default)
- Capacity type**:
 - Provisioned**: You provision and manage the server instance sizes.
 - Serverless**: You specify the minimum and maximum of resources for a DB cluster. Aurora scales the capacity based on database load (currently available for Aurora MySQL 5.6).

6. In the **DB cluster identifier** field, type the name for your restored DB cluster, and complete the other fields.
7. In the **Capacity settings** section, modify the scaling configuration.

The screenshot shows the 'Capacity settings' section of the AWS console. It includes fields for 'Minimum Aurora capacity unit' (set to 1, 2GB RAM) and 'Maximum Aurora capacity unit' (set to 256, 488GB RAM). Below these are sections for 'Additional scaling configuration' and 'Pause compute capacity after consecutive minutes of inactivity'. The 'Pause compute capacity after consecutive minutes of inactivity' option is checked, and its settings are shown: 0 hours, 5 minutes, and 0 seconds. A note indicates a maximum of 24 hours.

Capacity settings

This billing estimate is based on published prices. [Learn more](#)

Minimum Aurora capacity unit [Info](#)

1
2GB RAM

Maximum Aurora capacity unit [Info](#)

256
488GB RAM

▼ Additional scaling configuration

Force scaling the capacity to the specified values when the timeout is reached [Info](#)
Enable to force capacity scaling as soon as possible. Disable to cancel the capacity changes when a timeout is reached

Pause compute capacity after consecutive minutes of inactivity [Info](#)
You are only charged for database storage while the compute capacity is paused

0 ▾ hours 5 ▾ minutes 0 ▾ seconds

Max: 24 hours

8. Choose **Restore DB Cluster**.

To connect to an Aurora Serverless v1 DB cluster, use the database endpoint. For details, see the instructions in [Connecting to an Amazon Aurora DB cluster \(p. 281\)](#).

Note

If you encounter the following error message, your account requires additional permissions:
Unable to create the resource. Verify that you have permission to create service linked role. Otherwise wait and try again later.
For more information, see [Using service-linked roles for Amazon Aurora \(p. 1783\)](#).

AWS CLI

You can configure an Aurora Serverless v1 DB cluster when you restore from a snapshot of another DB cluster. You can do so with the AWS CLI by using the `restore-db-cluster-from-snapshot` CLI command. With your command, you include the following required parameters:

- `--db-cluster-identifier mynewdbcluster`
- `--snapshot-identifier mydbclustersnapshot`
- `--engine-mode serverless`

To restore a snapshot to an Aurora Serverless v1 cluster with MySQL 5.7 compatibility, include the following additional parameters:

- `--engine aurora-mysql`

- **--engine-version 5.7**

The `--engine` and `--engine-version` parameters let you create a MySQL 5.7-compatible Aurora Serverless v1 cluster from a MySQL 5.6-compatible Aurora or Aurora Serverless v1 snapshot. The following example restores a snapshot from a MySQL 5.6-compatible cluster named `mydbclustersnapshot` to a MySQL 5.7-compatible Aurora Serverless v1 cluster named `mynewdbcluster`.

For Linux, macOS, or Unix:

```
aws rds restore-db-cluster-from-snapshot \
--db-cluster-identifier mynewdbcluster \
--snapshot-identifier mydbclustersnapshot \
--engine-mode serverless \
--engine aurora-mysql \
--engine-version 5.7
```

For Windows:

```
aws rds restore-db-cluster-from-snapshot ^
--db-instance-identifier mynewdbcluster ^
--db-snapshot-identifier mydbclustersnapshot ^
--engine aurora-mysql ^
--engine-version 5.7
```

You can optionally specify the `--scaling-configuration` option to configure the minimum capacity, maximum capacity, and automatic pause when there are no connections. Valid capacity values include the following:

- Aurora MySQL: 1, 2, 4, 8, 16, 32, 64, 128, and 256.
- Aurora PostgreSQL: 2, 4, 8, 16, 32, 64, 192, and 384.

In the following example, you restore from a previously created DB cluster snapshot named `mydbclustersnapshot` to a new DB cluster named `mynewdbcluster`. You set the `--scaling-configuration` so that the new Aurora Serverless DB cluster can scale from 8 ACUs to 64 ACUs (Aurora capacity units) as needed to process the workload. After processing completes and after 1000 seconds with no connections to support, the cluster shuts down until connection requests prompt it to restart.

For Linux, macOS, or Unix:

```
aws rds restore-db-cluster-from-snapshot \
--db-cluster-identifier mynewdbcluster \
--snapshot-identifier mydbclustersnapshot \
--engine-mode serverless --scaling-configuration
MinCapacity=8,MaxCapacity=64,TimeoutAction='ForceApplyCapacityChange',SecondsUntilAutoPause=1000,AutoP
```

For Windows:

```
aws rds restore-db-cluster-from-snapshot ^
--db-instance-identifier mynewdbcluster ^
--db-snapshot-identifier mydbclustersnapshot ^
--engine-mode serverless --scaling-configuration
MinCapacity=8,MaxCapacity=64,TimeoutAction='ForceApplyCapacityChange',SecondsUntilAutoPause=1000,AutoP
```

RDS API

To configure an Aurora Serverless v1 DB cluster when you restore from a DB cluster using the RDS API, run the [RestoreDBClusterFromSnapshot](#) operation and specify `serverless` for the `EngineMode` parameter.

You can optionally specify the `ScalingConfiguration` parameter to configure the minimum capacity, maximum capacity, and automatic pause when there are no connections. Valid capacity values include the following:

- Aurora MySQL: 1, 2, 4, 8, 16, 32, 64, 128, and 256.
- Aurora PostgreSQL: 2, 4, 8, 16, 32, 64, 192, and 384.

Modifying an Aurora Serverless v1 DB cluster

After you configure an Aurora Serverless v1 DB cluster, you can modify its scaling configuration with the AWS Management Console, the AWS CLI, or the RDS API.

You can set the minimum and maximum capacity for the DB cluster. Each capacity unit is equivalent to a specific compute and memory configuration. Aurora Serverless v1 automatically creates scaling rules for thresholds for CPU utilization, connections, and available memory. You can also set whether Aurora Serverless v1 pauses the database when there's no activity and then resumes when activity begins again.

You can set the following specific values:

- **Minimum Aurora capacity unit** – Aurora Serverless v1 can reduce capacity down to this capacity unit.
- **Maximum Aurora capacity unit** – Aurora Serverless v1 can increase capacity up to this capacity unit.
- **Autoscaling timeout and action** – This section specifies how long Aurora Serverless waits to find a scaling point before timing out. It also specifies the action to take when a capacity modification times out because it can't find a scaling point. Aurora can force the capacity change to set the capacity to the specified value as soon as possible. Or, it can roll back the capacity change to cancel it. For more information, see [Timeout action for capacity changes \(p. 154\)](#).
- **Pause after inactivity** – The amount of time with no database traffic to scale to zero processing capacity. When database traffic resumes, Aurora automatically resumes processing capacity and scales to handle the traffic.

Console

You can modify the scaling configuration of an Aurora DB cluster with the AWS Management Console.

To modify an Aurora Serverless v1 DB cluster

1. Open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. In the navigation pane, choose **Databases**.
3. Choose the Aurora Serverless v1 DB cluster that you want to modify.
4. For **Actions**, choose **Modify cluster**.
5. In the **Capacity settings** section, modify the scaling configuration.

Capacity settings

This billing estimate is based on published prices. [Learn more](#)

Minimum Aurora capacity unit Info	Maximum Aurora capacity unit Info
<input type="text" value="1"/> 2GB RAM	<input type="text" value="256"/> 488GB RAM

▼ Additional scaling configuration

Force scaling the capacity to the specified values when the timeout is reached [Info](#)
Enable to force capacity scaling as soon as possible. Disable to cancel the capacity changes when a timeout is reached

Pause compute capacity after consecutive minutes of inactivity [Info](#)
You are only charged for database storage while the compute capacity is paused

<input type="text" value="0"/> hours	<input type="text" value="5"/> minutes	<input type="text" value="0"/> seconds
Max: 24 hours		

6. Choose **Continue**.
7. Choose **Modify cluster**.

The change is applied immediately.

AWS CLI

To modify the scaling configuration of an Aurora Serverless v1 DB cluster using the AWS CLI, run the [modify-db-cluster](#) AWS CLI command. Specify the `--scaling-configuration` option to configure the minimum capacity, maximum capacity, and automatic pause when there are no connections. Valid capacity values include the following:

- Aurora MySQL: 1, 2, 4, 8, 16, 32, 64, 128, and 256.
- Aurora PostgreSQL: 2, 4, 8, 16, 32, 64, 192, and 384.

In this example, you modify the scaling configuration of an Aurora Serverless v1 DB cluster named `sample-cluster`.

For Linux, macOS, or Unix:

```
aws rds modify-db-cluster --db-cluster-identifier sample-cluster \
--scaling-configuration
MinCapacity=8,MaxCapacity=64,SecondsUntilAutoPause=500,TimeoutAction='ForceApplyCapacityChange',AutoPa
```

For Windows:

```
aws rds modify-db-cluster --db-cluster-identifier sample-cluster ^
--scaling-configuration
MinCapacity=8,MaxCapacity=64,SecondsUntilAutoPause=500,TimeoutAction='ForceApplyCapacityChange',AutoPa
```

RDS API

You can modify the scaling configuration of an Aurora DB cluster with the [ModifyDBCluster API](#) operation. Specify the `ScalingConfiguration` parameter to configure the minimum capacity, maximum capacity, and automatic pause when there are no connections. Valid capacity values include the following:

- Aurora MySQL: 1, 2, 4, 8, 16, 32, 64, 128, and 256.
- Aurora PostgreSQL: 2, 4, 8, 16, 32, 64, 192, and 384.

Scaling Aurora Serverless v1 DB cluster capacity manually

Typically, Aurora Serverless v1 DB clusters scale seamlessly based on the workload. However, capacity might not always scale fast enough to meet sudden extremes, such as an exponential increase in transactions. In such cases you can initiate the scaling operation manually by setting a new capacity value. After you set the capacity explicitly, Aurora Serverless v1 automatically scales the DB cluster. It does so based on the cooldown period for scaling down.

You can explicitly set the capacity of an Aurora Serverless v1 DB cluster to a specific value with the AWS Management Console, the AWS CLI, or the RDS API.

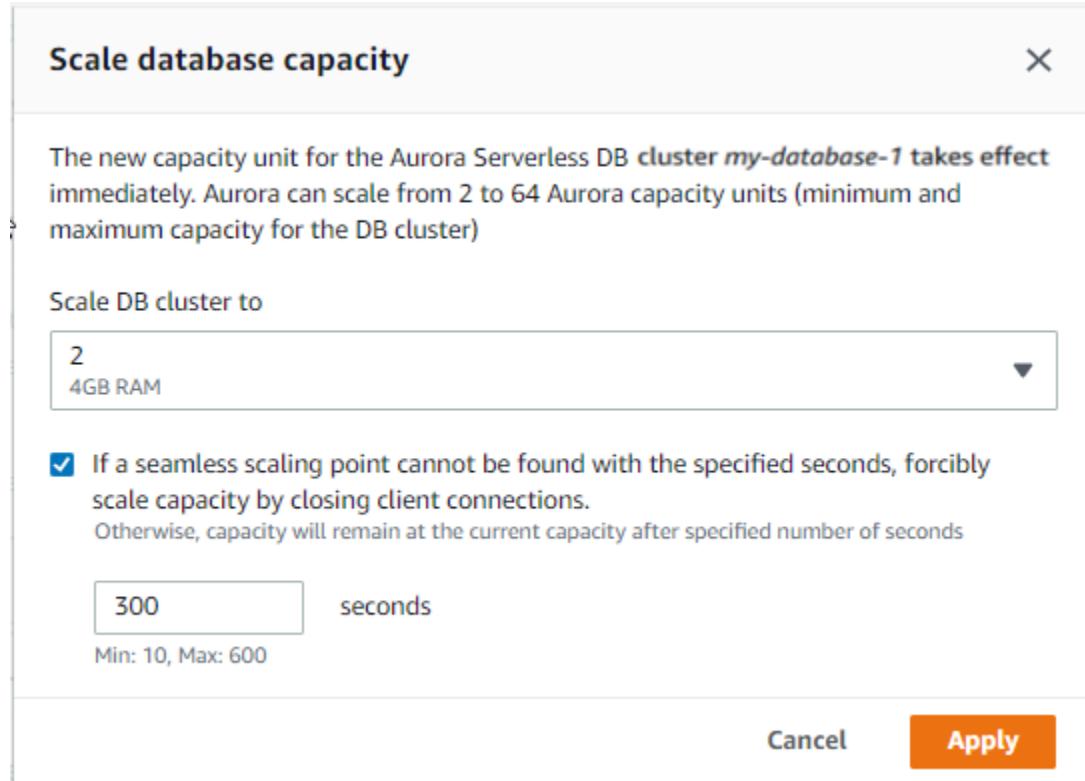
Console

You can set the capacity of an Aurora DB cluster with the AWS Management Console.

To modify an Aurora Serverless v1 DB cluster

1. Open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. In the navigation pane, choose **Databases**.
3. Choose the Aurora Serverless v1 DB cluster that you want to modify.
4. For **Actions**, choose **Set capacity**.
5. In the **Scale database capacity** window, choose the following:
 - a. For the **Scale DB cluster to** drop-down selector, choose the new capacity that you want for your DB cluster.
 - b. For the **If a seamless scaling point cannot be found...** checkbox, choose the behavior you want for your Aurora Serverless v1 DB cluster's `TimeoutAction` setting, as follows:
 - Uncheck this option if you want your capacity to remain unchanged if Aurora Serverless v1 can't find a scaling point before timing out.
 - Check this option if you want to force your Aurora Serverless v1 DB cluster change its capacity even if it can't find a scaling point before timing out. This option can result Aurora Serverless v1 dropping connections that prevent it from finding a scaling point.

- c. In the **seconds** field, enter the amount of time you want to allow your Aurora Serverless v1 DB cluster to look for a scaling point before timing out. You can specify anywhere from 60 seconds to 600 seconds (10 minutes). The default is five minutes (300 seconds). This following example forces the Aurora Serverless v1 DB cluster to scale down to 2 ACUs even if it can't find a scaling point within five minutes.



6. Choose **Apply**.

To learn more about scaling points, `TimeoutAction`, and cooldown periods, see [Autoscaling for Aurora Serverless v1 \(p. 153\)](#).

AWS CLI

To set the capacity of an Aurora Serverless v1 DB cluster using the AWS CLI, run the [modify-current-db-cluster-capacity](#) AWS CLI command, and specify the `--capacity` option. Valid capacity values include the following:

- Aurora MySQL: 1, 2, 4, 8, 16, 32, 64, 128, and 256.
- Aurora PostgreSQL: 2, 4, 8, 16, 32, 64, 192, and 384.

In this example, you set the capacity of an Aurora Serverless v1 DB cluster named `sample-cluster` to 64.

```
aws rds modify-current-db-cluster-capacity --db-cluster-identifier sample-cluster --capacity 64
```

RDS API

You can set the capacity of an Aurora DB cluster with the [ModifyCurrentDBClusterCapacity](#) API operation. Specify the **Capacity** parameter. Valid capacity values include the following:

- Aurora MySQL: 1, 2, 4, 8, 16, 32, 64, 128, and 256.
- Aurora PostgreSQL: 2, 4, 8, 16, 32, 64, 192, and 384.

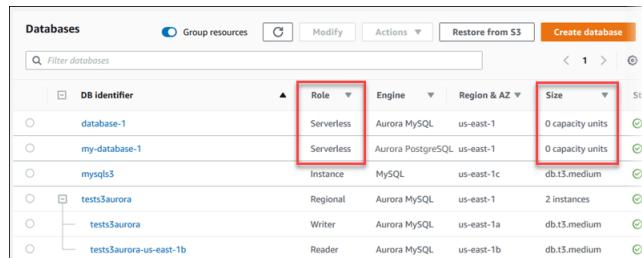
Viewing Aurora Serverless v1 DB clusters

After you create one or more Aurora Serverless v1 DB clusters, you can view which DB clusters are type **Serverless** and which are type **Instance**. You can also view the current number of Aurora capacity units (ACUs) each Aurora Serverless v1 DB cluster is using. Each ACU is a combination of processing (CPU) and memory (RAM) capacity.

To view your Aurora Serverless v1 DB clusters

1. Sign in to the AWS Management Console and open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. In the upper-right corner of the AWS Management Console, choose the AWS Region in which you created the Aurora Serverless v1 DB clusters.
3. In the navigation pane, choose **Databases**.

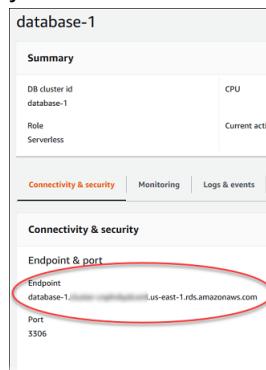
For each DB cluster, the DB cluster type is shown under **Role**. The Aurora Serverless v1 DB clusters show **Serverless** for the type. You can view an Aurora Serverless v1 DB cluster's current capacity under **Size**.



DB identifier	Role	Engine	Region & AZ	Size
database-1	Serverless	Aurora MySQL	us-east-1	0 capacity units
my-database-1	Serverless	Aurora PostgreSQL	us-east-1	0 capacity units
mysqls3	Instance	MySQL	us-east-1c	db.t3.medium
testsaurora	Regional	Aurora MySQL	us-east-1	2 instances
testsaurora	Writer	Aurora MySQL	us-east-1a	db.t3.medium
testsaurora-us-east-1b	Reader	Aurora MySQL	us-east-1b	db.t3.medium

4. Choose the name of an Aurora Serverless v1 DB cluster to display its details.

On the **Connectivity & security** tab, note the database endpoint. Use this endpoint to connect to your Aurora Serverless v1 DB cluster.



Summary	
DB cluster id	CPU
Role	Current activity

Connectivity & security	
Endpoint & port	Net:
Endpoint: database-1.us-east-1.rds.amazonaws.com	VPC: vpc-0
Port: 3306	Subnet: default
	Subnet: subn

Choose the **Configuration** tab to view the capacity settings.

Capacity settings

- Minimum Aurora capacity unit: 2 capacity units
- Maximum Aurora capacity unit: 16 capacity units
- Pause compute capacity after consecutive minutes of inactivity: 5 minutes

Force scaling the capacity to the specified values when the timeout is reached: Enabled

A *scaling event* is generated when the DB cluster scales up, scales down, pauses, or resumes. Choose the **Logs & events** tab to see recent events. The following image shows examples of these events.

Recent events (2)

Mon Aug 06 17:04:15 GMT-700 2018 The DB cluster has scaled from 8 capacity units to 4 capacity units.

Mon Aug 06 17:04:09 GMT-700 2018 Scaling DB cluster from 8 capacity units to 4 capacity units for this.

Monitoring capacity and scaling events for your Aurora Serverless v1 DB cluster

You can view your Aurora Serverless v1 DB cluster in CloudWatch to monitor the capacity allocated to the DB cluster with the `ServerlessDatabaseCapacity` metric. You can also monitor all of the standard Aurora CloudWatch metrics, such as `CPUUtilization`, `DatabaseConnections`, `Queries`, and so on.

You can have Aurora publish some or all database logs to CloudWatch. You select the logs to publish by enabling the [configuration parameters such as `general_log` and `slow_query_log` in the DB cluster parameter group \(p. 156\)](#) associated with the Aurora Serverless v1 cluster. Unlike provisioned clusters, Aurora Serverless v1 clusters don't require you to specify in the DB cluster settings which log types to upload to CloudWatch. Aurora Serverless v1 clusters automatically upload all the available logs. When you disable a log configuration parameter, publishing of the log to CloudWatch stops. You can also delete the logs in CloudWatch if they are no longer needed.

To get started with Amazon CloudWatch for your Aurora Serverless v1 DB cluster, see [Viewing Aurora Serverless v1 logs with Amazon CloudWatch \(p. 159\)](#). To learn more about how to monitor Aurora DB clusters through CloudWatch, see [Monitoring log events in Amazon CloudWatch \(p. 1020\)](#).

To connect to an Aurora Serverless v1 DB cluster, use the database endpoint. For more information, see [Connecting to an Amazon Aurora DB cluster \(p. 281\)](#).

Note

You can't connect directly to specific DB instances in your Aurora Serverless v1 DB clusters.

Deleting an Aurora Serverless v1 DB cluster

When you create an Aurora Serverless v1 DB cluster using the AWS Management Console, the **Enable default protection** option is enabled by default unless you deselect it. That means that you can't immediately delete an Aurora Serverless v1 DB cluster that has **Deletion protection** enabled. To delete Aurora Serverless v1 DB clusters that have deletion protection by using the AWS Management Console,

you first modify the cluster to remove this protection. For information about using the AWS CLI for this task, see [AWS CLI \(p. 177\)](#).

To disable deletion protection using the AWS Management Console

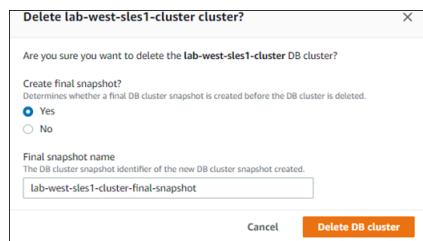
1. Sign in to the AWS Management Console and open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. In the navigation pane, choose **DB clusters**.
3. Choose your Aurora Serverless v1 DB cluster from the list.
4. Choose **Modify** to open your DB cluster's configuration. The Modify DB cluster page opens the Settings, Capacity settings, and other configuration details for your Aurora Serverless v1 DB cluster. Deletion protection is in the **Additional configuration** section.
5. Uncheck the **Enable deletion protection** option in the **Additional configuration** properties card.
6. Choose **Continue**. The **Summary of modifications** appears.
7. Choose **Modify cluster** to accept the summary of modifications. You can also choose **Back** to modify your changes or **Cancel** to discard your changes.

After deletion protection is no longer active, you can delete your Aurora Serverless v1 DB cluster by using the AWS Management Console.

Console

To delete an Aurora Serverless v1 DB cluster

1. Sign in to the AWS Management Console and open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. In the **Resources** section, choose **DB Clusters**.
3. Choose the Aurora Serverless v1 DB cluster that you want to delete.
4. For **Actions**, choose **Delete**. You're prompted to confirm that you want to delete your Aurora Serverless v1 DB cluster.
5. We recommend that you keep the preselected options:
 - **Yes for Create final snapshot?**
 - Your Aurora Serverless v1 DB cluster name plus `-final-snapshot` for **Final snapshot name**. However, you can change the name for your final snapshot in this field.



If you choose **No** for **Create final snapshot?** you can't restore your DB cluster using snapshots or point-in-time recovery.

6. Choose **Delete DB cluster**.

Aurora Serverless v1 deletes your DB cluster. If you chose to have a final snapshot, you see your Aurora Serverless v1 DB cluster's status change to "Backing-up" before it's deleted and no longer appears in the list.

AWS CLI

Before you begin, configure your AWS CLI with your AWS Access Key ID, AWS Secret Access Key, and the AWS Region where your Aurora Serverless v1 DB cluster resides. For more information, see [Configuration basics](#) in the AWS Command Line Interface User Guide.

You can't delete an Aurora Serverless v1 DB cluster until after you first disable deletion protection for clusters configured with this option. If you try to delete a cluster that has this protection option enabled, you see the following error message.

```
An error occurred (InvalidParameterCombination) when calling the DeleteDBCluster
operation: Cannot delete protected Cluster, please disable deletion protection and try
again.
```

You can change your Aurora Serverless v1 DB cluster's deletion-protection setting by using the [modify-db-cluster](#) AWS CLI command as shown in the following:

```
aws rds modify-db-cluster --db-cluster-identifier your-cluster-name --no-deletion-
protection
```

This command returns the revised properties for the specified DB cluster. You can now delete your Aurora Serverless v1 DB cluster.

We recommend that you always create a final snapshot whenever you delete an Aurora Serverless v1 DB cluster. The following example of using the AWS CLI [delete-db-cluster](#) shows you how. You provide the name of your DB cluster and a name for the snapshot.

For Linux, macOS, or Unix:

```
aws rds delete-db-cluster --db-cluster-identifier \
your-cluster-name --no-skip-final-snapshot \
--final-db-snapshot-identifier name-your-snapshot
```

For Windows:

```
aws rds delete-db-cluster --db-cluster-identifier ^
your-cluster-name --no-skip-final-snapshot ^
--final-db-snapshot-identifier name-your-snapshot
```

Aurora Serverless v1 and Aurora database engine versions

Aurora Serverless v1 is available in certain AWS Regions and for specific Aurora MySQL and Aurora PostgreSQL versions only. For the current list of AWS Regions that support Aurora Serverless v1 and the specific Aurora MySQL and Aurora PostgreSQL versions available in each Region, see [Aurora Serverless v1 \(p. 29\)](#).

Aurora Serverless v1 uses its associated Aurora database engine to identify specific supported releases for each database engine supported, as follows:

- Aurora MySQL Serverless
- Aurora PostgreSQL Serverless

When minor releases of the database engines become available for Aurora Serverless v1, they are applied automatically in the various AWS Regions where Aurora Serverless v1 is available. In other words,

you don't need to upgrade your Aurora Serverless v1 DB cluster to get a new minor release for your cluster's DB engine when it's available for Aurora Serverless v1.

Aurora MySQL Serverless

If you want to use Aurora MySQL-Compatible Edition for your Aurora Serverless v1 DB cluster, you can choose between Aurora MySQL 5.6-compatible or Aurora MySQL 5.7-compatible versions. These two editions of Aurora MySQL differ significantly. We recommend that you compare their differences before creating your Aurora Serverless v1 DB cluster so that you make the right choice for your use case. To learn about enhancements and bug fixes for Aurora MySQL Serverless 5.6 and 5.7, see [Database engine updates for Aurora MySQL Serverless clusters \(p. 1247\)](#).

Aurora PostgreSQL Serverless

If you want to use Aurora PostgreSQL for your Aurora Serverless v1 DB cluster, you have a single version available to use. Minor releases for Aurora PostgreSQL-Compatible Edition include only changes that are backward-compatible. Your Aurora Serverless v1 DB cluster is transparently upgraded when an Aurora PostgreSQL minor release becomes available for Aurora Serverless v1 in your Region.

For example, the minor version Aurora PostgreSQL 10.14 release was transparently applied to all Aurora Serverless v1 DB clusters running the prior Aurora PostgreSQL version. For more information about the Aurora PostgreSQL version 10.14 update, see [PostgreSQL 10.14, Aurora PostgreSQL release 2.7 \(p. 1631\)](#).

Using the Data API for Aurora Serverless

By using the Data API for Aurora Serverless, you can work with a web-services interface to your Aurora Serverless DB cluster. The Data API doesn't require a persistent connection to the DB cluster. Instead, it provides a secure HTTP endpoint and integration with AWS SDKs. You can use the endpoint to run SQL statements without managing connections.

All calls to the Data API are synchronous. By default, a call times out if it's not finished processing within 45 seconds. However, you can continue running a SQL statement if the call times out by using the `continueAfterTimeout` parameter. For an example, see [Running a SQL transaction \(p. 198\)](#).

Users don't need to pass credentials with calls to the Data API, because the Data API uses database credentials stored in AWS Secrets Manager. To store credentials in Secrets Manager, users must be granted the appropriate permissions to use Secrets Manager, and also the Data API. For more information about authorizing users, see [Authorizing access to the Data API \(p. 179\)](#).

You can also use Data API to integrate Aurora Serverless with other AWS applications such as AWS Lambda, AWS AppSync, and AWS Cloud9. The API provides a more secure way to use AWS Lambda. It enables you to access your DB cluster without your needing to configure a Lambda function to access resources in a virtual private cloud (VPC). For more information, see [AWS Lambda, AWS AppSync, and AWS Cloud9](#).

You can enable the Data API when you create the Aurora Serverless cluster. You can also modify the configuration later. For more information, see [Enabling the Data API \(p. 182\)](#).

After you enable the Data API, you can also use the query editor for Aurora Serverless. For more information, see [Using the query editor for Aurora Serverless \(p. 204\)](#).

Topics

- [Data API availability \(p. 179\)](#)
- [Authorizing access to the Data API \(p. 179\)](#)
- [Enabling the Data API \(p. 182\)](#)
- [Creating an Amazon VPC endpoint for the Data API \(AWS PrivateLink\) \(p. 183\)](#)
- [Calling the Data API \(p. 186\)](#)

- [Using the Java client library for Data API \(p. 199\)](#)
- [Troubleshooting Data API issues \(p. 201\)](#)

Data API availability

The Data API can be enabled for Aurora Serverless DB clusters using specific Aurora MySQL and Aurora PostgreSQL versions only. For more information, see [Data API for Aurora Serverless \(p. 31\)](#).

The following table shows the AWS Regions where the Data API is currently available for Aurora Serverless. To access the Data API in these Regions, use the HTTPS protocol.

Region	Link
US East (Ohio)	rds-data.us-east-2.amazonaws.com
US East (N. Virginia)	rds-data.us-east-1.amazonaws.com
US West (N. California)	rds-data.us-west-1.amazonaws.com
US West (Oregon)	rds-data.us-west-2.amazonaws.com
Asia Pacific (Mumbai)	rds-data.ap-south-1.amazonaws.com
Asia Pacific (Seoul)	rds-data.ap-northeast-2.amazonaws.com
Asia Pacific (Singapore)	rds-data.ap-southeast-1.amazonaws.com
Asia Pacific (Sydney)	rds-data.ap-southeast-2.amazonaws.com
Asia Pacific (Tokyo)	rds-data.ap-northeast-1.amazonaws.com
Canada (Central)	rds-data.ca-central-1.amazonaws.com
Europe (Frankfurt)	rds-data.eu-central-1.amazonaws.com
Europe (Ireland)	rds-data.eu-west-1.amazonaws.com
Europe (London)	rds-data.eu-west-2.amazonaws.com
Europe (Paris)	rds-data.eu-west-3.amazonaws.com

If you require cryptographic modules validated by FIPS 140-2 when accessing the Data API through a command line interface or an API, use a FIPS endpoint. For more information about the available FIPS endpoints, see [Federal Information Processing Standard \(FIPS\) 140-2](#).

Authorizing access to the Data API

Users can invoke Data API operations only if they are authorized to do so. You can give a user permission to use the Data API by attaching an AWS Identity and Access Management (IAM) policy that defines their privileges. You can also attach the policy to a role if you're using IAM roles. An AWS managed policy, `AmazonRDSDataFullAccess`, includes permissions for the RDS Data API.

The `AmazonRDSDataFullAccess` policy also includes permissions for the user to get the value of a secret from AWS Secrets Manager. Users need to use Secrets Manager to store secrets that they can use in their calls to the Data API. Using secrets means that users don't need to include database credentials for the resources that they target in their calls to the Data API. The Data API transparently calls Secrets Manager, which allows (or denies) the user's request for the secret. For information about setting up secrets to use with the Data API, see [Storing database credentials in AWS Secrets Manager \(p. 181\)](#).

The `AmazonRDSDataFullAccess` policy provides complete access (through the Data API) to resources. You can narrow the scope by defining your own policies that specify the Amazon Resource Name (ARN) of a resource.

For example, the following policy shows an example of the minimum required permissions for a user to access the Data API for the DB cluster identified by its ARN. The policy includes the needed permissions to access Secrets Manager and get authorization to the DB instance for the user.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "SecretsManagerDbCredentialsAccess",  
            "Effect": "Allow",  
            "Action": [  
                "secretsmanager:GetSecretValue"  
            ],  
            "Resource": "arn:aws:secretsmanager:*::secret:rds-db-credentials/*"  
        },  
        {  
            "Sid": "RDSDataserviceAccess",  
            "Effect": "Allow",  
            "Action": [  
                "rds-data:BatchExecuteStatement",  
                "rds-data:BeginTransaction",  
                "rds-data:CommitTransaction",  
                "rds-data:ExecuteStatement",  
                "rds-data:RollbackTransaction"  
            ],  
            "Resource": "arn:aws:rds:us-east-2:111122223333:cluster:prod"  
        }  
    ]  
}
```

We recommend that you use a specific ARN for the "Resources" element in your policy statements (as shown in the example) rather than a wildcard (*).

Working with tag-based authorization

The Data API and Secrets Manager both support tag-based authorization. *Tags* are key-value pairs that label a resource, such as an RDS cluster, with an additional string value, for example:

- `environment:production`
- `environment:development`

You can apply tags to your resources for cost allocation, operations support, access control, and many other reasons. (If you don't already have tags on your resources and you want to apply them, you can learn more at [Tagging Amazon RDS resources](#).) You can use the tags in your policy statements to limit access to the RDS clusters that are labeled with these tags. As an example, an Aurora DB cluster might have tags that identify its environment as either production or development.

The following example shows how you can use tags in your policy statements. This statement requires that both the cluster and the secret passed in the Data API request have an `environment:production` tag.

Here's how the policy gets applied: When a user makes a call using the Data API, the request is sent to the service. The Data API first verifies that the cluster ARN passed in the request is tagged with `environment:production`. It then calls Secrets Manager to retrieve the value of the user's secret in the request. Secrets Manager also verifies that the user's secret is tagged with

`environment:production`. If so, Data API then uses the retrieved value for the user's DB password. Finally, if that's also correct, the Data API request is invoked successfully for the user.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "SecretsManagerDbCredentialsAccess",
            "Effect": "Allow",
            "Action": [
                "secretsmanager:GetSecretValue"
            ],
            "Resource": "arn:aws:secretsmanager:*::secret:rds-db-credentials/*",
            "Condition": {
                "StringEquals": {
                    "aws:ResourceTag/environment": [
                        "production"
                    ]
                }
            }
        },
        {
            "Sid": "RDSDataServiceAccess",
            "Effect": "Allow",
            "Action": [
                "rds-data:*
```

The example shows separate actions for `rds-data` and `secretsmanager` for the Data API and Secrets Manager. However, you can combine actions and define tag conditions in many different ways to support your specific use cases. For more information, see [Using identity-based policies \(IAM policies\) for Secrets Manager](#).

In the "Condition" element of the policy, you can choose tag keys from among the following:

- `aws:TagKeys`
- `aws:ResourceTag/${TagKey}`

To learn more about resource tags and how to use `aws:TagKeys`, see [Controlling access to AWS resources using resource tags](#).

Note

Both the Data API and AWS Secrets Manager authorize users. If you don't have permissions for all actions defined in a policy, you get an `AccessDeniedException` error.

Storing database credentials in AWS Secrets Manager

When you call the Data API, you can pass credentials for the Aurora Serverless DB cluster by using a secret in Secrets Manager. To pass credentials in this way, you specify the name of the secret or the Amazon Resource Name (ARN) of the secret.

To store DB cluster credentials in a secret

1. Use Secrets Manager to create a secret that contains credentials for the Aurora DB cluster.
For instructions, see [Creating a Basic Secret](#) in the *AWS Secrets Manager User Guide*.
2. Use the Secrets Manager console to view the details for the secret you created, or run the `aws secretsmanager describe-secret` AWS CLI command.

Note the name and ARN of the secret. You can use them in calls to the Data API.

For more information about using Secrets Manager, see the [AWS Secrets Manager User Guide](#).

To understand how Amazon Aurora manages identity and access management, see [How Amazon Aurora works with IAM](#).

For more information about creating an IAM policy, see [Creating IAM Policies](#) in the *IAM User Guide*. For information about adding an IAM policy to a user, see [Adding and Removing IAM Identity Permissions](#) in the *IAM User Guide*.

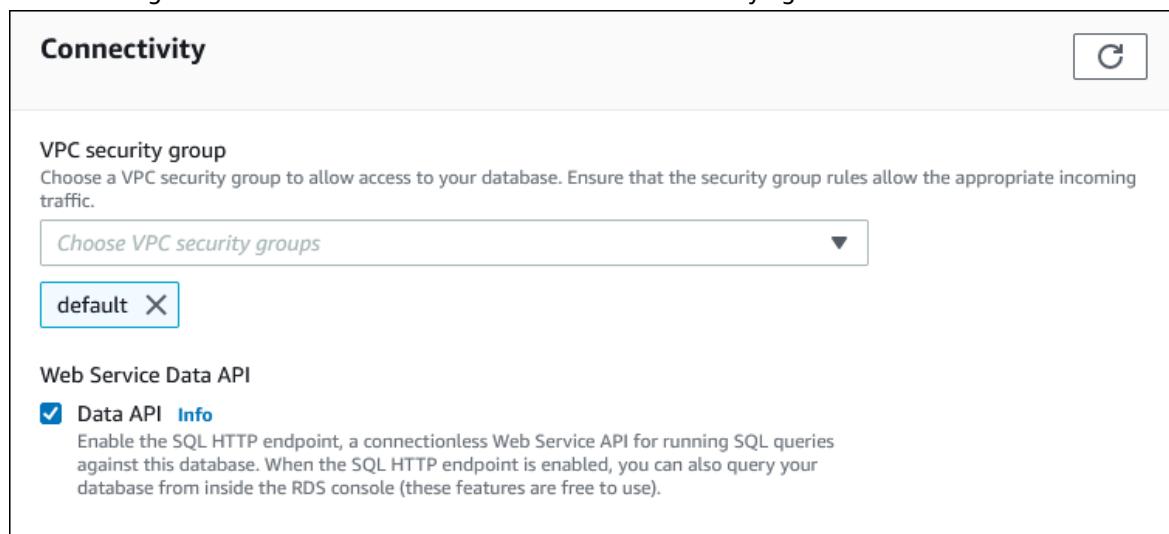
Enabling the Data API

To use the Data API, enable it for your Aurora Serverless DB cluster. You can enable the Data API when you create or modify the DB cluster.

Console

You can enable the Data API by using the RDS console when you create or modify an Aurora Serverless DB cluster. When you create or modify an Aurora Serverless DB cluster, you do so by enabling the Data API in the RDS console's **Connectivity** section.

The following screenshot shows the enabled **Data API** when modifying an Aurora Serverless DB cluster.



For instructions, see [Creating an Aurora Serverless v1 DB cluster \(p. 161\)](#) and [Modifying an Aurora Serverless v1 DB cluster \(p. 170\)](#).

AWS CLI

When you create or modify an Aurora Serverless DB cluster using AWS CLI commands, the Data API is enabled when you specify `--enable-http-endpoint`.

You can specify the `--enable-http-endpoint` using the following AWS CLI commands:

- [create-db-cluster](#)
- [modify-db-cluster](#)

The following example modifies `sample-cluster` to enable the Data API.

For Linux, macOS, or Unix:

```
aws rds modify-db-cluster \
  --db-cluster-identifier sample-cluster \
  --enable-http-endpoint
```

For Windows:

```
aws rds modify-db-cluster ^
  --db-cluster-identifier sample-cluster ^
  --enable-http-endpoint
```

RDS API

When you create or modify an Aurora Serverless DB cluster using Amazon RDS API operations, you enable the Data API by setting the `EnableHttpEndpoint` value to `true`.

You can set the `EnableHttpEndpoint` value using the following API operations:

- [CreateDBCluster](#)
- [ModifyDBCluster](#)

Creating an Amazon VPC endpoint for the Data API (AWS PrivateLink)

Amazon VPC enables you to launch AWS resources, such as Aurora DB clusters and applications, into a virtual private cloud (VPC). AWS PrivateLink provides private connectivity between VPCs and AWS services with high security on the Amazon network. Using AWS PrivateLink, you can create Amazon VPC endpoints, which enable you to connect to services across different accounts and VPCs based on Amazon VPC. For more information about AWS PrivateLink, see [VPC Endpoint Services \(AWS PrivateLink\)](#) in the [Amazon Virtual Private Cloud User Guide](#).

You can call the Data API with Amazon VPC endpoints. Using an Amazon VPC endpoint keeps traffic between applications in your Amazon VPC and the Data API in the AWS network, without using public IP addresses. Amazon VPC endpoints can help you meet compliance and regulatory requirements related to limiting public internet connectivity. For example, if you use an Amazon VPC endpoint, you can keep traffic between an application running on an Amazon EC2 instance and the Data API in the VPCs that contain them.

After you create the Amazon VPC endpoint, you can start using it without making any code or configuration changes in your application.

To create an Amazon VPC endpoint for the Data API

1. Sign in to the AWS Management Console and open the Amazon VPC console at <https://console.aws.amazon.com/vpc/>.

2. Choose **Endpoints**, and then choose **Create Endpoint**.
3. On the **Create Endpoint** page, for **Service category**, choose **AWS services**. For **Service Name**, choose **rds-data**.

The screenshot shows the 'Create Endpoint' wizard. In the 'Service category' section, 'AWS services' is selected. In the 'Service Name' field, 'com.amazonaws.eu-west-1.rds-data' is entered. Below this, a search results table lists various AWS services, with 'com.amazonaws.eu-west-1.rds-data' highlighted by a red box.

Service Name	Owner	Type
com.amazonaws.eu-west-1.elastic-infer...	amazon	Interface
com.amazonaws.eu-west-1.elasticfilesystem	amazon	Interface
com.amazonaws.eu-west-1.elasticfilesystem...	amazon	Interface
com.amazonaws.eu-west-1.elasticloadbal...	amazon	Interface
com.amazonaws.eu-west-1.elasticmapred...	amazon	Interface
com.amazonaws.eu-west-1.events	amazon	Interface
com.amazonaws.eu-west-1.execute-api	amazon	Interface
com.amazonaws.eu-west-1.glue	amazon	Interface
com.amazonaws.eu-west-1.kinesis-firehose	amazon	Interface
com.amazonaws.eu-west-1.kinesis-streams	amazon	Interface
com.amazonaws.eu-west-1.logs	amazon	Interface
com.amazonaws.eu-west-1.rds-data	amazon	Interface
com.amazonaws.eu-west-1.rekognition	amazon	Interface
com.amazonaws.eu-west-1.s3	amazon	Gateway

4. For **VPC**, choose the VPC to create the endpoint in.

Choose the VPC that contains the application that makes Data API calls.

5. For **Subnets**, choose the subnet for each Availability Zone (AZ) used by the AWS service that is running your application.

The screenshot shows the 'Subnets' configuration section. A VPC is selected ('vpc-c20d2da4'). Three subnets are listed: 'subnet-2c8ee364' (eu-west-1a), 'subnet-cefa2594' (eu-west-1b), and 'subnet-20c45946' (eu-west-1c). Each subnet is associated with a specific Availability Zone.

Availability Zone	Subnet ID
eu-west-1a (euw1-az2)	subnet-2c8ee364
eu-west-1b (euw1-az3)	subnet-cefa2594
eu-west-1c (euw1-az1)	subnet-20c45946

To create an Amazon VPC endpoint, specify the private IP address range in which the endpoint will be accessible. To do this, choose the subnet for each Availability Zone. Doing so restricts the VPC endpoint to the private IP address range specific to each Availability Zone and also creates an Amazon VPC endpoint in each Availability Zone.

6. For **Enable DNS name**, select **Enable for this endpoint**.

The screenshot shows the 'Enable DNS name' configuration section. A checkbox is checked for 'Enable for this endpoint'. A note below states: 'To use private DNS names, ensure that the attributes 'Enable DNS hostnames' and 'Enable DNS Support' are set to 'true' for your VPC (vpc-c20d2da4). Learn more.'

Private DNS resolves the standard Data API DNS hostname (<https://rds-data.region.amazonaws.com>) to the private IP addresses associated with the DNS hostname

specific to your Amazon VPC endpoint. As a result, you can access the Data API VPC endpoint using the AWS CLI or AWS SDKs without making any code or configuration changes to update the Data API endpoint URL.

7. For **Security group**, choose a security group to associate with the Amazon VPC endpoint.

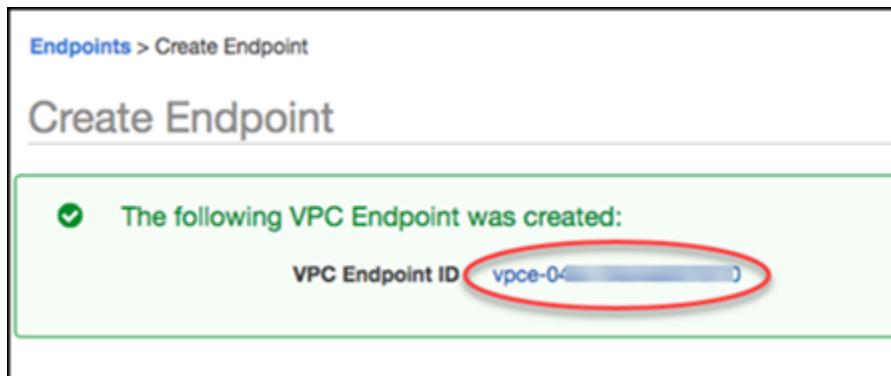
Choose the security group that allows access to the AWS service that is running your application. For example, if an Amazon EC2 instance is running your application, choose the security group that allows access to the Amazon EC2 instance. The security group enables you to control the traffic to the Amazon VPC endpoint from resources in your VPC.

8. For **Policy**, choose **Full Access** to allow anyone inside the Amazon VPC to access the Data API through this endpoint. Or choose **Custom** to specify a policy that limits access.

If you choose **Custom**, enter the policy in the policy creation tool.

9. Choose **Create endpoint**.

After the endpoint is created, choose the link in the AWS Management Console to view the endpoint details.



The endpoint **Details** tab shows the DNS hostnames that were generated while creating the Amazon VPC endpoint.

Endpoint: vpce-04ec15ecbab57bf10							
		Details	Subnets	Security Groups	Policy	Notifications	Tags
Endpoint ID	vpce-04ec15ecbab57bf10	VPC ID	vpce-04ec15ecbab57bf10				
Status	available	Status message					
Creation time	January 31, 2020 at 7:02:32 PM UTC-8	Service name	com.amazonaws.eu-west-1.rds-data				
Endpoint type	Interface	DNS names	vpce-04ec15ecbab57bf10.vpc.amazonaws.com	(redacted)	(redacted)	(redacted)	(redacted)
			vpce-04ec15ecbab57bf10.eu-west-1b.rds-data.eu-west-1.vpc.amazonaws.com	(redacted)	(redacted)	(redacted)	(redacted)
			vpce-04ec15ecbab57bf10.eu-west-1c.rds-data.eu-west-1.vpc.amazonaws.com	(redacted)	(redacted)	(redacted)	(redacted)
			vpce-04ec15ecbab57bf10.eu-west-1a.rds-data.eu-west-1.vpc.amazonaws.com	(redacted)	(redacted)	(redacted)	(redacted)
			rds-data.eu-west-1.amazonaws.com	(redacted)	(redacted)	(redacted)	(redacted)
Private DNS names enabled	true	Private DNS name	rds-data.eu-west-1.amazonaws.com				

You can use the standard endpoint (`rds-data.region.amazonaws.com`) or one of the VPC-specific endpoints to call the Data API within the Amazon VPC. The standard Data API endpoint automatically routes to the Amazon VPC endpoint. This routing occurs because the Private DNS hostname was enabled when the Amazon VPC endpoint was created.

When you use an Amazon VPC endpoint in a Data API call, all traffic between your application and the Data API remains in the Amazon VPCs that contain them. You can use an Amazon VPC endpoint for any type of Data API call. For information about calling the Data API, see [Calling the Data API \(p. 186\)](#).

Calling the Data API

With the Data API enabled on your Aurora Serverless DB cluster, you can run SQL statements on the Aurora DB cluster by using the Data API or the AWS CLI. The Data API supports the programming languages supported by the AWS SDKs. For more information, see [Tools to build on AWS](#).

Note

Currently, the Data API doesn't support arrays of Universal Unique Identifiers (UUIDs).

The Data API provides the following operations to perform SQL statements.

Data API operation	AWS CLI command	Description
<code>ExecuteStatement</code>	<code>rds-data execute-statement</code>	Runs a SQL statement on a database.
<code>BatchExecuteStatement</code>	<code>rds-data batch-execute-statement</code>	Runs a batch SQL statement over an array of data for bulk update and insert operations. You can run a data manipulation language (DML) statement with an array of parameter sets. A batch SQL statement can provide a significant performance improvement over individual insert and update statements.

You can use either operation to run individual SQL statements or to run transactions. For transactions, the Data API provides the following operations.

Data API operation	AWS CLI command	Description
<code>BeginTransaction</code>	<code>rds-data begin-transaction</code>	Starts a SQL transaction.
<code>CommitTransaction</code>	<code>rds-data commit-transaction</code>	Ends a SQL transaction and commits the changes.
<code>RollbackTransaction</code>	<code>rds-data rollback-transaction</code>	Performs a rollback of a transaction.

The operations for performing SQL statements and supporting transactions have the following common Data API parameters and AWS CLI options. Some operations support other parameters or options.

Data API operation parameter	AWS CLI command option	Required	Description
<code>resourceArn</code>	<code>--resource-arn</code>	Yes	The Amazon Resource Name (ARN) of the Aurora Serverless DB cluster.
<code>secretArn</code>	<code>--secret-arn</code>	Yes	The name or ARN of the secret that enables access to the DB cluster.

You can use parameters in Data API calls to `ExecuteStatement` and `BatchExecuteStatement`, and when you run the AWS CLI commands `execute-statement` and `batch-execute-statement`. To use

a parameter, you specify a name-value pair in the `SqlParameter` data type. You specify the value with the `Field` data type. The following table maps Java Database Connectivity (JDBC) data types to the data types that you specify in Data API calls.

JDBC data type	Data API data type
INTEGER, TINYINT, SMALLINT, BIGINT	LONG (or STRING)
FLOAT, REAL, DOUBLE	DOUBLE
DECIMAL	STRING
BOOLEAN, BIT	BOOLEAN
BLOB, BINARY, LONGVARBINARY, VARBINARY	BLOB
CLOB	STRING
Other types (including types related to date and time)	STRING

Note

You can specify the `LONG` or `STRING` data type in your Data API call for `LONG` values returned by the database. We recommend that you do so to avoid losing precision for extremely large numbers, which can happen when you work with JavaScript.

Certain types, such as `DECIMAL` and `TIME`, require a hint so that the Data API passes `String` values to the database as the correct type. To use a hint, include values for `typeHint` in the `SqlParameter` data type. The possible values for `typeHint` are the following:

- `DATE` – The corresponding `String` parameter value is sent as an object of `DATE` type to the database. The accepted format is `YYYY-MM-DD`.
- `DECIMAL` – The corresponding `String` parameter value is sent as an object of `DECIMAL` type to the database.
- `JSON` – The corresponding `String` parameter value is sent as an object of `JSON` type to the database.
- `TIME` – The corresponding `String` parameter value is sent as an object of `TIME` type to the database. The accepted format is `HH:MM:SS[.FFF]`.
- `TIMESTAMP` – The corresponding `String` parameter value is sent as an object of `TIMESTAMP` type to the database. The accepted format is `YYYY-MM-DD HH:MM:SS[.FFF]`.
- `UUID` – The corresponding `String` parameter value is sent as an object of `UUID` type to the database.

Note

For Amazon Aurora PostgreSQL, the Data API always returns the Aurora PostgreSQL data type `TIMESTAMPTZ` in UTC time zone.

Calling the Data API with the AWS CLI

You can call the Data API using the AWS CLI.

The following examples use the AWS CLI for the Data API. For more information, see [AWS CLI reference for the Data API](#).

In each example, replace the Amazon Resource Name (ARN) for the DB cluster with the ARN for your Aurora Serverless DB cluster. Also, replace the secret ARN with the ARN of the secret in Secrets Manager that allows access to the DB cluster.

Note

The AWS CLI can format responses in JSON.

Topics

- [Starting a SQL transaction \(p. 188\)](#)
- [Running a SQL statement \(p. 188\)](#)
- [Running a batch SQL statement over an array of data \(p. 191\)](#)
- [Committing a SQL transaction \(p. 193\)](#)
- [Rolling back a SQL transaction \(p. 193\)](#)

Starting a SQL transaction

You can start a SQL transaction using the `aws rds-data begin-transaction` CLI command. The call returns a transaction identifier.

Important

A transaction times out if there are no calls that use its transaction ID in three minutes. If a transaction times out before it's committed, it's rolled back automatically.

Data definition language (DDL) statements inside a transaction cause an implicit commit. We recommend that you run each DDL statement in a separate `execute-statement` command with the `--continue-after-timeout` option.

In addition to the common options, specify the `--database` option, which provides the name of the database.

For example, the following CLI command starts a SQL transaction.

For Linux, macOS, or Unix:

```
aws rds-data begin-transaction --resource-arn "arn:aws:rds:us-east-1:123456789012:cluster:mydbcluster" \
--database "mydb" --secret-arn "arn:aws:secretsmanager:us-east-1:123456789012:secret:mysecret"
```

For Windows:

```
aws rds-data begin-transaction --resource-arn "arn:aws:rds:us-east-1:123456789012:cluster:mydbcluster" ^
--database "mydb" --secret-arn "arn:aws:secretsmanager:us-east-1:123456789012:secret:mysecret"
```

The following is an example of the response.

```
{  
    "transactionId": "ABC1234567890xyz"  
}
```

Running a SQL statement

You can run a SQL statement using the `aws rds-data execute-statement` CLI command.

You can run the SQL statement in a transaction by specifying the transaction identifier with the `--transaction-id` option. You can start a transaction using the `aws rds-data begin-transaction` CLI command. You can end and commit a transaction using the `aws rds-data commit-transaction` CLI command.

Important

If you don't specify the `--transaction-id` option, changes that result from the call are committed automatically.

In addition to the common options, specify the following options:

- `--sql` (required) – A SQL statement to run on the DB cluster.
- `--transaction-id` (optional) – The identifier of a transaction that was started using the `begin-transaction` CLI command. Specify the transaction ID of the transaction that you want to include the SQL statement in.
- `--parameters` (optional) – The parameters for the SQL statement.
- `--include-result-metadata` | `--no-include-result-metadata` (optional) – A value that indicates whether to include metadata in the results. The default is `--no-include-result-metadata`.
- `--database` (optional) – The name of the database.
- `--continue-after-timeout` | `--no-continue-after-timeout` (optional) – A value that indicates whether to continue running the statement after the call times out. The default is `--no-continue-after-timeout`.

For data definition language (DDL) statements, we recommend continuing to run the statement after the call times out to avoid errors and the possibility of corrupted data structures.

The DB cluster returns a response for the call.

Note

The response size limit is 1 MiB. If the call returns more than 1 MiB of response data, the call is terminated.

The maximum number of requests per second is 1,000.

For example, the following CLI command runs a single SQL statement and omits the metadata in the results (the default).

For Linux, macOS, or Unix:

```
aws rds-data execute-statement --resource-arn "arn:aws:rds:us-east-1:123456789012:cluster:mydbcluster" \
--database "mydb" --secret-arn "arn:aws:secretsmanager:us-east-1:123456789012:secret:mysecret" \
--sql "select * from mytable"
```

For Windows:

```
aws rds-data execute-statement --resource-arn "arn:aws:rds:us-east-1:123456789012:cluster:mydbcluster" ^
--database "mydb" --secret-arn "arn:aws:secretsmanager:us-east-1:123456789012:secret:mysecret" ^
--sql "select * from mytable"
```

The following is an example of the response.

```
{
    "numberOfRecordsUpdated": 0,
    "records": [
        [
            {
                "longValue": 1
            }
        ]
    ]
}
```

```

        },
        {
            "stringValue": "ValueOne"
        }
    ],
    [
        {
            "longValue": 2
        },
        {
            "stringValue": "ValueTwo"
        }
    ],
    [
        {
            "longValue": 3
        },
        {
            "stringValue": "ValueThree"
        }
    ]
}

```

The following CLI command runs a single SQL statement in a transaction by specifying the --transaction-id option.

For Linux, macOS, or Unix:

```
aws rds-data execute-statement --resource-arn "arn:aws:rds:us-east-1:123456789012:cluster:mydbcluster" \
--database "mydb" --secret-arn "arn:aws:secretsmanager:us-east-1:123456789012:secret:mysecret" \
--sql "update mytable set quantity=5 where id=201" --transaction-id "ABC1234567890xyz"
```

For Windows:

```
aws rds-data execute-statement --resource-arn "arn:aws:rds:us-east-1:123456789012:cluster:mydbcluster" ^
--database "mydb" --secret-arn "arn:aws:secretsmanager:us-east-1:123456789012:secret:mysecret" ^
--sql "update mytable set quantity=5 where id=201" --transaction-id "ABC1234567890xyz"
```

The following is an example of the response.

```
{
    "numberOfRecordsUpdated": 1
}
```

The following CLI command runs a single SQL statement with parameters.

For Linux, macOS, or Unix:

```
aws rds-data execute-statement --resource-arn "arn:aws:rds:us-east-1:123456789012:cluster:mydbcluster" \
--database "mydb" --secret-arn "arn:aws:secretsmanager:us-east-1:123456789012:secret:mysecret" \
--sql "insert into mytable values (:id, :val)" --parameters "[{\\"name\\": \"id\", \\"value\\": \\"longValue\\": 1}, {\\"name\\": \"val\", \\"value\\": {\"stringValue\": \"value1\"}}]"
```

For Windows:

```
aws rds-data execute-statement --resource-arn "arn:aws:rds:us-east-1:123456789012:cluster:mydbcluster" ^
--database "mydb" --secret-arn "arn:aws:secretsmanager:us-east-1:123456789012:secret:mysecret" ^
--sql "insert into mytable values (:id, :val)" --parameters "[{"name": "\id", "value": "\longValue": 1}, {"name": "\val", "value": {"stringValue": "\value1"}}]"
```

The following is an example of the response.

```
{  
    "numberOfRecordsUpdated": 1  
}
```

The following CLI command runs a data definition language (DDL) SQL statement. The DDL statement renames column `job` to column `role`.

Important

For DDL statements, we recommend continuing to run the statement after the call times out. When a DDL statement terminates before it is finished running, it can result in errors and possibly corrupted data structures. To continue running a statement after a call times out, specify the `--continue-after-timeout` option.

For Linux, macOS, or Unix:

```
aws rds-data execute-statement --resource-arn "arn:aws:rds:us-east-1:123456789012:cluster:mydbcluster" ^
--database "mydb" --secret-arn "arn:aws:secretsmanager:us-east-1:123456789012:secret:mysecret" ^
--sql "alter table mytable change column job role varchar(100)" --continue-after-timeout
```

For Windows:

```
aws rds-data execute-statement --resource-arn "arn:aws:rds:us-east-1:123456789012:cluster:mydbcluster" ^
--database "mydb" --secret-arn "arn:aws:secretsmanager:us-east-1:123456789012:secret:mysecret" ^
--sql "alter table mytable change column job role varchar(100)" --continue-after-timeout
```

The following is an example of the response.

```
{  
    "generatedFields": [],  
    "numberOfRecordsUpdated": 0  
}
```

Note

The `generatedFields` data isn't supported by Aurora PostgreSQL. To get the values of generated fields, use the `RETURNING` clause. For more information, see [Returning data from modified rows](#) in the PostgreSQL documentation.

Running a batch SQL statement over an array of data

You can run a batch SQL statement over an array of data by using the `aws rds-data batch-execute-statement` CLI command. You can use this command to perform a bulk import or update operation.

You can run the SQL statement in a transaction by specifying the transaction identifier with the `--transaction-id` option. You can start a transaction by using the `aws rds-data begin-transaction` CLI command. You can end and commit a transaction by using the `aws rds-data commit-transaction` CLI command.

Important

If you don't specify the `--transaction-id` option, changes that result from the call are committed automatically.

In addition to the common options, specify the following options:

- `--sql` (required) – A SQL statement to run on the DB cluster.

Tip

For MySQL-compatible statements, don't include a semicolon at the end of the `--sql` parameter. A trailing semicolon might cause a syntax error.

- `--transaction-id` (optional) – The identifier of a transaction that was started using the `begin-transaction` CLI command. Specify the transaction ID of the transaction that you want to include the SQL statement in.
- `--parameter-set` (optional) – The parameter sets for the batch operation.
- `--database` (optional) – The name of the database.

The DB cluster returns a response to the call.

Note

There isn't a fixed upper limit on the number of parameter sets. However, the maximum size of the HTTP request submitted through the Data API is 4 MiB. If the request exceeds this limit, the Data API returns an error and doesn't process the request. This 4 MiB limit includes the size of the HTTP headers and the JSON notation in the request. Thus, the number of parameter sets that you can include depends on a combination of factors, such as the size of the SQL statement and the size of each parameter set.

The response size limit is 1 MiB. If the call returns more than 1 MiB of response data, the call is terminated.

The maximum number of requests per second is 1,000.

For example, the following CLI command runs a batch SQL statement over an array of data with a parameter set.

For Linux, macOS, or Unix:

```
aws rds-data batch-execute-statement --resource-arn "arn:aws:rds:us-east-1:123456789012:cluster:mydbcluster" \
--database "mydb" --secret-arn "arn:aws:secretsmanager:us-east-1:123456789012:secret:mysecret" \
--sql "insert into mytable values (:id, :val)" \
--parameter-sets "[[{"name": "\id", "value": {"longValue": 1}}, {"name": "\val", "value": {"stringValue": "ValueOne"}}, {"name": "\id", "value": {"longValue": 2}}, {"name": "\val", "value": {"stringValue": "ValueTwo"}}, {"name": "\id", "value": {"longValue": 3}}, {"name": "\val", "value": {"stringValue": "ValueThree"}}]]"
```

For Windows:

```
aws rds-data batch-execute-statement --resource-arn "arn:aws:rds:us-east-1:123456789012:cluster:mydbcluster" ^
--database "mydb" --secret-arn "arn:aws:secretsmanager:us-east-1:123456789012:secret:mysecret" ^
```

```
--sql "insert into mytable values (:id, :val)" ^
--parameter-sets "[[{"name": "\$id", "value": {"longValue": 1}}, {"name": "\$val",
  "value": {"stringValue": "ValueOne"}}], [{"name": "\$id", "value": {"longValue": 2}}, {"name": "\$val", "value": {"stringValue": "ValueTwo"}}], [{"name": "\$id", "value": {"longValue": 3}}, {"name": "\$val", "value": {"stringValue": "ValueThree"}}]]"
```

Note

Don't include line breaks in the --parameter-sets option.

Committing a SQL transaction

Using the aws rds-data commit-transaction CLI command, you can end a SQL transaction that you started with aws rds-data begin-transaction and commit the changes.

In addition to the common options, specify the following option:

- **--transaction-id** (required) – The identifier of a transaction that was started using the begin-transaction CLI command. Specify the transaction ID of the transaction that you want to end and commit.

For example, the following CLI command ends a SQL transaction and commits the changes.

For Linux, macOS, or Unix:

```
aws rds-data commit-transaction --resource-arn "arn:aws:rds:us-
east-1:123456789012:cluster:mydbcluster" \
--secret-arn "arn:aws:secretsmanager:us-east-1:123456789012:secret:mysecret" \
--transaction-id "ABC1234567890xyz"
```

For Windows:

```
aws rds-data commit-transaction --resource-arn "arn:aws:rds:us-
east-1:123456789012:cluster:mydbcluster" ^
--secret-arn "arn:aws:secretsmanager:us-east-1:123456789012:secret:mysecret" ^
--transaction-id "ABC1234567890xyz"
```

The following is an example of the response.

```
{  
    "transactionStatus": "Transaction Committed"  
}
```

Rolling back a SQL transaction

Using the aws rds-data rollback-transaction CLI command, you can roll back a SQL transaction that you started with aws rds-data begin-transaction. Rolling back a transaction cancels its changes.

Important

If the transaction ID has expired, the transaction was rolled back automatically. In this case, an aws rds-data rollback-transaction command that specifies the expired transaction ID returns an error.

In addition to the common options, specify the following option:

- **--transaction-id** (required) – The identifier of a transaction that was started using the `begin-transaction` CLI command. Specify the transaction ID of the transaction that you want to roll back.

For example, the following AWS CLI command rolls back a SQL transaction.

For Linux, macOS, or Unix:

```
aws rds-data rollback-transaction --resource-arn "arn:aws:rds:us-east-1:123456789012:cluster:mydbcluster" \
--secret-arn "arn:aws:secretsmanager:us-east-1:123456789012:secret:mysecret" \
--transaction-id "ABC1234567890xyz"
```

For Windows:

```
aws rds-data rollback-transaction --resource-arn "arn:aws:rds:us-east-1:123456789012:cluster:mydbcluster" ^
--secret-arn "arn:aws:secretsmanager:us-east-1:123456789012:secret:mysecret" ^
--transaction-id "ABC1234567890xyz"
```

The following is an example of the response.

```
{  
    "transactionStatus": "Rollback Complete"  
}
```

Calling the Data API from a Python application

You can call the Data API from a Python application.

The following examples use the AWS SDK for Python (Boto). For more information about Boto, see the [AWS SDK for Python \(Boto 3\) documentation](#).

In each example, replace the DB cluster's Amazon Resource Name (ARN) with the ARN for your Aurora Serverless DB cluster. Also, replace the secret ARN with the ARN of the secret in Secrets Manager that allows access to the DB cluster.

Topics

- [Running a SQL query \(p. 194\)](#)
- [Running a DML SQL statement \(p. 195\)](#)
- [Running a SQL transaction \(p. 196\)](#)

Running a SQL query

You can run a `SELECT` statement and fetch the results with a Python application.

The following example runs a SQL query.

```
import boto3  
  
rdsData = boto3.client('rds-data')  
  
cluster_arn = 'arn:aws:rds:us-east-1:123456789012:cluster:mydbcluster'  
secret_arn = 'arn:aws:secretsmanager:us-east-1:123456789012:secret:mysecret'  
  
response1 = rdsData.execute_statement(  
    resourceArn = cluster_arn,  
    secretArn = secret_arn,
```

```
        database = 'mydb',
        sql = 'select * from employees limit 3')

print (response1['records'])
[
    [
        {
            'longValue': 1
        },
        {
            'stringValue': 'ROSALEZ'
        },
        {
            'stringValue': 'ALEJANDRO'
        },
        {
            'stringValue': '2016-02-15 04:34:33.0'
        }
    ],
    [
        {
            'longValue': 1
        },
        {
            'stringValue': 'DOE'
        },
        {
            'stringValue': 'JANE'
        },
        {
            'stringValue': '2014-05-09 04:34:33.0'
        }
    ],
    [
        {
            'longValue': 1
        },
        {
            'stringValue': 'STILES'
        },
        {
            'stringValue': 'JOHN'
        },
        {
            'stringValue': '2017-09-20 04:34:33.0'
        }
    ]
]
```

Running a DML SQL statement

You can run a data manipulation language (DML) statement to insert, update, or delete data in your database. You can also use parameters in DML statements.

Important

If a call isn't part of a transaction because it doesn't include the `transactionID` parameter, changes that result from the call are committed automatically.

The following example runs an insert SQL statement and uses parameters.

```
import boto3

cluster_arn = 'arn:aws:rds:us-east-1:123456789012:cluster:mydbcluster'
secret_arn = 'arn:aws:secretsmanager:us-east-1:123456789012:secret:mysecret'
```

```
rdsData = boto3.client('rds-data')

param1 = {'name':'firstname', 'value':{'stringValue': 'JACKSON'}}
param2 = {'name':'lastname', 'value':{'stringValue': 'MATEO'}}
paramSet = [param1, param2]

response2 = rdsData.execute_statement(resourceArn=cluster_arn,
                                      secretArn=secret_arn,
                                      database='mydb',
                                      sql='insert into employees(first_name, last_name)
VALUES(:firstname, :lastname)',
                                      parameters = paramSet)

print (response2["numberOfRecordsUpdated"])
```

Running a SQL transaction

You can start a SQL transaction, run one or more SQL statements, and then commit the changes with a Python application.

Important

A transaction times out if there are no calls that use its transaction ID in three minutes. If a transaction times out before it's committed, it's rolled back automatically.
If you don't specify a transaction ID, changes that result from the call are committed automatically.

The following example runs a SQL transaction that inserts a row in a table.

```
import boto3

rdsData = boto3.client('rds-data')

cluster_arn = 'arn:aws:rds:us-east-1:123456789012:cluster:mydbcluster'
secret_arn = 'arn:aws:secretsmanager:us-east-1:123456789012:secret:mysecret'

tr = rdsData.begin_transaction(
    resourceArn = cluster_arn,
    secretArn = secret_arn,
    database = 'mydb')

response3 = rdsData.execute_statement(
    resourceArn = cluster_arn,
    secretArn = secret_arn,
    database = 'mydb',
    sql = 'insert into employees(first_name, last_name) values('XIULAN', 'WANG')',
    transactionId = tr['transactionId'])

cr = rdsData.commit_transaction(
    resourceArn = cluster_arn,
    secretArn = secret_arn,
    transactionId = tr['transactionId'])

cr['transactionStatus']
'Transaction Committed'

response3['numberOfRecordsUpdated']
1
```

Note

If you run a data definition language (DDL) statement, we recommend continuing to run the statement after the call times out. When a DDL statement terminates before it is finished

running, it can result in errors and possibly corrupted data structures. To continue running a statement after a call times out, set the `continueAfterTimeout` parameter to `true`.

Calling the Data API from a Java application

You can call the Data API from a Java application.

The following examples use the AWS SDK for Java. For more information, see the [AWS SDK for Java Developer Guide](#).

In each example, replace the DB cluster's Amazon Resource Name (ARN) with the ARN for your Aurora Serverless DB cluster. Also, replace the secret ARN with the ARN of the secret in Secrets Manager that allows access to the DB cluster.

Topics

- [Running a SQL query \(p. 197\)](#)
- [Running a SQL transaction \(p. 198\)](#)
- [Running a batch SQL operation \(p. 198\)](#)

Running a SQL query

You can run a `SELECT` statement and fetch the results with a Java application.

The following example runs a SQL query.

```
package com.amazonaws.rdsdata.examples;

import com.amazonaws.services.rdsdata.AWSRDSData;
import com.amazonaws.services.rdsdata.AWSRDSDataClient;
import com.amazonaws.services.rdsdata.model.ExecuteStatementRequest;
import com.amazonaws.services.rdsdata.model.ExecuteStatementResult;
import com.amazonaws.services.rdsdata.model.Field;

import java.util.List;

public class FetchResultsExample {
    public static final String RESOURCE_ARN = "arn:aws:rds:us-
east-1:123456789012:cluster:mydbcluster";
    public static final String SECRET_ARN = "arn:aws:secretsmanager:us-
east-1:123456789012:secret:mysecret";

    public static void main(String[] args) {
        AWSRDSData rdsData = AWSRDSDataClient.builder().build();

        ExecuteStatementRequest request = new ExecuteStatementRequest()
            .withResourceArn(RESOURCE_ARN)
            .withSecretArn(SECRET_ARN)
            .withDatabase("mydb")
            .withSql("select * from mytable");

        ExecuteStatementResult result = rdsData.executeStatement(request);

        for (List<Field> fields: result.getRecords()) {
            String stringValue = fields.get(0).getStringValue();
            long numberValue = fields.get(1).getLongValue();

            System.out.println(String.format("Fetched row: string = %s, number = %d",
                stringValue, numberValue));
        }
    }
}
```

Running a SQL transaction

You can start a SQL transaction, run one or more SQL statements, and then commit the changes with a Java application.

Important

A transaction times out if there are no calls that use its transaction ID in three minutes. If a transaction times out before it's committed, it's rolled back automatically.
If you don't specify a transaction ID, changes that result from the call are committed automatically.

The following example runs a SQL transaction.

```
package com.amazonaws.rdsdata.examples;

import com.amazonaws.services.rdsdata.AWSRDSData;
import com.amazonaws.services.rdsdata.AWSRDSDataClient;
import com.amazonaws.services.rdsdata.model.BeginTransactionRequest;
import com.amazonaws.services.rdsdata.model.BeginTransactionResult;
import com.amazonaws.services.rdsdata.model.CommitTransactionRequest;
import com.amazonaws.services.rdsdata.model.ExecuteStatementRequest;

public class TransactionExample {
    public static final String RESOURCE_ARN = "arn:aws:rds:us-
east-1:123456789012:cluster:mydbcluster";
    public static final String SECRET_ARN = "arn:aws:secretsmanager:us-
east-1:123456789012:secret:mysecret";

    public static void main(String[] args) {
        AWSRDSData rdsData = AWSRDSDataClient.builder().build();

        BeginTransactionRequest beginTransactionRequest = new BeginTransactionRequest()
            .withResourceArn(RESOURCE_ARN)
            .withSecretArn(SECRET_ARN)
            .withDatabase("mydb");
        BeginTransactionResult beginTransactionResult =
        rdsData.beginTransaction(beginTransactionRequest);
        String transactionId = beginTransactionResult.getTransactionId();

        ExecuteStatementRequest executeStatementRequest = new ExecuteStatementRequest()
            .withTransactionId(transactionId)
            .withResourceArn(RESOURCE_ARN)
            .withSecretArn(SECRET_ARN)
            .withSql("INSERT INTO test_table VALUES ('hello world!')");
        rdsData.executeStatement(executeStatementRequest);

        CommitTransactionRequest commitTransactionRequest = new CommitTransactionRequest()
            .withTransactionId(transactionId)
            .withResourceArn(RESOURCE_ARN)
            .withSecretArn(SECRET_ARN);
        rdsData.commitTransaction(commitTransactionRequest);
    }
}
```

Note

If you run a data definition language (DDL) statement, we recommend continuing to run the statement after the call times out. When a DDL statement terminates before it is finished running, it can result in errors and possibly corrupted data structures. To continue running a statement after a call times out, set the `continueAfterTimeout` parameter to `true`.

Running a batch SQL operation

You can run bulk insert and update operations over an array of data with a Java application. You can run a DML statement with array of parameter sets.

Important

If you don't specify a transaction ID, changes that result from the call are committed automatically.

The following example runs a batch insert operation.

```
package com.amazonaws.rdsdata.examples;

import com.amazonaws.services.rdsdata.AWSRDSData;
import com.amazonaws.services.rdsdata.AWSRDSDataClient;
import com.amazonaws.services.rdsdata.model.BatchExecuteStatementRequest;
import com.amazonaws.services.rdsdata.model.Field;
import com.amazonaws.services.rdsdata.model.SqlParameter;

import java.util.Arrays;

public class BatchExecuteExample {
    public static final String RESOURCE_ARN = "arn:aws:rds:us-
east-1:123456789012:cluster:mydbcluster";
    public static final String SECRET_ARN = "arn:aws:secretsmanager:us-
east-1:123456789012:secret:mysecret";

    public static void main(String[] args) {
        AWSRDSData rdsData = AWSRDSDataClient.builder().build();

        BatchExecuteStatementRequest request = new BatchExecuteStatementRequest()
            .withDatabase("test")
            .withResourceArn(RESOURCE_ARN)
            .withSecretArn(SECRET_ARN)
            .withSql("INSERT INTO test_table2 VALUES (:string, :number)")
            .withParameterSets(Arrays.asList(
                Arrays.asList(
                    new SqlParameter().withName("string").WithValue(new
Field().withStringValue("Hello")),
                    new SqlParameter().withName("number").WithValue(new
Field().withLongValue(1L))
                ),
                Arrays.asList(
                    new SqlParameter().withName("string").WithValue(new
Field().withStringValue("World")),
                    new SqlParameter().withName("number").WithValue(new
Field().withLongValue(2L))
                )
            ));
        rdsData.batchExecuteStatement(request);
    }
}
```

Using the Java client library for Data API

You can download and use a Java client library for the Data API. This Java client library provides an alternative way to use the Data API. Using this library, you can map your client-side classes to requests and responses of the Data API. This mapping support can ease integration with some specific Java types, such as Date, Time, and BigDecimal.

Downloading the Java client library for Data API

The Data API Java client library is open source in GitHub at the following location:

<https://github.com/awslabs/rds-data-api-client-library-java>

You can build the library manually from the source files, but the best practice is to consume the library using Apache Maven dependency management. Add the following dependency to your Maven POM file.

For version 2.x, which is compatible with AWS SDK 2.x, use the following:

```
<dependency>
    <groupId>software.amazon.rdsdata</groupId>
    <artifactId>rds-data-api-client-library-java</artifactId>
    <version>2.0.0</version>
</dependency>
```

For version 1.x, which is compatible with AWS SDK 1.x, use the following:

```
<dependency>
    <groupId>software.amazon.rdsdata</groupId>
    <artifactId>rds-data-api-client-library-java</artifactId>
    <version>1.0.8</version>
</dependency>
```

Java client library examples

Following, you can find some common examples of using the Data API Java client library. These examples assume that you have a table accounts with two columns: accountID and name. You also have the following data transfer object (DTO).

```
public class Account {
    int accountId;
    String name;
    // getters and setters omitted
}
```

The client library enables you to pass DTOs as input parameters. The following example shows how customer DTOs are mapped to input parameters sets.

```
var account1 = new Account(1, "John");
var account2 = new Account(2, "Mary");
client.forSql("INSERT INTO accounts(accountId, name) VALUES(:accountId, :name)")
    .withParamSets(account1, account2)
    .execute();
```

In some cases, it's easier to work with simple values as input parameters. You can do so with the following syntax.

```
client.forSql("INSERT INTO accounts(accountId, name) VALUES(:accountId, :name)")
    .withParameter("accountId", 3)
    .withParameter("name", "Zhang")
    .execute();
```

The following is another example that works with simple values as input parameters.

```
client.forSql("INSERT INTO accounts(accountId, name) VALUES(?, ?)", 4, "Carlos")
    .execute();
```

The client library provides automatic mapping to DTOs when a result is returned. The following examples show how the result is mapped to your DTOs.

```
List<Account> result = client.forSql("SELECT * FROM accounts")
    .execute()
    .mapToList(Account.class);

Account result = client.forSql("SELECT * FROM accounts WHERE account_id = 1")
    .execute()
    .mapToSingle(Account.class);
```

In many cases, the database result set contains only a single value. In order to simplify retrieving such results, the client library offers the following API:

```
int numberOfAccounts = client.forSql("SELECT COUNT(*) FROM accounts")
    .execute()
    .singleValue(Integer.class);
```

Troubleshooting Data API issues

Use the following sections, titled with common error messages, to help troubleshoot problems that you have with the Data API.

Topics

- [Transaction <transaction_ID> is not found \(p. 201\)](#)
- [Packet for query is too large \(p. 201\)](#)
- [Database response exceeded size limit \(p. 201\)](#)
- [HttpEndpoint is not enabled for cluster <cluster_ID> \(p. 202\)](#)

Transaction <transaction_ID> is not found

In this case, the transaction ID specified in a Data API call wasn't found. The cause for this issue is almost always one of the following:

- The specified transaction ID wasn't created by a [BeginTransaction](#) call.
- The specified transaction ID has expired.

A transaction expires if no call uses the transaction ID within three minutes.

To solve the issue, make sure that your call has a valid transaction ID. Also make sure that each transaction call runs within three minutes of the last one.

For information about running transactions, see [Calling the Data API \(p. 186\)](#).

Packet for query is too large

In this case, the result set returned for a row was too large. The Data API size limit is 64 KB per row in the result set returned by the database.

To solve this issue, make sure that each row in a result set is 64 KB or less.

Database response exceeded size limit

In this case, the size of the result set returned by the database was too large. The Data API limit is 1 MiB in the result set returned by the database.

To solve this issue, make sure that calls to the Data API return 1 MiB of data or less. If you need to return more than 1 MiB, you can use multiple [ExecuteStatement](#) calls with the `LIMIT` clause in your query.

For more information about the `LIMIT` clause, see [SELECT syntax](#) in the MySQL documentation.

HttpEndpoint is not enabled for cluster <cluster_ID>

The cause for this issue is almost always one of the following:

- The Data API isn't enabled for the Aurora Serverless DB cluster. To use the Data API with an Aurora Serverless DB cluster, the Data API must be enabled for the DB cluster.
- The DB cluster was renamed after the Data API was enabled for it.

If the Data API has not been enabled for the DB cluster, enable it.

If the DB cluster was renamed after the Data API was enabled for the DB cluster, disable the Data API and then enable it again.

For information about enabling the Data API, see [Enabling the Data API \(p. 182\)](#).

Logging Data API calls with AWS CloudTrail

Data API is integrated with AWS CloudTrail, a service that provides a record of actions taken by a user, role, or an AWS service in Data API. CloudTrail captures all API calls for Data API as events, including calls from the Amazon RDS console and from code calls to the Data API operations. If you create a trail, you can enable continuous delivery of CloudTrail events to an Amazon S3 bucket, including events for Data API. If you don't configure a trail, you can still view the most recent events in the CloudTrail console in **Event history**. Using the data collected by CloudTrail, you can determine a lot of information. This information includes the request that was made to Data API, the IP address the request was made from, who made the request, when it was made, and additional details.

To learn more about CloudTrail, see the [AWS CloudTrail User Guide](#).

Working with Data API information in CloudTrail

CloudTrail is enabled on your AWS account when you create the account. When activity occurs in Data API, that activity is recorded in a CloudTrail event along with other AWS service events in **Event history**. You can view, search, and download recent events in your AWS account. For more information, see [Viewing events with CloudTrail event history](#) in the [AWS CloudTrail User Guide](#).

For an ongoing record of events in your AWS account, including events for Data API, create a trail. A *trail* enables CloudTrail to deliver log files to an Amazon S3 bucket. By default, when you create a trail in the console, the trail applies to all AWS Regions. The trail logs events from all AWS Regions in the AWS partition and delivers the log files to the Amazon S3 bucket that you specify. Additionally, you can configure other AWS services to further analyze and act upon the event data collected in CloudTrail logs. For more information, see the following topics in the [AWS CloudTrail User Guide](#):

- [Overview for creating a trail](#)
- [CloudTrail supported services and integrations](#)
- [Configuring Amazon SNS notifications for CloudTrail](#)
- [Receiving CloudTrail log files from multiple Regions](#) and [Receiving CloudTrail log files from multiple accounts](#)

All Data API operations are logged by CloudTrail and documented in the [Amazon RDS data service API reference](#). For example, calls to the `BatchExecuteStatement`, `BeginTransaction`, `CommitTransaction`, and `ExecuteStatement` operations generate entries in the CloudTrail log files.

Every event or log entry contains information about who generated the request. The identity information helps you determine the following:

- Whether the request was made with root or IAM user credentials.
- Whether the request was made with temporary security credentials for a role or federated user.
- Whether the request was made by another AWS service.

For more information, see the [CloudTrail userIdentity element](#).

Understanding Data API log file entries

A *trail* is a configuration that enables delivery of events as log files to an Amazon S3 bucket that you specify. CloudTrail log files contain one or more log entries. An *event* represents a single request from any source and includes information about the requested action, the date and time of the action, request parameters, and so on. CloudTrail log files aren't an ordered stack trace of the public API calls, so they don't appear in any specific order.

The following example shows a CloudTrail log entry that demonstrates the `ExecuteStatement` operation.

```
{  
    "eventVersion": "1.05",  
    "userIdentity": {  
        "type": "IAMUser",  
        "principalId": "AKIAIOSFODNN7EXAMPLE",  
        "arn": "arn:aws:iam::123456789012:user/johndoe",  
        "accountId": "123456789012",  
        "accessKeyId": "AKIAI44QH8DHBEEXAMPLE",  
        "userName": "johndoe"  
    },  
    "eventTime": "2019-12-18T00:49:34Z",  
    "eventSource": "rdsdata.amazonaws.com",  
    "eventName": "ExecuteStatement",  
    "awsRegion": "us-east-1",  
    "sourceIPAddress": "192.0.2.0",  
    "userAgent": "aws-cli/1.16.102 Python/3.7.2 Windows/10 botocore/1.12.92",  
    "requestParameters": {  
        "continueAfterTimeout": false,  
        "database": "*****",  
        "includeResultMetadata": false,  
        "parameters": [],  
        "resourceArn": "arn:aws:rds:us-east-1:123456789012:cluster:my-database-1",  
        "schema": "*****",  
        "secretArn": "arn:aws:secretsmanager:us-east-1:123456789012:secret:dataapisecret-  
ABC123",  
        "sql": "*****"  
    },  
    "responseElements": null,  
    "requestID": "6ba9a36e-b3aa-4ca8-9a2e-15a9eada988e",  
    "eventID": "a2c7a357-ee8e-4755-a0d0-aed11ed4253a",  
    "eventType": "AwsApiCall",  
    "recipientAccountId": "123456789012"  
}
```

Excluding Data API events from an AWS CloudTrail trail

Most Data API users rely on the events in an AWS CloudTrail trail to provide a record of Data API operations. The trail can be a valuable source of data for auditing critical events, such as a SQL

statement that deleted rows in a table. In some cases, the metadata in a CloudTrail log entry can help you to avoid or resolve errors.

However, because the Data API can generate a large number of events, you can exclude Data API events from a CloudTrail trail. This per-trail setting excludes all Data API events. You can't exclude particular Data API events.

To exclude Data API events from a trail, do the following:

- In the CloudTrail console, choose the **Exclude Amazon RDS Data API events** setting when you [create a trail](#) or [update a trail](#).
- In the CloudTrail API, use the [PutEventSelectors](#) operation. Add the `ExcludeManagementEventSources` attribute to your event selectors with a value of `rdsdata.amazonaws.com`. For more information, see [Creating, updating, and managing trails with the AWS Command Line Interface](#) in the *AWS CloudTrail User Guide*.

Warning

Excluding Data API events from a CloudTrail log can obscure Data API actions. Be cautious when giving principals the `cloudtrail:PutEventSelectors` permission that is required to perform this operation.

You can turn off this exclusion at any time by changing the console setting or the event selectors for a trail. The trail will then start recording Data API events. However, it can't recover Data API events that occurred while the exclusion was effective.

When you exclude Data API events by using the console or API, the resulting CloudTrail `PutEventSelectors` API operation is also logged in your CloudTrail logs. If Data API events don't appear in your CloudTrail logs, look for a `PutEventSelectors` event with the `ExcludeManagementEventSources` attribute set to `rdsdata.amazonaws.com`.

For more information, see [Logging management events for trails](#) in the *AWS CloudTrail User Guide*.

Using the query editor for Aurora Serverless

With the query editor for Aurora Serverless, you can run SQL queries in the RDS console. You can run any valid SQL statement on the Aurora Serverless DB cluster, including data manipulation and data definition statements.

The query editor requires an Aurora Serverless DB cluster with the Data API enabled. For information about creating an Aurora Serverless DB cluster with the Data API enabled, see [Using the Data API for Aurora Serverless \(p. 178\)](#).

Availability of the query editor

The query editor is only available for the following Aurora Serverless DB clusters:

- Aurora with MySQL version 5.6 compatibility
- Aurora with MySQL version 5.7 compatibility
- Aurora with PostgreSQL version 10.7 compatibility

The query editor is currently available for Aurora Serverless in the following AWS Regions:

- US East (Ohio)
- US East (N. Virginia)

- US West (N. California)
- US West (Oregon)
- Asia Pacific (Mumbai)
- Asia Pacific (Seoul)
- Asia Pacific (Singapore)
- Asia Pacific (Sydney)
- Asia Pacific (Tokyo)
- Canada (Central)
- Europe (Frankfurt)
- Europe (Ireland)
- Europe (London)
- Europe (Paris)

Authorizing access to the query editor

A user must be authorized to run queries in the query editor. You can authorize a user to run queries in the query editor by adding the `AmazonRDSDataFullAccess` policy, a predefined AWS Identity and Access Management (IAM) policy, to that user.

You can also create an IAM policy that grants access to the query editor. After you create the policy, add it to each user that requires access to the query editor.

The following policy provides the minimum required permissions for a user to access the query editor.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "QueryEditor0",  
            "Effect": "Allow",  
            "Action": [  
                "secretsmanager:GetSecretValue",  
                "secretsmanager:PutResourcePolicy",  
                "secretsmanager:PutSecretValue",  
                "secretsmanager>DeleteSecret",  
                "secretsmanager:DescribeSecret",  
                "secretsmanager:TagResource"  
            ],  
            "Resource": "arn:aws:secretsmanager:*:secret:rds-db-credentials/*"  
        },  
        {  
            "Sid": "QueryEditor1",  
            "Effect": "Allow",  
            "Action": [  
                "secretsmanager:GetRandomPassword",  
                "tag:GetResources",  
                "secretsmanager>CreateSecret",  
                "secretsmanager>ListSecrets",  
                "dbqms>CreateFavoriteQuery",  
                "dbqms:DescribeFavoriteQueries",  
                "dbqms:UpdateFavoriteQuery",  
                "dbqms>DeleteFavoriteQueries",  
                "dbqms:GetQueryString",  
                "dbqms>CreateQueryHistory",  
                "dbqms:UpdateQueryHistory",  
                "dbqms:DeleteQueryHistory"  
            ]  
        }  
    ]  
}
```

```
        "dbqms>DeleteQueryHistory",
        "dbqms>DescribeQueryHistory",
        "rds-data:BatchExecuteStatement",
        "rds-data:BeginTransaction",
        "rds-data:CommitTransaction",
        "rds-data:ExecuteStatement",
        "rds-data:RollbackTransaction"
    ],
    "Resource": "*"
}
]
```

For information about creating an IAM policy, see [Creating IAM policies](#) in the *AWS Identity and Access Management User Guide*.

For information about adding an IAM policy to a user, see [Adding and removing IAM identity permissions](#) in the *AWS Identity and Access Management User Guide*.

Running queries in the query editor

You can run SQL statements on an Aurora Serverless DB cluster in the query editor.

To run a query in the query editor

1. Sign in to the AWS Management Console and open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. In the upper-right corner of the AWS Management Console, choose the AWS Region in which you created the Aurora Serverless DB clusters that you want to query.
3. In the navigation pane, choose **Databases**.
4. Choose the Aurora Serverless DB cluster that you want to run SQL queries against.
5. For **Actions**, choose **Query**. If you haven't connected to the database before, the **Connect to database** page opens.

Connect to database

You need to choose a database and enter the database credentials to use the query editor. We will be storing your credentials and the connection in the AWS Secrets Manager service. [Learn more](#)

Database instance or cluster

database-1

Database username

Add new database credentials

Enter database username

Enter database password

Enter the name of the database or schema (optional)

Enter the name for schemas collection

Enter database or schema name

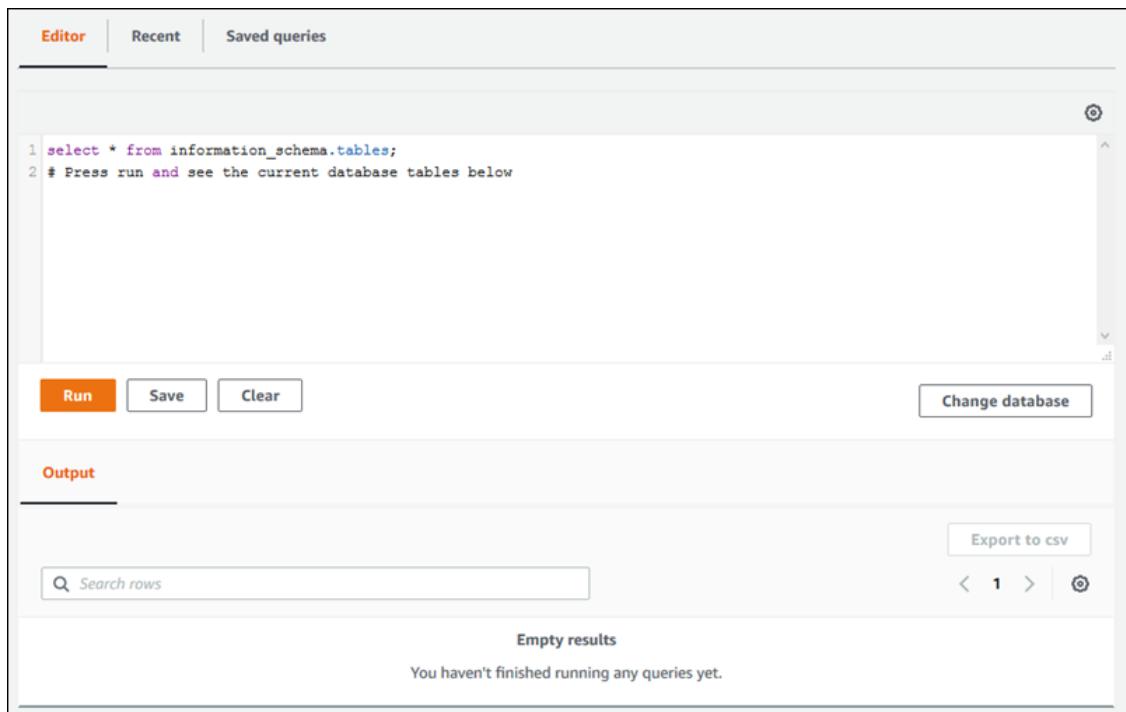
Cancel **Connect to database**

6. Enter the following information:
 - a. For **Database instance or cluster**, choose the Aurora Serverless DB cluster that you want to run SQL queries on.
 - b. For **Database username**, choose the user name of the database user to connect with, or choose **Add new database credentials**. If you choose **Add new database credentials**, enter the user name for the new database credentials in **Enter database username**.
 - c. For **Enter database password**, enter the password for the database user that you chose.
 - d. In the last box, enter the name of the database or schema that you want to use for the Aurora DB cluster.
 - e. Choose **Connect to database**.

Note

If your connection is successful, your connection and authentication information are stored in AWS Secrets Manager. You don't need to enter the connection information again.

7. In the query editor, enter the SQL query that you want to run on the database.



A screenshot of the Amazon Aurora Query Editor interface. At the top, there are three tabs: 'Editor' (which is selected), 'Recent', and 'Saved queries'. Below the tabs, a code editor contains the following SQL script:

```
1 select * from information_schema.tables;
2 # Press run and see the current database tables below
```

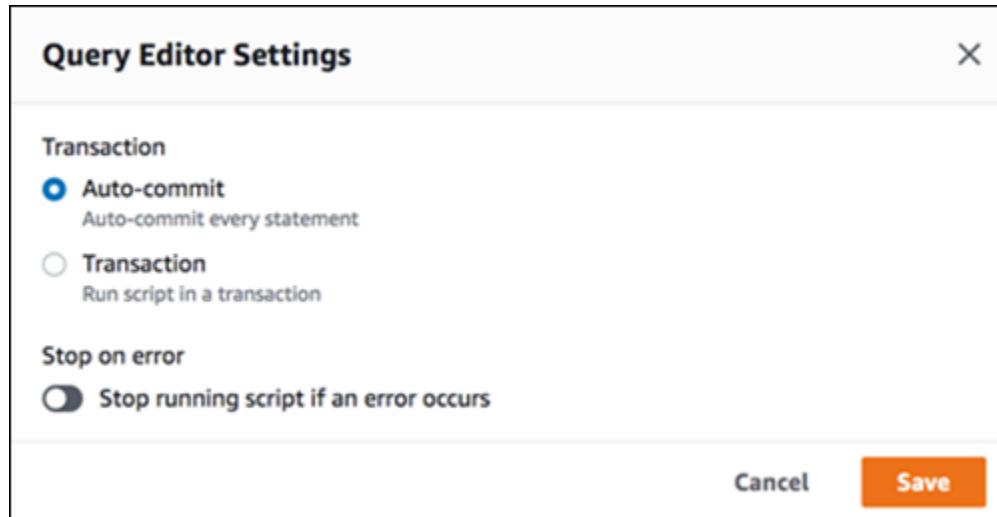
Below the code editor are four buttons: 'Run' (orange), 'Save', 'Clear', and 'Change database'. The 'Output' tab is selected, showing a search bar ('Search rows') and a message: 'Empty results' and 'You haven't finished running any queries yet.' There is also a 'Change database' button and an 'Export to csv' link.

Each SQL statement can commit automatically, or you can run SQL statements in a script as part of a transaction. To control this behavior, choose the gear icon above the query window.



A screenshot of the Amazon Aurora Query Editor interface, similar to the one above but with a red circle highlighting the gear icon located at the top right of the main window area.

The **Query Editor Settings** window appears.



If you choose **Auto-commit**, every SQL statement commits automatically. If you choose **Transaction**, you can run a group of statements in a script. Statements are automatically committed at the end of the script unless explicitly committed or rolled back before then. Also, you can choose to stop a running script if an error occurs by enabling **Stop on error**.

Note

In a group of statements, data definition language (DDL) statements can cause previous data manipulation language (DML) statements to commit. You can also include COMMIT and ROLLBACK statements in a group of statements in a script.

After you make your choices in the **Query Editor Settings** window, choose **Save**.

8. Choose **Run** or press Ctrl+Enter, and the query editor displays the results of your query.

After running the query, save it to **Saved queries** by choosing **Save**.

Export the query results to spreadsheet format by choosing **Export to csv**.

You can find, edit, and rerun previous queries. To do so, choose the **Recent** tab or the **Saved queries** tab, choose the query text, and then choose **Run**.

To change the database, choose **Change database**.

Database Query Metadata Service (DBQMS) API reference

The Database Query Metadata Service (dbqms) is an internal-only service. It provides your recent and saved queries for the query editor on the AWS Management Console for multiple AWS services, including Amazon RDS.

The following DBQMS actions are supported:

Topics

- [CreateFavoriteQuery \(p. 210\)](#)
- [CreateQueryHistory \(p. 210\)](#)
- [CreateTab \(p. 210\)](#)
- [DeleteFavoriteQueries \(p. 210\)](#)
- [DeleteQueryHistory \(p. 210\)](#)
- [DeleteTab \(p. 210\)](#)

- [DescribeFavoriteQueries \(p. 210\)](#)
- [DescribeQueryHistory \(p. 210\)](#)
- [DescribeTabs \(p. 210\)](#)
- [GetQueryString \(p. 210\)](#)
- [UpdateFavoriteQuery \(p. 210\)](#)
- [UpdateQueryHistory \(p. 210\)](#)
- [UpdateTab \(p. 211\)](#)

CreateFavoriteQuery

Save a new favorite query. Each IAM user can create up to 1000 saved queries. This limit is subject to change in the future.

CreateQueryHistory

Save a new query history entry.

CreateTab

Save a new query tab. Each IAM user can create up to 10 query tabs.

DeleteFavoriteQueries

Delete one or more saved queries.

DeleteQueryHistory

Delete query history entries.

DeleteTab

Delete query tab entries.

DescribeFavoriteQueries

List saved queries created by an IAM user in a given account.

DescribeQueryHistory

List query history entries.

DescribeTabs

List query tabs created by an IAM user in a given account.

GetQueryString

Retrieve full query text from a query ID.

UpdateFavoriteQuery

Update the query string, description, name, or expiration date.

UpdateQueryHistory

Update the status of query history.

UpdateTab

Update the query tab name and query string.

Using Amazon Aurora Serverless v2 (preview)

Amazon Aurora Serverless v2 with MySQL compatibility is in preview release and is subject to change. Aurora Serverless v2 (preview) is not covered by the Amazon RDS service level agreement (SLA). Don't use Aurora Serverless v2 (preview) for production databases. All resources and data will be deleted when the preview ends.

Continuously monitoring and adjusting capacity for multiple databases so that you stay within budget can be a daunting task. You don't want to pay for more computing resources than you use. But you also can't afford to spend a lot of time reallocating resources to achieve a better price-performance ratio. For unpredictable workloads, multitenant, and distributed database environments that can have wide variances in consumption, this task is especially challenging.

By using Amazon Aurora Serverless v2 (Amazon Aurora Serverless version 2), now in preview, you can get optimal cost performance for your database clusters. Capacity is adjusted automatically based on application demand, and you're charged only for the resources that your DB clusters consume.

Topics

- [How Aurora Serverless v2 \(preview\) works \(p. 212\)](#)
- [Limitations of Aurora Serverless v2 \(preview\) \(p. 215\)](#)
- [Creating an Aurora Serverless v2 \(preview\) DB cluster \(p. 216\)](#)
- [Creating a snapshot of an Aurora Serverless v2 \(preview\) DB cluster \(p. 219\)](#)
- [Modifying an Aurora Serverless v2 \(preview\) DB cluster \(p. 220\)](#)
- [Deleting an Aurora Serverless v2 \(preview\) DB cluster \(p. 222\)](#)
- [Restoring an Aurora Serverless v2 \(preview\) DB cluster to a point in time \(p. 223\)](#)

How Aurora Serverless v2 (preview) works

Amazon Aurora Serverless v2 with MySQL compatibility is in preview release and is subject to change. Aurora Serverless v2 (preview) is not covered by the Amazon RDS service level agreement (SLA). Don't use Aurora Serverless v2 (preview) for production databases. All resources and data will be deleted when the preview ends.

Amazon Aurora Serverless v2 (preview) has been architected from the ground up to support serverless DB clusters that are instantly scalable. The Aurora Serverless v2 (preview) architecture rests on a lightweight foundation that's engineered to provide the security and isolation needed in multitenant serverless cloud environments. This foundation has very little overhead so it can respond quickly. It's also powerful enough to meet dramatic increases in processing demand.

Instant autoscaling

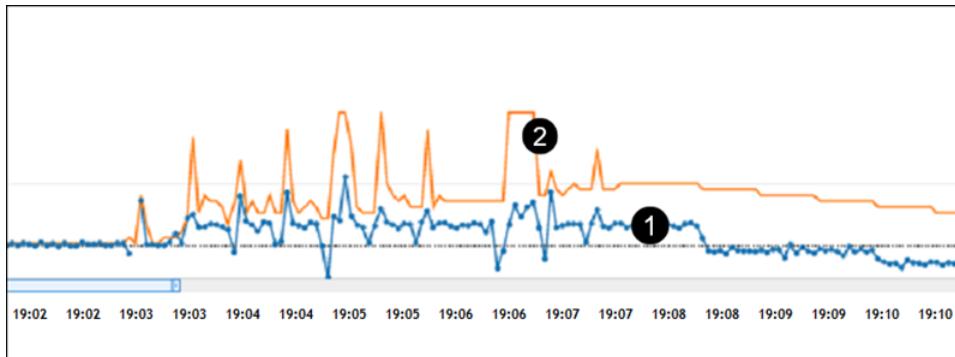
When you create your Aurora Serverless v2 (preview) DB cluster, you define its capacity as a range between minimum and maximum number of Aurora capacity units (ACUs):

- Minimum Aurora capacity units – The smallest number of ACUs down to which your Aurora Serverless v2 (preview) DB cluster can scale.
- Maximum Aurora capacity units – The largest number of ACUs up to which your Aurora Serverless v2 (preview) DB cluster can scale.

Each ACU provides 2 GiB (gibibytes) of memory (RAM) and associated virtual processor (vCPU) with networking.

Unlike Aurora Serverless v1, which scales by doubling ACUs each time the DB cluster reaches a threshold, Aurora Serverless v2 (preview) can increase ACUs incrementally. When your workload demand begins to reach the current resource capacity, your Aurora Serverless v2 (preview) DB cluster scales the number of ACUs. Your cluster scales ACUs in the increments required to provide the best performance for the resources consumed.

The following screenshot shows instant autoscaling in action. It's an extract from Amazon CloudWatch comparing processing load to the number of ACUs consumed by an Aurora Serverless v2 (preview) DB cluster over time for a simulated "flash sale" scenario. The simulation models an order system that processes about 10 orders per second (using 4 ACUs) during regular operations. A load testing tool generates various increases in orders mimicking a "flash sale," until the system is processing 275 orders per second (and 22 ACUs) at its peak.



In the screenshot, these numbers indicate this information:

1. Orders processed each second – Processing load as customers respond to a "flash sale" for a product. The line shows the number of orders being processed each second. Order processing involves multiple database actions. These include checking inventory, processing the new order, creating a shipment order, adjusting the inventory amount, and initiating shipping by notifying the warehouse system.
2. Aurora capacity units (ACUs) – Memory and CPU applied over time to increasing and decreasing demand. The line shows the surge of ACUs applied to the workload (line 1) when orders reach their highest point, about 275 per second.

An ACU is made up of both memory (RAM) and processor (CPU). Increases in CPU utilization respond immediately to workload demands. When the demand starts to decline from its peak, the scale down from the maximum ACU occurs more slowly, as memory is more gradually released (than CPU). This is a deliberate architectural choice. Aurora Serverless v2 (preview) releases memory more gradually as demand lessens to avoid affecting the workload.

Logging with Amazon CloudWatch

As with all Aurora DB clusters, error logs for Aurora Serverless v2 (preview) are enabled by default. However, unlike with provisioned Aurora DB clusters, you can't view the logs for Aurora Serverless v2 (preview) in the Amazon RDS console. Aurora Serverless v2 (preview) automatically uploads the error logs to Amazon CloudWatch.

Aurora Serverless v2 (preview) also uploads your Aurora MySQL log data to CloudWatch for the types of logs that you specify. You choose the logs for uploading by changing values for several log-related DB cluster parameters for your Aurora Serverless v2 (preview) DB cluster. As with any type of Aurora DB cluster, you can't modify the default DB cluster parameter group. Instead, create your own DB cluster parameter group based on a default parameter for your DB cluster and engine type. For Aurora Serverless v2 (preview) and Aurora Serverless v1, you use a DB cluster parameter group only.

We recommend that you create your custom DB cluster parameter group before creating your Aurora Serverless v2 (preview) DB cluster, so that it's available to choose when you create a database on the console.

You can also modify your Aurora Serverless v2 (preview) DB cluster later to use your custom DB cluster parameter group. For more information, see [Modifying your DB cluster to use a custom DB cluster parameter group \(p. 221\)](#).

For Aurora MySQL logging, you can activate the following parameters:

- `general_log` – Set to 1 to turn on the general log (default is off, or 0).
- `slow_query_log` – Set to 1 to turn on the slow query log. (default is off, or 0).
- `server_audit_logging` – Set to 1 to turn on server audit logging. If you turn this on, you can specify the audit events to send to CloudWatch by listing them in the `server_audit_events` parameter.
- `server_audit_events` – The list of events to capture in the logs.

For more information, see [Using Advanced Auditing with an Amazon Aurora MySQL DB cluster \(p. 914\)](#).

After you apply your modified DB cluster parameter group to your Aurora Serverless v2 (preview) DB cluster, you can view the logs in CloudWatch.

To view logs for your Aurora Serverless v2 (preview) DB cluster

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. Choose **US East (N. Virginia)** for the Region.
3. Choose **Log groups**.
4. Choose your Aurora Serverless v2 (preview) DB cluster log from the list. For error logs, the naming pattern is as follows.

```
/aws/rds/cluster/cluster-name/error
```

For more information on viewing details on your logs, see [Monitoring log events in Amazon CloudWatch \(p. 1020\)](#).

Monitoring capacity with Amazon CloudWatch

Aurora Serverless v2 (preview) introduces a new metric for monitoring Aurora DB cluster capacity, `ServerlessDatabaseCapacity`. You can use CloudWatch to view your DB cluster's capacity as it scales up and down. You can also compare `ServerlessDatabaseCapacity` to other metrics to see how changes in workloads affect resource consumption.

To monitor your Aurora Serverless v2 (preview) DB cluster's capacity

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. Choose the US East (N. Virginia) Region.
3. Choose **Metrics**. All available metrics appear as cards in the console, grouped by service name.
4. Choose **RDS**.

You can also use the **Search** box to find `ServerlessDatabaseCapacity`.

We recommend that you set up a CloudWatch dashboard to monitor your Aurora Serverless v2 (preview) DB cluster capacity using this new metric. To learn how, see [Building dashboards with CloudWatch](#). You

can compare `ServerlessDatabaseCapacity` to `DatabaseUsedMemory`, `DatabaseConnections`, and `DMLThroughput` to assess how your DB cluster is responding during operations.

To learn more about using Amazon CloudWatch with Amazon Aurora, see [Publishing Amazon Aurora MySQL logs to Amazon CloudWatch Logs \(p. 1017\)](#).

Limitations of Aurora Serverless v2 (preview)

Amazon Aurora Serverless v2 with MySQL compatibility is in preview release and is subject to change. Aurora Serverless v2 (preview) is not covered by the Amazon RDS service level agreement (SLA). Don't use Aurora Serverless v2 (preview) for production databases. All resources and data will be deleted when the preview ends.

The following limitations apply to Amazon Aurora Serverless v2 (preview):

- You can work with Amazon Aurora Serverless v2 (preview) in the preview environment only. You are limited to working with Aurora Serverless v2 (preview) in this environment. To use any of your other existing AWS services, such as Amazon EC2 and Amazon CloudWatch, access them through the AWS Management Console in the US East (N. Virginia) Region.
- You can work with Amazon Aurora Serverless v2 (preview) using the console only. AWS CLI commands and Amazon RDS API operations for creating and working Aurora Serverless v2 (preview) DB clusters aren't currently available.
- You can create only Aurora MySQL 5.7 (2.07) DB clusters using Aurora Serverless v2 (preview). Aurora PostgreSQL isn't currently available for Aurora Serverless v2 (preview).
- Aurora Serverless v2 (preview) clusters can be created in the Availability Zones with these zoneids only:
 - use1-az2
 - use1-az4
 - use1-az6
- Your Aurora Serverless v2 (preview) DB clusters have a default capacity that ranges from a minimum of 4 Aurora capacity units (ACUs) to 32 ACUs. Each ACU provides the equivalent of approximately 2 gibabytes (GiB) of RAM and associated CPU and networking.
- You can't give an Aurora Serverless v2 (preview) DB cluster a public IP address.
- You must create your Aurora Serverless v2 (preview) DB cluster in a virtual private cloud (VPC) based on the Amazon VPC service. You can access your Aurora Serverless v2 (preview) DB cluster only from within a VPC based on Amazon VPC.
- Aurora Serverless v2 (preview) databases are accessible only through port 3306. Aurora Serverless v2 (preview) assigns port 3306 to any Aurora MySQL DB instances that you create. You can't change the port number.
- Aurora Serverless v2 (preview) doesn't support the following Aurora features:
 - Amazon RDS Performance Insights
 - Amazon RDS Proxy
 - Aurora backtracking
 - Aurora cloning
 - Aurora global databases
 - Aurora multi-master clusters
 - Aurora Replicas
 - AWS Identity and Access Management (IAM) database authentication
 - Data API for Aurora Serverless v1
 - Exporting snapshots created from Aurora Serverless v2 (preview) DB clusters to Amazon S3 buckets

- Importing data from Amazon S3 into Aurora Serverless v2 (preview) DB cluster tables
- Invoking AWS Lambda functions from within your Aurora Serverless v2 (preview) database
- Query editor for Aurora Serverless v1
- Restoring snapshots from Aurora provisioned DB clusters

Creating an Aurora Serverless v2 (preview) DB cluster

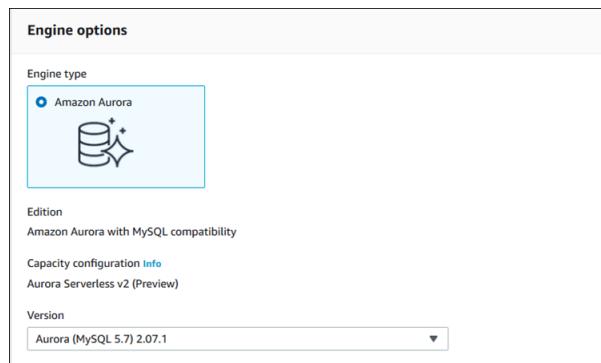
Amazon Aurora Serverless v2 with MySQL compatibility is in preview release and is subject to change. Aurora Serverless v2 (preview) is not covered by the Amazon RDS service level agreement (SLA). Don't use Aurora Serverless v2 (preview) for production databases. All resources and data will be deleted when the preview ends.

To work with Amazon Aurora Serverless v2 (preview), you must apply for access. For more information, see the [Aurora Serverless v2 \(preview\)](#) page.

After you're approved for access, you can sign in to the preview using the console. Currently, you can create an Amazon Aurora Serverless v2 (preview) DB cluster with the console only.

To create an Aurora Serverless v2 (preview) DB cluster

1. Sign in to the preview using the AWS Management Console and open the Amazon RDS console.
2. Choose **Create Database**. For this preview, you see the available choices preselected for **Engine options**:
 - Amazon Aurora for **Engine type**
 - Amazon Aurora with MySQL compatibility for **Edition**
 - Aurora Serverless v2 (preview) for **Capacity configuration**
 - Aurora (MySQL 5.7) 2.07.1 for **Version**



3. For **Settings**, do the following:
 - a. Accept the default DB cluster identifier or choose your own.
 - b. Enter your own password for the default `admin` account for the DB cluster, or have Aurora Serverless v2 (preview) generate one for you. If you choose **Auto generate a password**, you get an option to copy the password.

DB cluster identifier [Info](#)
Type a name for your DB cluster. The name must be unique across all DB clusters owned by your AWS account in the current AWS Region.

The DB cluster identifier is case-insensitive, but is stored as all lowercase (as in "mydbcluster"). Constraints: 1 to 60 alphanumeric characters or hyphens. First character must be a letter. Can't contain two consecutive hyphens. Can't end with a hyphen.

Credentials Settings

Master username [Info](#)
Type a login ID for the master user of your DB instance.

1 to 16 alphanumeric characters. First character must be a letter
 Auto generate a password
Amazon RDS can generate a password for you, or you can specify your own password

Master password [Info](#)

Constraints: At least 8 printable ASCII characters. Can't contain any of the following: / (slash), '(single quote)', "(double quote)" and @ (at sign).
Confirm password [Info](#)

- For **Capacity settings**, you can accept the default range (4 ACUs minimum to 32 ACUs maximum). Or you can choose other values for minimum and maximum capacity units.

For more information about Aurora Serverless v2 (preview) capacity units, see [Instant autoscaling \(p. 212\)](#).

Capacity settings [Info](#)
Specify the minimum and maximum values for the capacity range. Database capacity is measured in Aurora capacity units (ACUs). 1 ACU has approximately 2 GB of memory with corresponding CPU and networking, similar to what is used in Aurora user-provisioned instances.

Minimum Aurora capacity unit 8 GiB	Maximum Aurora capacity unit 64 GiB
<input type="text" value="4"/> ACUs	<input type="text" value="32"/> ACUs

- For **Connectivity**, choose the virtual private cloud (VPC) based on Amazon VPC that defines the virtual networking environment for this DB instance. You can choose the defaults to simplify this task.

To use your own VPC, we recommend that you create it along with related subnets, subnet group, and security group in advance. If you do this, these are available for you to choose when you're creating your Aurora Serverless v2 (preview) DB cluster.

To learn how, see [How to create a VPC for use with Amazon Aurora](#).

Connectivity

Virtual private cloud (VPC) [Info](#)
VPC that defines the virtual networking environment for this DB cluster.

Only VPCs with a corresponding DB subnet group are listed.

After a database is created, you can't change the VPC selection.

Additional connectivity configuration

Subnet group [Info](#)
DB subnet group that defines which subnets and IP ranges the DB instance can use in the VPC you selected.

VPC security group
Choose one or more RDS security groups to allow access to your database. Ensure that the security group rules allow incoming traffic from EC2 instances and devices outside your VPC. (Security groups are required for publicly accessible databases.)

Choose existing
Choose existing VPC security groups

Create new
Create new VPC security group

Existing VPC security groups

6. For **Additional configuration**, enter a name for **Initial database name** to create a database for your Aurora Serverless v2 (preview) cluster.

If you created a custom DB cluster parameter group, choose it for **DB cluster parameter group**. If you want to view your Aurora MySQL logs in Amazon CloudWatch, make sure to use a custom DB cluster parameter group. For more information, see [Logging with Amazon CloudWatch \(p. 213\)](#).

Additional configuration
Database options, encryption enabled, backup enabled, backtrack disabled, delete protection enabled

Database options

Initial database name [Info](#)
 If you do not specify a database name, Amazon RDS does not create a database.

DB cluster parameter group [Info](#)

Backup
Creates a point-in-time snapshot of your database

Backup retention period [Info](#)
Choose the number of days that RDS should retain automatic backups for this instance.

Copy tags to snapshots

Encryption

Master key [Info](#)

Account

KMS key ID

Deletion protection

Enable deletion protection
Protects the database from being deleted accidentally. While this option is enabled, you can't delete the database.

The Aurora Serverless v2 (preview) cluster volume is always encrypted. You can't disable encryption, but you can choose your own encryption key. For more information, see [Encrypting Amazon Aurora resources](#).

7. Acknowledge the limited service agreement.

Aurora Serverless v2 is now available in preview
Aurora Serverless v2 is not covered by the Amazon RDS service level agreement (SLA), published at <https://aws.amazon.com/rds/sla/>.

I acknowledge this limited service agreement for Aurora Serverless v2 and I will not configure Aurora Serverless v2 for production databases.

You are responsible for ensuring that you have all of the necessary rights for any third-party products or services that you use with AWS services.

Create database

8. Choose **Create database** to create your Aurora Serverless v2 (preview) DB cluster.

You can connect to your Aurora Serverless v2 (preview) DB cluster by using its endpoint. The endpoint is listed on the **Connectivity & security** tab of the console, under **Endpoint & Port**. For more information about how to connect to Aurora DB clusters, see [Connecting to an Amazon Aurora DB cluster \(p. 281\)](#).

Aurora Serverless v2 (preview) creates your DB instance using port 3306. Make sure to configure the security group for your Aurora Serverless v2 (preview) DB cluster to allow access to the MySQL/Aurora port (3306).

However, you can't access the Amazon VPC configurations directly from the preview console.

To modify your security group settings

1. Sign in to your <https://console.aws.amazon.com/vpc/>.
2. Choose the US East (N. Virginia) Region.

3. For **Security Group**, choose the security group associated with your Aurora Serverless v2 (preview) DB cluster.
4. Edit values for **Inbound rules** and **Outbound rules** as needed.

To learn more about configuring your VPC for Aurora, see [Amazon Virtual Private Cloud VPCs and Amazon Aurora](#).

Creating a snapshot of an Aurora Serverless v2 (preview) DB cluster

Amazon Aurora Serverless v2 with MySQL compatibility is in preview release and is subject to change. Aurora Serverless v2 (preview) is not covered by the Amazon RDS service level agreement (SLA). Don't use Aurora Serverless v2 (preview) for production databases. All resources and data will be deleted when the preview ends.

Amazon Aurora Serverless v2 (preview) routinely creates snapshots of your DB cluster as backups. Unlike automated backups, manual snapshots aren't subject to the backup retention period. Snapshots don't expire. For more information about snapshots in general, see [Creating a DB cluster snapshot \(p. 495\)](#).

Currently, you can create an Aurora Serverless v2 (preview) DB cluster snapshot using the console only.

To create a DB cluster snapshot

1. Sign in to the preview using the AWS Management Console and open the Amazon RDS console.
2. In the navigation pane, choose **Databases**.
3. Choose the Aurora Serverless v2 (preview) DB cluster use for your snapshot.
4. Choose **Actions**, and then choose **Take snapshot**.

The **Take DB Snapshot** window appears.

5. For **Snapshot name**, enter the name of your Aurora Serverless v2 (preview) DB cluster.

Take DB Snapshot

Settings
To take a snapshot of this DB instance you must provide a name for the snapshot.

DB instance
The unique key that identifies a DB instance. This parameter isn't case-sensitive.
cluster5

Snapshot name
The Identifier for the DB Snapshot.

Cancel **Take Snapshot**

6. Choose **Take Snapshot**.

Modifying an Aurora Serverless v2 (preview) DB cluster

Amazon Aurora Serverless v2 with MySQL compatibility is in preview release and is subject to change. Aurora Serverless v2 (preview) is not covered by the Amazon RDS service level agreement (SLA). Don't use Aurora Serverless v2 (preview) for production databases. All resources and data will be deleted when the preview ends.

You can change configuration settings for your Aurora Serverless v2 (preview) DB cluster at any time, such as to do the following:

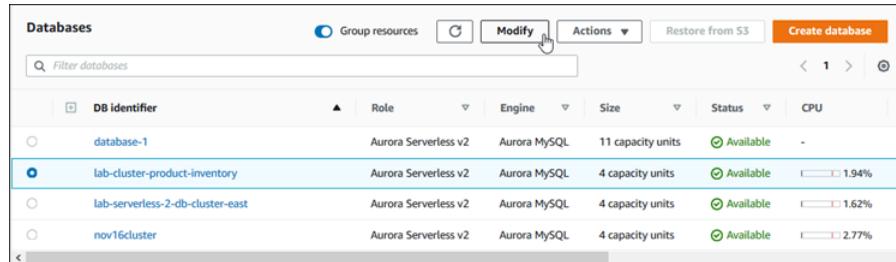
- Change your Aurora Serverless v2 (preview) DB cluster name.
- Turn on (or off) deletion protection for your Aurora Serverless v2 (preview) DB cluster.
- Modify your Aurora Serverless v2 (preview) DB cluster's capacity settings.
- Modify **Additional configuration** settings, such as choosing a custom DB cluster parameter group.

The only configuration option that you can't change for Aurora Serverless v2 (preview) DB cluster is its virtual private cloud (VPC) that you chose when you created it.

Use the following procedure to modify your Aurora Serverless v2 (preview) DB cluster's configuration by using the console.

To modify the configuration of your Aurora Serverless v2 (preview) DB cluster

1. Sign in to the preview using the AWS Management Console and open the Amazon RDS console.
2. In the navigation pane, choose **DB clusters**.
3. Choose your Aurora Serverless v2 (preview) DB cluster from the list.



DB identifier	Role	Engine	Size	Status	CPU
database-1	Aurora Serverless v2	Aurora MySQL	11 capacity units	Available	-
lab-cluster-product-inventory	Aurora Serverless v2	Aurora MySQL	4 capacity units	Available	1.94%
lab-serverless-2-db-cluster-east	Aurora Serverless v2	Aurora MySQL	4 capacity units	Available	1.62%
nov16cluster	Aurora Serverless v2	Aurora MySQL	4 capacity units	Available	2.77%

The configuration settings for your Aurora Serverless v2 (preview) DB cluster appear. Now you can change your cluster's name, password, credentials, and other settings.

Modifying Aurora Serverless v2 (preview) DB cluster capacity

Currently, you can modify the capacity of your Aurora Serverless v2 (preview) DB clusters with the console only.

To modify the capacity of your Aurora Serverless v2 (preview) DB cluster

1. Sign in to the preview using the AWS Management Console and open the Amazon RDS console.
2. In the navigation pane, choose **DB Clusters**.

3. Choose your Aurora Serverless v2 (preview) DB cluster from the list, and then choose **Modify** to open its configuration.

Capacity settings Info
Specify the minimum and maximum values for the capacity range. Database capacity is measured in Aurora capacity units (ACUs). 1 ACU has approximately 2 GiB of memory with corresponding CPU and networking, similar to what is used in Aurora user-provisioned instances.

Minimum Aurora capacity unit 8 GiB 4	ACUs	Maximum Aurora capacity unit 52 GiB 26	ACUs
--	------	--	------

4. For **Capacity settings**, change the minimum number of ACUs or the maximum number of ACUs that you now want for your Aurora Serverless v2 (preview) DB cluster.
5. Choose **Continue**. The **Summary of modifications** appears.

Modify DB cluster: lab-cluster-product-inventory

Summary of modifications
You are about to submit the following modifications. Only values that will change are displayed. Carefully verify your changes and click Modify Cluster.

Attribute	Current value	New value
Maximum Aurora capacity unit	32	26

Cancel Back **Modify cluster**

6. Choose **Modify cluster** to accept the summary of modifications. You can also choose **Back** to modify your changes or **Cancel** to discard your changes.

To learn more about ACUs and scaling for Aurora Serverless v2 (preview), see [Instant autoscaling \(p. 212\)](#).

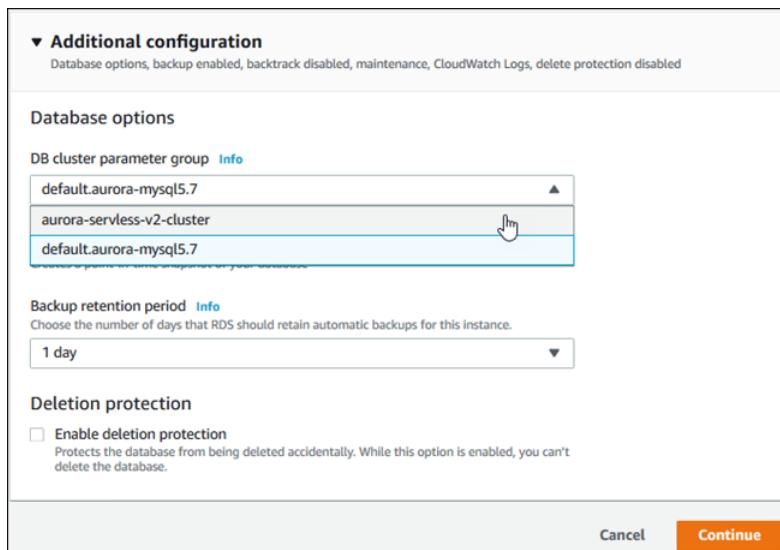
Modifying your DB cluster to use a custom DB cluster parameter group

To use a custom DB cluster parameter group after your Aurora Serverless v2 (preview) is running, modify your existing Aurora Serverless v2 (preview) DB cluster.

Before you can use the following procedure, your custom DB cluster parameter group must exist. To learn how to create a custom DB cluster parameter group, see [Parameter groups and Aurora Serverless v1 \(p. 156\)](#).

To modify your Aurora Serverless v2 (preview) DB cluster to use a custom DB cluster parameter group

1. Sign in to the preview using the AWS Management Console and open the Amazon RDS console.
2. In the navigation pane, choose **DB Clusters**.
3. Choose your Aurora Serverless v2 (preview) DB cluster from the list, and then choose **Modify**.
4. Under **Additional configuration**, choose your custom DB cluster parameter group.



5. Choose **Continue**. The **Summary of modifications** page appears.
6. Choose **Modify cluster** to accept the summary of modifications. Or choose **Back** to modify your changes or **Cancel** to discard your changes.

To learn more about creating custom DB cluster parameter groups, see [Parameter groups and Aurora Serverless v1 \(p. 156\)](#).

Deleting an Aurora Serverless v2 (preview) DB cluster

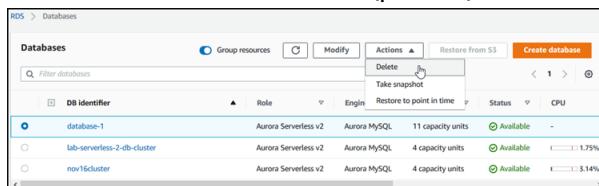
Amazon Aurora Serverless v2 with MySQL compatibility is in preview release and is subject to change. Aurora Serverless v2 (preview) is not covered by the Amazon RDS service level agreement (SLA). Don't use Aurora Serverless v2 (preview) for production databases. All resources and data will be deleted when the preview ends.

When you create your Aurora Serverless v2 (preview) DB clusters, you can choose to disable deletion protection or keep it as is. If the Aurora Serverless v2 (preview) DB cluster that you want to delete was created with deletion protection, make sure to modify your DB cluster to remove deletion protection. Otherwise, you can't delete it. To learn how do this, see [Modifying an Aurora Serverless v2 \(preview\) DB cluster \(p. 220\)](#).

Currently, you can delete an Amazon Aurora Serverless v2 (preview) DB cluster with the console only.

To delete an Aurora Serverless v2 (preview) DB cluster

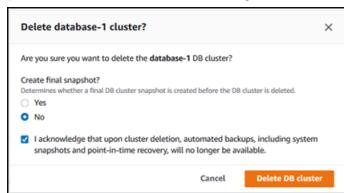
1. Sign in to the preview using the AWS Management Console and open the Amazon RDS console.
2. In the **Resources** section, choose **DB Clusters**.
3. Choose the Aurora Serverless v2 (preview) DB cluster that you want to delete.



4. For **Actions**, choose **Delete**. You're prompted to confirm that you want to delete your Aurora Serverless v2 (preview) DB cluster.
5. We recommend that you keep the preselected options:
 - Yes for **Create final snapshot?**
 - Your Aurora Serverless v2 (preview) DB cluster name plus **-final-snapshot** for **Final snapshot name**. However, you can change the name for your final snapshot in this field.



If you choose **No** for **Create final snapshot?** you can't restore your DB cluster using snapshots or point-in-time recovery.



6. Choose **Delete DB cluster**.

Aurora Serverless v2 (preview) deletes your DB cluster.

Restoring an Aurora Serverless v2 (preview) DB cluster to a point in time

Amazon Aurora Serverless v2 with MySQL compatibility is in preview release and is subject to change. Aurora Serverless v2 (preview) is not covered by the Amazon RDS service level agreement (SLA). Don't use Aurora Serverless v2 (preview) for production databases. All resources and data will be deleted when the preview ends.

You can create a new Aurora Serverless v2 (preview) DB cluster by restoring an existing DB cluster to a specific point in time. You can also use this approach to recover from a failure by recreating your Aurora Serverless v2 (preview) DB cluster from its most recent log files.

To restore your DB cluster to a specific point in time, Aurora creates a new DB cluster. It then applies all transactions from the logs of the existing DB cluster to the new cluster. Depending on the quantity and scope of transactions contained in the logs, this operation might take several hours to complete. For more information about point-in-time recovery, see [Restoring a DB cluster to a specified time \(p. 537\)](#).

Currently, you can restore your Aurora Serverless v2 (preview) DB cluster using the console only.

To restore an Aurora Serverless v2 (preview) DB cluster to a specified point in time

1. Sign in to the preview using the AWS Management Console and open the Amazon RDS console.
2. Choose **Databases**.

3. Choose the Aurora Serverless v2 (preview) DB cluster that you want to restore.
4. For **Actions**, choose **Restore to point in time**. The **Restore DB cluster** page appears.

Restore DB Cluster

You are creating a new DB instance from a source DB instance at a specified time. This new DB instance will have the default DB security group and DB parameter groups.

Restore time

Point in time to restore from

Latest restorable time
West Nov 25 2020 18:42:43 GMT-0800 (Pacific Standard Time)

Custom
Specify a custom date and time to restore from. The date must be before the latest restorable time for the DB instance.

Custom Date Start time
2020/11/22 12 : 00 : 00 UTC

Instance specifications

DB engine
Name of the database engine to be used for this instance.
Aurora MySQL

Source DB cluster
nov16cluster

DB cluster identifier
lab-restored-db-cluster-fiscal-year-end

5. Choose **Latest restorable time**. Or choose **Custom** and specify a date and time that is earlier than the latest restorable time.
6. For **Instance specifications**, keep Aurora MySQL selected for the database engine.
7. For **DB cluster identifier**, enter the name for your newly restored DB cluster.
8. In the **Capacity settings** section, choose the minimum and maximum values that you want for your restored Aurora Serverless v2 (preview) DB cluster.

Capacity settings Info

Specify the minimum and maximum values for the capacity range. Database capacity is measured in Aurora capacity units (ACUs). 1 ACU provides 2 GiB of memory, 1/4th of a vCPU and corresponding networking.

Minimum Aurora capacity unit
8 GiB
4 ACUs

Maximum Aurora capacity unit
44 GiB
22 ACUs

9. In the **Connectivity** section, accept the defaults.
10. For **Additional configuration**, choose the encryption key that you used for the DB cluster you're restoring. Choose the default key unless you used your own key when creating the DB cluster.
11. When you complete the settings on the page, choose **Restore DB Cluster**.

Using Amazon Aurora global databases

Amazon Aurora global databases span multiple AWS Regions, enabling low latency global reads and providing fast recovery from the rare outage that might affect an entire AWS Region. An Aurora global database has a primary DB cluster in one Region, and up to five secondary DB clusters in different Regions.

Topics

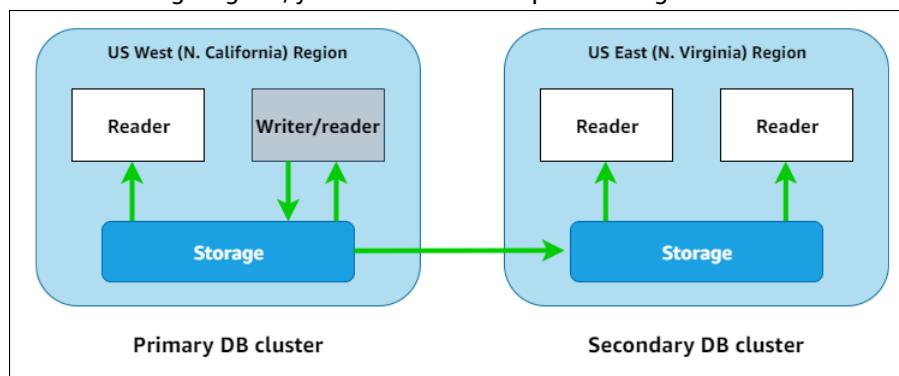
- [Overview of Amazon Aurora global databases \(p. 225\)](#)
- [Advantages of Amazon Aurora global databases \(p. 226\)](#)
- [Limitations of Amazon Aurora global databases \(p. 226\)](#)
- [Getting started with Amazon Aurora global databases \(p. 228\)](#)
- [Managing an Amazon Aurora global database \(p. 249\)](#)
- [Connecting to an Amazon Aurora global database \(p. 254\)](#)
- [Using write forwarding in an Amazon Aurora global database \(p. 255\)](#)
- [Using failover in an Amazon Aurora global database \(p. 266\)](#)
- [Monitoring an Amazon Aurora global database \(p. 276\)](#)
- [Using Amazon Aurora global databases with other AWS services \(p. 279\)](#)
- [Upgrading an Amazon Aurora global database \(p. 280\)](#)

Overview of Amazon Aurora global databases

By using an Amazon Aurora global database, you can run your globally distributed applications using a single Aurora database that spans multiple AWS Regions.

An Aurora global database consists of one *primary* AWS Region where your data is written, and up to five read-only *secondary* AWS Regions. You issue write operations directly to the primary DB cluster in the primary AWS Region. Aurora replicates data to the secondary AWS Regions using dedicated infrastructure, with latency typically under a second.

In the following diagram, you can find an example Aurora global database that spans two AWS Regions.



You can scale up each secondary cluster independently, by adding one or more Aurora Replicas (read-only Aurora DB instances) to serve read-only workloads.

Only the primary cluster performs write operations. Clients that perform write operations connect to the DB cluster endpoint of the primary DB cluster. As shown in the diagram, Aurora global database uses the cluster storage volume and not the database engine for replication. To learn more, see [Overview of Aurora storage \(p. 64\)](#).

Aurora global databases are designed for applications with a worldwide footprint. The read-only secondary DB clusters (AWS Regions) allow you to support read operations closer to application users. By using features such as write forwarding, you can also configure an Aurora MySQL-based global database so that secondary clusters send data to the primary. For more information, see [Using write forwarding in an Amazon Aurora global database \(p. 255\)](#).

An Aurora global database supports two different approaches to failover. To recover your Aurora global database after an outage in the primary Region, you use the *manual unplanned failover* process. With this process, you fail over your primary to another Region (cross-Region failover). For more information about this process, see [Recovering an Amazon Aurora global database from an unplanned outage \(p. 267\)](#).

For planned operational procedures such as maintenance, you use *managed planned failover*. With this feature, you can relocate the primary cluster of a healthy Aurora global database to one of its secondary Regions with no data loss. To learn more, see [Performing managed planned failovers for Amazon Aurora global databases \(p. 268\)](#).

Advantages of Amazon Aurora global databases

By using Aurora global databases, you can get the following advantages:

- **Global reads with local latency** – If you have offices around the world, you can use an Aurora global database to keep your main sources of information updated in the primary AWS Region. Offices in your other Regions can access the information in their own Region, with local latency.
- **Scalable secondary Aurora DB clusters** – You can scale your secondary clusters by adding more read-only instances (Aurora Replicas) to a secondary AWS Region. The secondary cluster is read-only, so it can support up to 16 read-only Aurora Replica instances rather than the usual limit of 15 for a single Aurora cluster.
- **Fast replication from primary to secondary Aurora DB clusters** – The replication performed by an Aurora global database has little performance impact on the primary DB cluster. The resources of the DB instances are fully devoted to serve application read and write workloads.
- **Recovery from Region-wide outages** – The secondary clusters allow you to make an Aurora global database available in a new primary AWS Region more quickly (lower RTO) and with less data loss (lower RPO) than traditional replication solutions.

Limitations of Amazon Aurora global databases

The following limitations currently apply to Aurora global databases:

- Aurora global databases are available in certain AWS Regions and for specific Aurora MySQL and Aurora PostgreSQL versions only. For more information, see [Aurora global databases \(p. 21\)](#).
- Aurora global databases have certain configuration requirements for supported Aurora DB instance classes, maximum number of AWS Regions, and so on. For more information, see [Configuration requirements of an Amazon Aurora global database \(p. 228\)](#).
- Managed planned failover for Aurora global databases requires one of the following Aurora database engines:
 - Aurora MySQL with MySQL 8.0 compatibility, version 3.01.0 and higher
 - Aurora MySQL with MySQL 5.7 compatibility, version 2.09.1 and higher
 - Aurora MySQL with MySQL 5.6 compatibility, version 1.23.1 and higher
 - Aurora PostgreSQL versions 13.3 and higher, 12.4 and higher, 11.9 and higher, and 10.14 and higher
- Aurora global databases currently don't support the following Aurora features:
 - Aurora multi-master clusters

- Aurora Serverless v1
- Backtracking in Aurora
- Amazon RDS Proxy
- Automatic minor version upgrade doesn't apply to Aurora MySQL and Aurora PostgreSQL clusters that are part of an Aurora global database. Note that you can specify this setting for a DB instance that is part of a global database cluster, but the setting has no effect.
- Aurora global databases currently don't support Aurora Auto Scaling for secondary DB clusters.
- You can start database activity streams on Aurora global databases running the following Aurora MySQL and Aurora PostgreSQL versions only.

Database engine	Primary AWS Region	Secondary AWS Regions
Aurora MySQL 5.7	version 2.08 and higher	version 2.08 and higher
Aurora PostgreSQL	version 13.3 and higher	version 13.3 and higher
	version 12.4 and higher	version 12.4 and higher
	version 11.7 and higher	version 11.9 and higher
	version 10.11 and higher	version 10.14 and higher

For information about database activity streams, see [Monitoring Amazon Aurora with Database Activity Streams \(p. 714\)](#).

- With an Aurora global database based on Aurora PostgreSQL, you can't perform a major version upgrade of the Aurora DB engine if the recovery point objective (RPO) feature is turned on. For information about the RPO feature, see [Managing RPOs for Aurora PostgreSQL-based global databases \(p. 272\)](#). For information about upgrading an Aurora global database, see [Upgrading an Amazon Aurora global database \(p. 280\)](#).
- You can't stop or start the Aurora DB clusters in your Aurora global database individually. To learn more, see [Stopping and starting an Amazon Aurora DB cluster \(p. 368\)](#).
- Aurora Replicas attached to the secondary Aurora DB cluster can restart under certain circumstances. If the primary AWS Region's writer DB instance restarts or fails over, Aurora Replicas in secondary Regions also restart. The secondary cluster is then unavailable until all replicas are back in sync with the primary DB cluster's writer instance. This behavior is expected, as documented in [Replication with Amazon Aurora \(p. 70\)](#). Be sure that you understand the impacts to your Aurora global database before making changes to your primary DB cluster. To learn more, see [Recovering an Amazon Aurora global database from an unplanned outage \(p. 267\)](#).
- Aurora PostgreSQL-based DB clusters running in an Aurora global database have the following limitations:
 - Cluster cache management isn't supported for Aurora PostgreSQL DB clusters that are part of Aurora global databases.
 - If the primary DB cluster of your Aurora global database is based on a replica of an Amazon RDS PostgreSQL instance, you can't create a secondary cluster. Don't attempt to create a secondary from that cluster using the AWS Management Console, the AWS CLI, or the `CreateDBCluster` API operation. Attempts to do so time out, and the secondary cluster isn't created.

We recommend that you create secondary DB clusters for your Aurora global databases by using the same version of the Aurora DB engine as the primary. For more information, see [Creating an Amazon Aurora global database \(p. 229\)](#).

Getting started with Amazon Aurora global databases

To get started with Aurora global databases, first decide which Aurora DB engine you want to use and in which AWS Regions. Only specific versions of the Aurora MySQL and Aurora PostgreSQL database engines in certain AWS Regions support Aurora global databases. For the complete list, see [Aurora global databases \(p. 21\)](#).

You can create an Aurora global database in one of the following ways:

- **Create a new Aurora global database with new Aurora DB clusters and Aurora DB instances** – You can do this by following the steps in [Creating an Amazon Aurora global database \(p. 229\)](#). After you create the primary Aurora DB cluster, you then add the secondary AWS Region by following the steps in [Adding an AWS Region to an Amazon Aurora global database \(p. 242\)](#).
- **Use an existing Aurora DB cluster that supports the Aurora global database feature and add an AWS Region to it** – You can do this only if your existing Aurora DB cluster uses a DB engine version that supports the Aurora global mode or is global-compatible. For some DB engine versions, this mode is explicit, but for others, it's not.

Check whether you can choose **Add region** for **Action** on the AWS Management Console when your Aurora DB cluster is selected. If you can, you can use that Aurora DB cluster for your Aurora global cluster. For more information, see [Adding an AWS Region to an Amazon Aurora global database \(p. 242\)](#).

Before creating an Aurora global database, we recommend that you understand all configuration requirements.

Topics

- [Configuration requirements of an Amazon Aurora global database \(p. 228\)](#)
- [Creating an Amazon Aurora global database \(p. 229\)](#)
- [Adding an AWS Region to an Amazon Aurora global database \(p. 242\)](#)
- [Creating a headless Aurora DB cluster in a secondary Region \(p. 246\)](#)
- [Using a snapshot for your Amazon Aurora global database \(p. 248\)](#)

Configuration requirements of an Amazon Aurora global database

An Aurora global database spans at least two AWS Regions. The primary AWS Region supports an Aurora DB cluster that has one writer Aurora DB instance. A secondary AWS Region runs a read-only Aurora DB cluster made up entirely of Aurora Replicas. At least one secondary AWS Region is required, but an Aurora global database can have up to five secondary AWS Regions. The table lists the maximum Aurora DB clusters, Aurora DB instances, and Aurora Replicas allowed in an Aurora global database.

Description	Primary AWS Region	Secondary AWS Regions
Aurora DB clusters	1	5 (maximum)
Writer instances	1	0
Read-only instances (Aurora replicas), per Aurora DB cluster	15 (max)	16 (total)

Description	Primary AWS Region	Secondary AWS Regions
Read-only instances (max allowed, given actual number of secondary Regions)	15 - s	s = total number of secondary AWS Regions

The Aurora DB clusters that make up an Aurora global database have the following specific requirements:

- **DB instance class requirements** – An Aurora global database requires DB instance classes that are optimized for memory-intensive applications. For information about the memory optimized DB instance classes, see [DB instance classes](#). We recommend that you use a db.r5 or higher instance class.
- **AWS Region requirements** – An Aurora global database needs a primary Aurora DB cluster in one AWS Region, and at least one secondary Aurora DB cluster in a different Region. You can create up to five secondary (read-only) Aurora DB clusters, and each must be in a different Region. In other words, no two Aurora DB clusters in an Aurora global database can be in the same AWS Region.
- **Naming requirements** – The names you choose for each of your Aurora DB clusters must be unique, across all AWS Regions. You can't use the same name for different Aurora DB clusters even though they're in different Regions.

Before you can follow the procedures in this section, you need an AWS account. Complete the setup tasks for working with Amazon Aurora. For more information, see [Setting up your environment for Amazon Aurora \(p. 84\)](#). You also need to complete other preliminary steps for creating any Aurora DB cluster. To learn more, see [Creating an Amazon Aurora DB cluster \(p. 125\)](#).

Creating an Amazon Aurora global database

In some cases, you might have an existing Aurora provisioned DB cluster running an Aurora database engine that's global-compatible. If so, you can add another AWS Region to it to create your Aurora global database. To do so, see [Adding an AWS Region to an Amazon Aurora global database \(p. 242\)](#).

To create an Aurora global database by using the AWS Management Console, the AWS CLI, or the RDS API, use the following steps.

Console

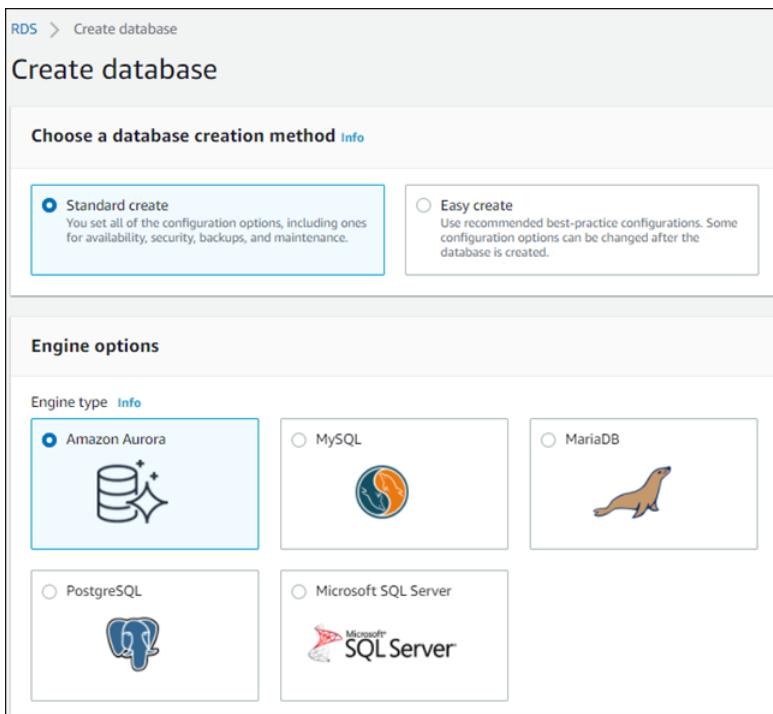
The steps for creating an Aurora global database begin by signing in to an AWS Region that supports the Aurora global database feature. For a complete list, see [Aurora global databases \(p. 21\)](#).

One of the following steps is choosing a virtual private cloud (VPC) based on Amazon VPC for your Aurora DB cluster. To use your own VPC, we recommend that you create it in advance so it's available for you to choose. At the same time, create any related subnets, and as needed a subnet group and security group. To learn how, see [How to create a VPC for use with Amazon Aurora](#).

For general information about creating an Aurora DB cluster, see [Creating an Amazon Aurora DB cluster \(p. 125\)](#).

To create an Aurora global database

1. Sign in to the AWS Management Console and open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. Choose **Create database**. On the **Create database** page, do the following:
 - For the database creation method, choose **Standard create**. (Don't choose Easy create.)
 - For **Engine type** in the **Engine options** section, choose **Amazon Aurora**.



Then choose **Amazon Aurora with MySQL compatibility** or **Amazon Aurora with PostgreSQL compatibility**, and continue creating your Aurora global database by using the steps from the following procedures.

Topics

- [Creating a global database using Aurora MySQL \(p. 230\)](#)
- [Creating a global database using Aurora PostgreSQL \(p. 234\)](#)

[Creating a global database using Aurora MySQL](#)

The following steps apply to all versions of Aurora MySQL except for Aurora MySQL 5.6.10a. To use Aurora MySQL 5.6.10a for your Aurora global database, see [Using Aurora MySQL 5.6.10a for an Aurora global database \(p. 233\)](#).

To create an Aurora global database using Aurora MySQL

Complete the **Create database** page.

1. For **Engine options**, choose the following:
 - a. For **Edition**, choose **Amazon Aurora with MySQL compatibility**.
 - b. For **Capacity type**, choose **Provisioned**.
 - c. Leave **Replication features** set to the default (single-master replication).
 - d. Turn on **Show versions that support the global database feature**.
 - e. For **Version**, choose the version of Aurora MySQL that you want to use for your Aurora global database.

Edition

- Amazon Aurora with MySQL compatibility
- Amazon Aurora with PostgreSQL compatibility

Capacity type [Info](#)

- Provisioned**
You provision and manage the server instance sizes.
- Serverless**
You specify the minimum and maximum amount of resources needed, and Aurora scales the capacity based on database load. This is a good option for intermittent or unpredictable workloads.

▶ Replication features [Info](#)
Single-master replication is currently selected

Engine version [Info](#)
View the engine versions that support the following database features.

- Show versions that support the global database feature
- Show versions that support the parallel query feature

Version

Aurora (MySQL 5.7) 2.07.2

To see more versions, modify the capacity types. [Info](#)

2. For **Templates**, choose **Production**. Or you can choose Dev/Test if appropriate for your use case. Don't use Dev/Test in production environments.
3. For **Settings**, do the following:
 - a. Enter a meaningful name for the DB cluster identifier. When you finish creating the Aurora global database, this name identifies the primary DB cluster.
 - b. Enter your own password for the **admin** user account for the DB instance, or have Aurora generate one for you. If you choose to autogenerated a password, you get an option to copy the password.

Settings

DB cluster identifier [Info](#)
Type a name for your DB cluster. The name must be unique across all DB clusters owned by your AWS account in the current AWS Region.

The DB cluster identifier is case-insensitive, but is stored as all lowercase (as in "mydbcluster"). Constraints: 1 to 60 alphanumeric characters or hyphens. First character must be a letter. Can't contain two consecutive hyphens. Can't end with a hyphen.

▼ Credentials Settings

Master username [Info](#)
Type a login ID for the master user of your DB instance.

1 to 16 alphanumeric characters. First character must be a letter

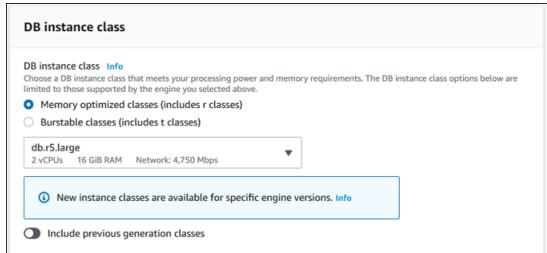
Auto generate a password
Amazon RDS can generate a password for you, or you can specify your own password

Master password [Info](#)

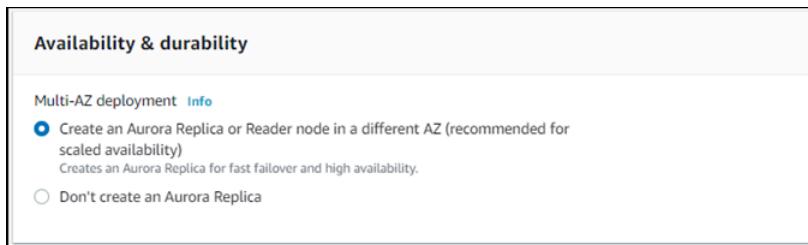
Constraints: At least 8 printable ASCII characters. Can't contain any of the following: / (slash), '(single quote), "(double quote) and @ (at sign).

Confirm password [Info](#)

4. For **DB instance class**, choose **db.r5.large** or another memory optimized DB instance class. We recommend that you use a db.r5 or higher instance class.



- For **Availability & durability**, we recommend that you choose to have Aurora create an Aurora Replica in a different Availability Zone (AZ) for you. If you don't create an Aurora Replica now, you need to do it later.



- For **Connectivity**, choose the virtual private cloud (VPC) based on Amazon VPC that defines the virtual networking environment for this DB instance. You can choose the defaults to simplify this task.
- Complete the **Database authentication** settings. To simplify the process, you can choose **Password authentication** now and set up AWS Identity and Access Management (IAM) later.
- For **Additional configuration**, do the following:
 - Enter a name for **Initial database name** to create the primary Aurora DB instance for this cluster. This is the writer node for the Aurora primary DB cluster.

Leave the defaults selected for the DB cluster parameter group and DB parameter group, unless you have your own custom parameter groups that you want to use.

 - Clear the **Enable backtrack** check box if it's selected. Aurora global databases don't support backtracking. Otherwise, accept the other default settings for **Additional configuration**.
- Choose **Create database**.

It can take several minutes for Aurora to complete the process of creating the Aurora DB instance, its Aurora Replica, and the Aurora DB cluster. You can tell when the Aurora DB cluster is ready to use as the primary DB cluster in an Aurora global database by its status. When that's so, its status and that of the writer and replica node is **Available**, as shown following.

Databases						
	DB identifier	Role	Engine	Region & AZ	Size	Status
<input type="checkbox"/> Group resources <input type="button" value="Modify"/> <input type="button" value="Actions"/> <input type="button" value="Restore from S3"/> <input type="button" value="Create database"/>						
<input type="text" value="Filter databases"/>						
	lab-demo-db-cluster	Regional	Aurora PostgreSQL	us-west-1	1 instance	Available
	lab-west-db-cluster	Regional	Aurora MySQL	us-west-1	2 instances	Available
	lab-west-db-cluster-instance-1	Writer	Aurora MySQL	us-west-1b	db.r4.large	Available
	lab-west-db-cluster-instance-1-us-west-1c	Reader	Aurora MySQL	us-west-1c	db.r4.large	Available

When your primary DB cluster is available, create the Aurora global database by adding a secondary cluster to it. To do this, follow the steps in [Adding an AWS Region to an Amazon Aurora global database \(p. 242\)](#).

Using Aurora MySQL 5.6.10a for an Aurora global database

The following steps apply to the 5.6.10a version of Aurora MySQL only. For other versions of Aurora MySQL, see [Creating a global database using Aurora MySQL \(p. 230\)](#).

To create an Aurora global database using Aurora MySQL 5.6.10a

Complete the **Create database** page.

1. For **Engine options**, choose the following:
 - a. For **Edition**, choose **Amazon Aurora with MySQL compatibility**.
 - b. For **Capacity type**, choose **Provisioned**.
 - c. Leave **Replication features** set to the default (single-master replication).
 - d. Turn on **Show versions that support the global database feature**.
 - e. For **Version**, choose **Aurora (MySQL 5.6) global_10a**.
2. For **Templates**, choose **Production**.
3. For **Global database settings**, do the following:
 - a. For **Global database identifier**, enter a meaningful name.
 - b. For **Credentials Settings**, enter your own password for the `postgres` user account for the DB instance, or have Aurora generate one for you. If you choose Auto generate a password, you get an option to copy the password.

Global database settings

Settings

Global database identifier [Info](#)
Enter a name for your global database. The name must be unique across all global databases in your AWS account.
lab-ams5610a-global-database

The global database identifier is case-insensitive, but is stored as all lowercase (as in "mydbinstance"). Constraints: 1 to 60 alphanumeric characters or hyphens. First character must be a letter. Can't contain two consecutive hyphens. Can't end with a hyphen.

▼ Credentials Settings

Master username [Info](#)
Type a login ID for the master user of your DB instance.
admin

1 to 16 alphanumeric characters. First character must be a letter

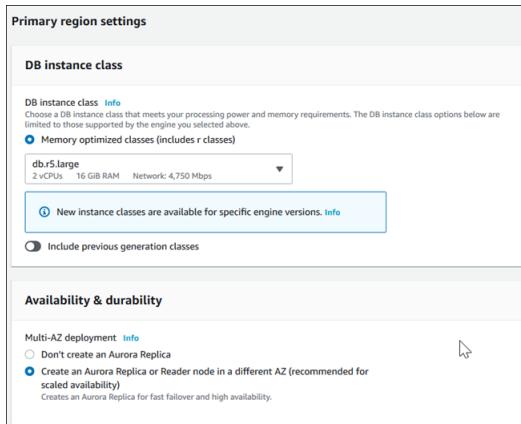
Auto generate a password
Amazon RDS can generate a password for you, or you can specify your own password

Master password [Info](#)
.....

Constraints: At least 8 printable ASCII characters. Can't contain any of the following: / (slash), '(single quote), "(double quote) and @ (at sign).

Confirm password [Info](#)
.....

4. For **Encryption**, enable or disable encryption as needed.
5. The remaining sections of the **Create database** page configure the **Primary region settings**. Complete these as follows:
 - a. For **DB instance class**, choose `db.r5.large` or another memory optimized DB instance class. We recommend that you use a `db.r5` or higher instance class.



- b. For **Availability & durability**, we recommend that you choose to have Aurora create an Aurora Replica in a different AZ for you. If you don't create an Aurora Replica now, you need to do it later.
- c. For **Connectivity**, choose the virtual private cloud (VPC) based on Amazon VPC that defines the virtual networking environment for this DB instance. You can choose the defaults to simplify this task.
- d. For **Encryption key**, choose the key to use. If you didn't choose **Encryption** earlier, disregard this option.
- e. Complete the **Database authentication** settings. To simplify the process, you can choose **Password authentication** now and set up IAM later.
- f. For **Additional configuration**, do the following:
 - i. For **DB instance identifier**, enter a name for the database instance, or use the default provided. This is the writer instance for the Aurora primary DB cluster for this Aurora global database.
 - ii. For **DB cluster identifier**, enter a meaningful name or accept the default provided.
 - iii. Leave the defaults selected for the DB cluster parameter group and DB parameter group, unless you have your own custom parameter groups that you want to use.
 - iv. You can accept all other default settings for **Additional configuration**.
- g. Choose **Create database**.

It can take several minutes for Aurora to complete the process of creating the Aurora DB instance, its Aurora Replica, and the Aurora DB cluster. You can tell when the Aurora DB cluster is ready to use as the primary DB cluster in an Aurora global database by its status. When that's so, its status and that of the writer and replica node is **Available**, as shown following.

		Global	Aurora MySQL	1 region	1 cluster
<input type="radio"/>	lab-ams5610a-global-database	Global	Aurora MySQL	1 region	1 cluster
<input type="radio"/>	lab-ams5610a-global-database-cluster-1	Primary	Aurora MySQL	us-west-1	2 instances
<input type="radio"/>	lab-ams5610a-global-database-instance-1	Reader	Aurora MySQL	us-west-1b	db.r4.large
<input type="radio"/>	lab-ams5610a-global-database-instance-1-us-west-1c	Reader	Aurora MySQL	us-west-1c	db.r4.large

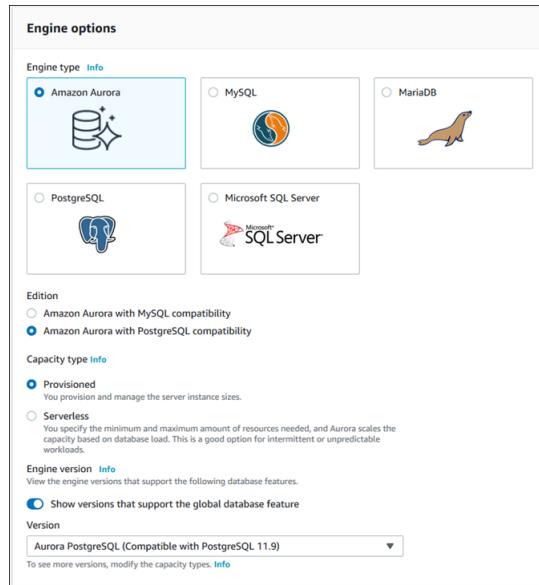
This Aurora global database still needs a secondary Aurora DB cluster. You can add that now, by following the steps in [Adding an AWS Region to an Amazon Aurora global database \(p. 242\)](#).

Creating a global database using Aurora PostgreSQL

To create an Aurora global database using Aurora PostgreSQL

Complete the **Create database** page.

1. For **Engine options**, choose the following:
 - a. For **Edition**, choose **Amazon Aurora with PostgreSQL compatibility**.
 - b. For **Capacity type**, choose **Provisioned**.
 - c. Turn on **Show versions that support the global database feature**.
 - d. For **Version**, choose the version of Aurora PostgreSQL that you want to use for your Aurora global database.



2. For **Templates**, choose **Production**. Or you can choose Dev/Test if appropriate. Don't use Dev/Test in production environments.
3. For **Settings**, do the following:
 - a. Enter a meaningful name for the DB cluster identifier. When you finish creating the Aurora global database, this name identifies the primary DB cluster.
 - b. Enter your own password for the default admin account for the DB cluster, or have Aurora generate one for you. If you choose Auto generate a password, you get an option to copy the password.

Settings

DB cluster identifier [Info](#)
Type a name for your DB cluster. The name must be unique across all DB clusters owned by your AWS account in the current AWS Region.

The DB cluster identifier is case-insensitive, but is stored as all lowercase (as in "mydbcluster"). Constraints: 1 to 60 alphanumeric characters or hyphens. First character must be a letter. Can't contain two consecutive hyphens. Can't end with a hyphen.

Credentials Settings

Master username [Info](#)
Type a login ID for the master user of your DB instance.

1 to 16 alphanumeric characters. First character must be a letter

Auto generate a password
Amazon RDS can generate a password for you, or you can specify your own password

Master password [Info](#)

Constraints: At least 8 printable ASCII characters. Can't contain any of the following: / (slash), {single quote), "(double quote) and @ (at sign).

Confirm password [Info](#)

- For **DB instance class**, choose `db.r5.large` or another memory optimized DB instance class. We recommend that you use a `db.r5` or higher instance class.

DB instance class

DB instance class [Info](#)
Choose a DB instance class that meets your processing power and memory requirements. The DB instance class options below are limited to those supported by the engine you selected above.

Memory optimized classes (includes r classes)
 Burstable classes (includes t classes)

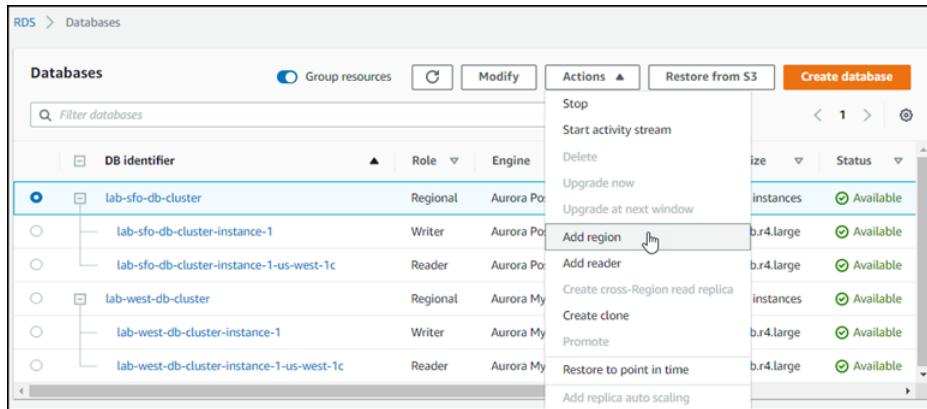
db.r5.large
2 vCPUs - 16 GiB RAM - Network: 4,750 Mbps

[New instance classes are available for specific engine versions. Info](#)

Include previous generation classes

- For **Availability & durability**, we recommend that you choose to have Aurora create an Aurora Replica in a different AZ for you. If you don't create an Aurora Replica now, you need to do it later.
- For **Connectivity**, choose the virtual private cloud (VPC) based on Amazon VPC that defines the virtual networking environment for this DB instance. You can choose the defaults to simplify this task.
- Complete the **Database authentication** settings. To simplify the process, you can choose **Password authentication** now and set up IAM or password and Kerberos authentication later.
- For **Additional configuration**, do the following:
 - Enter a name for **Initial database name** to create the primary Aurora DB instance for this cluster. This is the writer node for the Aurora primary DB cluster.
Leave the defaults selected for the DB cluster parameter group and DB parameter group, unless you have your own custom parameter groups that you want to use.
 - Accept all other default settings for **Additional configuration**, such as Monitoring, Log exports, and so on.
- Choose **Create database**.

It can take several minutes for Aurora to complete the process of creating the Aurora DB instance, its Aurora Replica, and the Aurora DB cluster. When the cluster is ready to use, the Aurora DB cluster and its writer and replica nodes display **Available** status. This becomes the primary DB cluster of your Aurora global database, after you add a secondary.



When your primary DB cluster is available, create one or more secondary clusters by following the steps in [Adding an AWS Region to an Amazon Aurora global database \(p. 242\)](#).

AWS CLI

The AWS CLI commands in the procedures following accomplish the following tasks:

1. Create an Aurora global database, giving it a name and specifying the Aurora database engine type that you plan to use.
2. Create an Aurora DB cluster for the Aurora global database.
3. Create the Aurora DB instance for the cluster.
4. Create an Aurora DB instance for the Aurora DB cluster.
5. Create a second DB instance for Aurora DB cluster. This is a reader to complete the Aurora DB cluster.
6. Create a second Aurora DB cluster in another Region and then add it to your Aurora global database, by following the steps in [Adding an AWS Region to an Amazon Aurora global database \(p. 242\)](#).

Follow the procedure for your Aurora database engine.

CLI examples

- [Creating a global database using Aurora MySQL \(p. 165\)](#)
- [Creating a global database using Aurora PostgreSQL \(p. 166\)](#)

[Creating a global database using Aurora MySQL](#)

To create an Aurora global database using Aurora MySQL

1. Use the `create-global-cluster` CLI command, passing the name of the AWS Region, Aurora database engine and version. Choose your parameters from those shown in the table for the version of Aurora MySQL that you want to use.

Other options for Aurora MySQL depend on the version of the Aurora MySQL database engine, as shown in the following table.

Parameter	Aurora MySQL version 1	Aurora MySQL version 2 and 3
<code>--engine</code>	<code>aurora</code>	<code>aurora-mysql</code>
<code>--engine-mode</code>	<code>global</code>	-

Parameter	Aurora MySQL version 1	Aurora MySQL version 2 and 3
--engine-version	5.6.10a, 5.6.mysql_aurora.1.22.0, 5.7.mysql_aurora.2.07.1, 5.6.mysql_aurora.1.22.1, 5.7.mysql_aurora.2.07.2, 5.6.mysql_aurora.1.22.2, 5.7.mysql_aurora.2.07.3, 5.6.mysql_aurora.1.22.3, 5.7.mysql_aurora.2.08.0, 5.6.mysql_aurora.1.23.0, 5.7.mysql_aurora.2.08.1, 5.6.mysql_aurora.1.23.1, 5.7.mysql_aurora.2.08.1, and later versions	5.7.mysql_aurora.2.07.0, 5.7.mysql_aurora.2.08.3, 5.7.mysql_aurora.2.09.0, 5.7.mysql_aurora.2.08.1, and later versions; 8.0.mysql_aurora.3.01.0 and later versions

For Linux, macOS, or Unix:

```
aws rds create-global-cluster --region primary_region \
    --global-cluster-identifier global_database_id \
    --engine aurora \
    --engine-version version # optional
```

For Windows:

```
aws rds create-global-cluster ^
    --global-cluster-identifier global_database_id ^
    --engine aurora ^
    --engine-version version # optional
```

This creates an "empty" Aurora global database, with just a name (identifier) and Aurora database engine. It can take a few minutes for the Aurora global database to be available. Before going to the next step, use the [describe-global-clusters](#) CLI command to see if it's available.

```
aws rds describe-global-clusters --region primary_region --global-cluster-
    identifier global_database_id
```

When the Aurora global database is available, you can create its primary Aurora DB cluster.

2. To create a primary Aurora DB cluster, use the [create-db-cluster](#) CLI command. Include the name of your Aurora global database by using the `--global-cluster-identifier`.

For Linux, macOS, or Unix:

```
aws rds create-db-cluster \
    --region primary_region \
    --db-cluster-identifier db_cluster_id \
    --master-username userid \
    --master-user-password password \
    --engine { aurora | aurora-mysql } \
    --engine-mode global # Required for --engine-version 5.6.10a only \
    --engine-version version \
    --global-cluster-identifier global_database_id
```

For Windows:

```
aws rds create-db-cluster ^
--region primary_region ^
--db-cluster-identifier db_cluster_id ^
--master-username userid ^
--master-user-password password ^
--engine { aurora | aurora-mysql } ^
--engine-mode global # Required for --engine-version 5.6.10a only ^
--engine-version version ^
--global-cluster-identifier global_database_id
```

Other options for Aurora MySQL depend on the version of the Aurora MySQL database engine.

Use the [describe-db-clusters](#) AWS CLI command to confirm that the Aurora DB cluster is ready. To single out a specific Aurora DB cluster, use `--db-cluster-identifier` parameter. Or you can leave out the Aurora DB cluster name in the command to get details about all your Aurora DB clusters in the given Region.

```
aws rds describe-db-clusters --region primary_region --db-cluster-
identifier db_cluster_id
```

When the response shows "Status": "available" for the cluster, it's ready to use.

3. Create the DB instance for your primary Aurora DB cluster. To do so, use the [create-db-instance](#) CLI command. Give the command your Aurora DB cluster's name, and specify the configuration details for the instance. You don't need to pass the `--master-username` and `--master-user-password` parameters in the command, because it gets those from the Aurora DB cluster.

For the `--db-instance-class`, you can use only those from the memory optimized classes, such as `db.r5.large`. We recommend that you use a `db.r5` or higher instance class. For information about these classes, see [DB instance classes](#).

For Linux, macOS, or Unix:

```
aws rds create-db-instance \
--db-cluster-identifier db_cluster_id \
--db-instance-class instance_class \
--db-instance-identifier db_instance_id \
--engine { aurora | aurora-mysql } \
--engine-mode global # Required for --engine-version 5.6.10a only \
--engine-version version \
--region primary_region
```

For Windows:

```
aws rds create-db-instance ^
--db-cluster-identifier db_cluster_id ^
--db-instance-class instance_class ^
--db-instance-identifier db_instance_id ^
--engine { aurora | aurora-mysql } ^
--engine-mode global # Required for --engine-version 5.6.10a only ^
--engine-version version ^
--region primary_region
```

The `create-db-instance` operation might take some time to complete. Check the status to see if the Aurora DB instance is available before continuing.

```
aws rds describe-db-clusters --db-cluster-identifier sample_secondary_db_cluster
```

When the command returns a status of "available," you can create another Aurora DB instance for your primary DB cluster. This is the reader instance (the Aurora Replica) for the Aurora DB cluster.

4. To create another Aurora DB instance for the cluster, use the [create-db-instance](#) CLI command.

For Linux, macOS, or Unix:

```
aws rds create-db-instance \
--db-cluster-identifier sample_secondary_db_cluster \
--db-instance-class instance_class \
--db-instance-identifier sample_replica_db \
--engine aurora
```

For Windows:

```
aws rds create-db-instance ^
--db-cluster-identifier sample_secondary_db_cluster ^
--db-instance-class instance_class ^
--db-instance-identifier sample_replica_db ^
--engine aurora
```

When the DB instance is available, replication begins from the writer node to the replica. Before continuing, check that the DB instance is available with the [describe-db-instances](#) CLI command.

At this point, you have an Aurora global database with its primary Aurora DB cluster containing a writer DB instance and an Aurora Replica. You can now add a read-only Aurora DB cluster in a different Region to complete your Aurora global database. To do so, follow the steps in [Adding an AWS Region to an Amazon Aurora global database \(p. 242\)](#).

Creating a global database using Aurora PostgreSQL

When you create Aurora objects for an Aurora global database by using the following commands, it can take a few minutes for each to become available. We recommend that after completing any given command, you check the specific Aurora object's status to make sure that the status is available.

To do so, use the [describe-global-clusters](#) CLI command.

```
aws rds describe-global-clusters --region primary_region
--global-cluster-identifier global_database_id
```

To create an Aurora global database using Aurora PostgreSQL

1. Use the [create-global-cluster](#) CLI command.

For Linux, macOS, or Unix:

```
aws rds create-global-cluster --region primary_region \
--global-cluster-identifier global_database_id \
--engine aurora-postgresql \
--engine-version version # optional
```

For Windows:

```
aws rds create-global-cluster ^
--global-cluster-identifier global_database_id ^
--engine aurora-postgresql ^
--engine-version version # optional
```

When the Aurora global database is available, you can create its primary Aurora DB cluster.

2. To create a primary Aurora DB cluster, use the [create-db-cluster](#) CLI command. Include the name of your Aurora global database by using the `--global-cluster-identifier`.

For Linux, macOS, or Unix:

```
aws rds create-db-cluster \
  --region primary_region \
  --db-cluster-identifier db_cluster_id \
  --master-username userid \
  --master-user-password password \
  --engine aurora-postgresql \
  --engine-version version \
  --global-cluster-identifier global_database_id
```

For Windows:

```
aws rds create-db-cluster ^
  --region primary_region ^
  --db-cluster-identifier db_cluster_id ^
  --master-username userid ^
  --master-user-password password ^
  --engine aurora-postgresql ^
  --engine-version version ^
  --global-cluster-identifier global_database_id
```

Check that the Aurora DB cluster is ready. When the response from the following command shows "Status": "available" for the Aurora DB cluster, you can continue.

```
aws rds describe-db-clusters --region primary_region --db-cluster-
  identifier db_cluster_id
```

3. Create the DB instance for your primary Aurora DB cluster. To do so, use the [create-db-instance](#) CLI command.

- Pass the name of your Aurora DB cluster with the `--db-instance-identifier` parameter.

You don't need to pass the `--master-username` and `--master-user-password` parameters in the command, because it gets those from the Aurora DB cluster.

For the `--db-instance-class`, you can use only those from the memory optimized classes, such as `db.r5.large`. We recommend that you use a `db.r5` or higher instance class. For information about these classes, see [DB instance classes](#).

For Linux, macOS, or Unix:

```
aws rds create-db-instance \
  --db-cluster-identifier db_cluster_id \
  --db-instance-class instance_class \
  --db-instance-identifier db_instance_id \
  --engine aurora-postgresql \
  --engine-version version \
  --region primary_region
```

For Windows:

```
aws rds create-db-instance ^
```

```
--db-cluster-identifier db_cluster_id ^
--db-instance-class instance_class ^
--db-instance-identifier db_instance_id ^
--engine aurora-postgresql ^
--engine-version version ^
--region primary_region
```

4. Check the status of the Aurora DB instance before continuing.

```
aws rds describe-db-clusters --db-cluster-identifier sample_secondary_db_cluster
```

If the response shows that Aurora DB instance status is "available," you can create another Aurora DB instance for your primary DB cluster.

5. To create an Aurora Replica for Aurora DB cluster, use the [create-db-instance](#) CLI command.

For Linux, macOS, or Unix:

```
aws rds create-db-instance \
--db-cluster-identifier sample_secondary_db_cluster \
--db-instance-class instance_class \
--db-instance-identifier sample_replica_db \
--engine aurora-postgresql
```

For Windows:

```
aws rds create-db-instance ^
--db-cluster-identifier sample_secondary_db_cluster ^
--db-instance-class instance_class ^
--db-instance-identifier sample_replica_db ^
--engine aurora-postgresql
```

When the DB instance is available, replication begins from the writer node to the replica. Before continuing, check that the DB instance is available with the [describe-db-instances](#) CLI command.

Your Aurora global database exists, but it has only its primary Region with an Aurora DB cluster made up of a writer DB instance and an Aurora Replica. You can now add a read-only Aurora DB cluster in a different Region to complete your Aurora global database. To do so, follow the steps in [Adding an AWS Region to an Amazon Aurora global database \(p. 242\)](#).

RDS API

To create an Aurora global database with the RDS API, run the [CreateGlobalCluster](#) operation.

Adding an AWS Region to an Amazon Aurora global database

An Aurora global database needs at least one secondary Aurora DB cluster in a different AWS Region than the primary Aurora DB cluster. You can attach up to five secondary DB clusters to your Aurora global database. For each secondary DB cluster that you add to your Aurora global database, reduce the number of Aurora Replicas allowed to the primary DB cluster by one.

For example, if your Aurora global database has 5 secondary Regions, your primary DB cluster can have only 10 (rather than 15) Aurora Replicas. For more information, see [Configuration requirements of an Amazon Aurora global database \(p. 228\)](#).

The number of Aurora Replicas (reader instances) in the primary DB cluster determines the number of secondary DB clusters you can add. The total number of reader instances in the primary DB cluster

plus the number of secondary clusters can't exceed 15. For example, if you have 14 reader instances in the primary DB cluster and 1 secondary cluster, you can't add another secondary cluster to the global database.

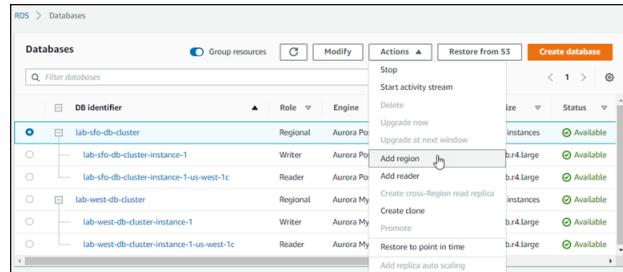
Note

For Aurora MySQL version 3, when you create a secondary cluster, make sure that the value of `lower_case_table_names` matches the value in the primary cluster. This setting is a database parameter that affects how the server handles identifier case sensitivity. For more information about database parameters, see [Working with DB parameter groups and DB cluster parameter groups \(p. 339\)](#).

Console

To add an AWS Region to an Aurora global database

1. Sign in to the AWS Management Console and open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. In the navigation pane of the AWS Management Console, choose **Databases**.
3. Choose the Aurora global database that needs a secondary Aurora DB cluster. Ensure that the primary Aurora DB cluster is Available.
4. For **Actions**, choose **Add region**.



5. On the **Add a region** page, choose the secondary AWS Region.

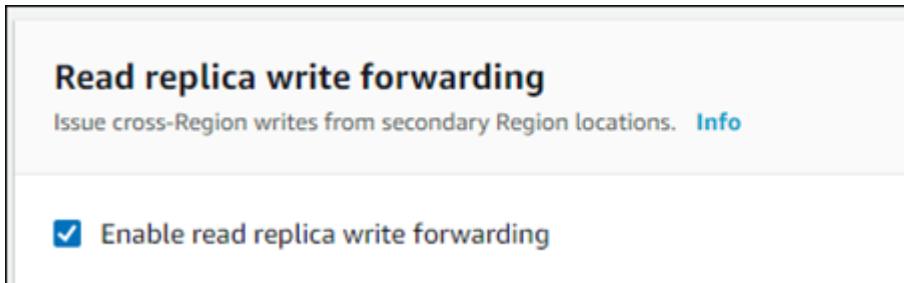
You can't choose an AWS Region that already has a secondary Aurora DB cluster for the same Aurora global database. Also, it can't be the same Region as the primary Aurora DB cluster.

The screenshot shows the 'Add a region' configuration page. It includes sections for 'Global database settings' and 'Region'.

- Global database settings:**
 - Global database identifier:** The input field contains 'lab-east-west-global'.
 - A note below states: 'The global database identifier is case-insensitive, but is stored as all lowercase (as in "mydbinstance"). Constraints: 1 to 60 alphanumeric characters or hyphens. First character must be a letter. Can't contain two consecutive hyphens. Can't end with a hyphen.'
- Region:**
 - Secondary region:** The dropdown menu shows 'US East (N. Virginia)'.

6. Complete the remaining fields for the secondary Aurora cluster in the new AWS Region. These are the same configuration options as for any Aurora DB cluster instance, except for the following option for Aurora MySQL-based Aurora global databases only:

- Enable read replica write forwarding – This optional setting lets your Aurora global database's secondary DB clusters forward write operations to the primary cluster. For more information, see [Using write forwarding in an Amazon Aurora global database \(p. 255\)](#).



7. Add region.

After you finish adding the Region to your Aurora global database, you can see it in the list of **Databases** in the AWS Management Console as shown in the screenshot.

Databases		Group resources		Modify	Actions ▾	Restore from S3
		Filter databases				
	DB identifier	Role	Engine	Region & AZ	Size	▼
○	lab-east-west-global	Global	Aurora PostgreSQL	2 regions	2 clusters	
○	lab-sfo-db-cluster	Primary	Aurora PostgreSQL	us-west-1	2 instances	
○	lab-sfo-db-cluster-instance-1	Writer	Aurora PostgreSQL	us-west-1b	db.r4.large	
○	lab-sfo-db-cluster-instance-1-us-west-1c	Reader	Aurora PostgreSQL	us-west-1c	db.r4.large	
○	lab-east-coast-db-cluster	Secondary	Aurora PostgreSQL	us-east-1	2 instances	
○	lab-east-coast-db-instance	Reader	Aurora PostgreSQL	us-east-1b	db.r4.large	
○	lab-east-coast-db-instance-us-east-1c	Reader	Aurora PostgreSQL	us-east-1c	db.r4.large	

AWS CLI

To add a secondary AWS Region to an Aurora global database

1. Use the `create-db-cluster` CLI command with the name (`--global-cluster-identifier`) of your Aurora global database. For other parameters, do the following:
 2. For `--region`, choose a different AWS Region than that of your Aurora primary Region.
 3. Do one of the following:
 - For an Aurora global database based on Aurora MySQL 5.6.10a only, use the following parameters:
 - `--engine - aurora`
 - `--engine-mode - global`
 - `--engine-version - 5.6.10a`

- For an Aurora global database based on other Aurora DB engines, choose specific values for the `--engine` and `--engine-version` parameters. These values are the same as those for the primary Aurora DB cluster in your Aurora global database.

The following table displays current options.

Parameter	Aurora MySQL 5.6	Aurora MySQL 5.7	Aurora PostgreSQL
<code>--engine</code>	aurora	aurora-mysql	aurora-postgresql
<code>--engine-version</code>	5.6.mysql_aurora.1.22.0, 5.6.mysql_aurora.1.22.1, 5.6.mysql_aurora.1.22.2, 5.6.mysql_aurora.1.22.3, 5.6.mysql_aurora.1.23.0, 5.6.mysql_aurora.1.23.1	5.7.mysql_aurora.2.07.0, 5.7.mysql_aurora.2.07.1, 5.7.mysql_aurora.2.07.2, 5.7.mysql_aurora.2.07.3, 5.7.mysql_aurora.2.08.0, 5.7.mysql_aurora.2.08.1, 5.7.mysql_aurora.2.08.2, 5.7.mysql_aurora.2.08.3, 5.7.mysql_aurora.2.09.0 (and later)	10.11 (and later), 11.7 (and later), 12.4 (and later)

- For an encrypted cluster, specify your primary AWS Region as the `--source-region` for encryption.

The following example creates a new Aurora DB cluster and attaches it to an Aurora global database as a read-only secondary Aurora DB cluster. In the last step, an Aurora DB instance is added to the new Aurora DB cluster.

For Linux, macOS, or Unix:

```
aws rds --region secondary_region \
create-db-cluster \
--db-cluster-identifier secondary_cluster_id \
--global-cluster-identifier global_database_id \
--engine { aurora | aurora-mysql | aurora-postgresql } \
--engine-version version

aws rds --region secondary_region \
create-db-instance \
--db-instance-class instance_class \
--db-cluster-identifier secondary_cluster_id \
--db-instance-identifier db_instance_id \
--engine { aurora | aurora-mysql | aurora-postgresql }
```

For Windows:

```
aws rds --region secondary_region ^
create-db-cluster ^
--db-cluster-identifier secondary_cluster_id ^
--global-cluster-identifier global_database_id_id ^
--engine { aurora | aurora-mysql | aurora-postgresql } ^
--engine-version version

aws rds --region secondary_region ^
create-db-instance ^
--db-instance-class instance_class ^
--db-cluster-identifier secondary_cluster_id ^
--db-instance-identifier db_instance_id ^
--engine { aurora | aurora-mysql | aurora-postgresql }
```

RDS API

To add a new AWS Region to an Aurora global database with the RDS API, run the [CreateDBCluster](#) operation. Specify the identifier of the existing global database using the `GlobalClusterIdentifier` parameter.

Creating a headless Aurora DB cluster in a secondary Region

Although an Aurora global database requires at least one secondary Aurora DB cluster in a different AWS Region than the primary, you can use a *headless* configuration for the secondary cluster. A headless secondary Aurora DB cluster is one without a DB instance. This type of configuration can lower expenses for an Aurora global database. In an Aurora DB cluster, compute and storage are decoupled. Without the DB instance, you're not charged for compute, only for storage. If it's set up correctly, a headless secondary's storage volume is kept in-sync with the primary Aurora DB cluster.

You add the secondary cluster as you normally do when creating an Aurora global database. However, after the primary Aurora DB cluster begins replication to the secondary, you delete the Aurora read-only DB instance from the secondary Aurora DB cluster. This secondary cluster is now considered "headless" because it no longer has a DB instance. Yet, the storage volume is kept in sync with the primary Aurora DB cluster.

Warning

With Aurora PostgreSQL, to create a headless cluster in a secondary AWS Region, use the AWS CLI or RDS API to add the secondary AWS Region. Skip the step to create the reader DB instance for the secondary cluster. Currently, creating a headless cluster isn't supported in the RDS Console. For the CLI and API procedures to use, see [Adding an AWS Region to an Amazon Aurora global database \(p. 242\)](#).

Creating a reader DB instance in a secondary Region and subsequently deleting it could lead to an Aurora PostgreSQL vacuum issue on the primary Region's writer DB instance. If you encounter this issue, restart the primary Region's writer DB instance after you delete the secondary Region's reader DB instance.

To add a headless secondary Aurora DB cluster to your Aurora global database

1. Sign in to the AWS Management Console and open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. In the navigation pane of the AWS Management Console, choose **Databases**.
3. Choose the Aurora global database that needs a secondary Aurora DB cluster. Ensure that the primary Aurora DB cluster is Available.
4. For **Actions**, choose **Add region**.
5. On the **Add a region** page, choose the secondary AWS Region.

You can't choose an AWS Region that already has a secondary Aurora DB cluster for the same Aurora global database. Also, it can't be the same Region as the primary Aurora DB cluster.

6. Complete the remaining fields for the secondary Aurora cluster in the new AWS Region. These are the same configuration options as for any Aurora DB cluster instance.

For an Aurora MySQL-based Aurora global database, disregard the **Enable read replica write forwarding** option. This option has no function after you delete the reader instance.

7. **Add region.** After you finish adding the Region to your Aurora global database, you can see it in the list of **Databases** in the AWS Management Console as shown in the screenshot.

DB identifier	Role	Engine	Region & AZ	Size	Status
west-coast-global	Global	Aurora MySQL	2 regions	2 clusters	Available
ams57west	Primary	Aurora MySQL	us-west-1	2 instances	Available
ams57west-instance-1	Writer	Aurora MySQL	us-west-1b	db.r5.large	Available
ams57west-instance-1-us-west-1c	Reader	Aurora MySQL	us-west-1c	db.r5.large	Available
west-coast-global-cluster-1	Secondary	Aurora MySQL	us-west-2	1 instance	Available
west-coast-global-instance-1	Reader	Aurora MySQL	us-west-2a	db.r5.large	Available

- Check the status of the secondary Aurora DB cluster and its reader instance before continuing, by using the AWS Management Console or the AWS CLI. For example:

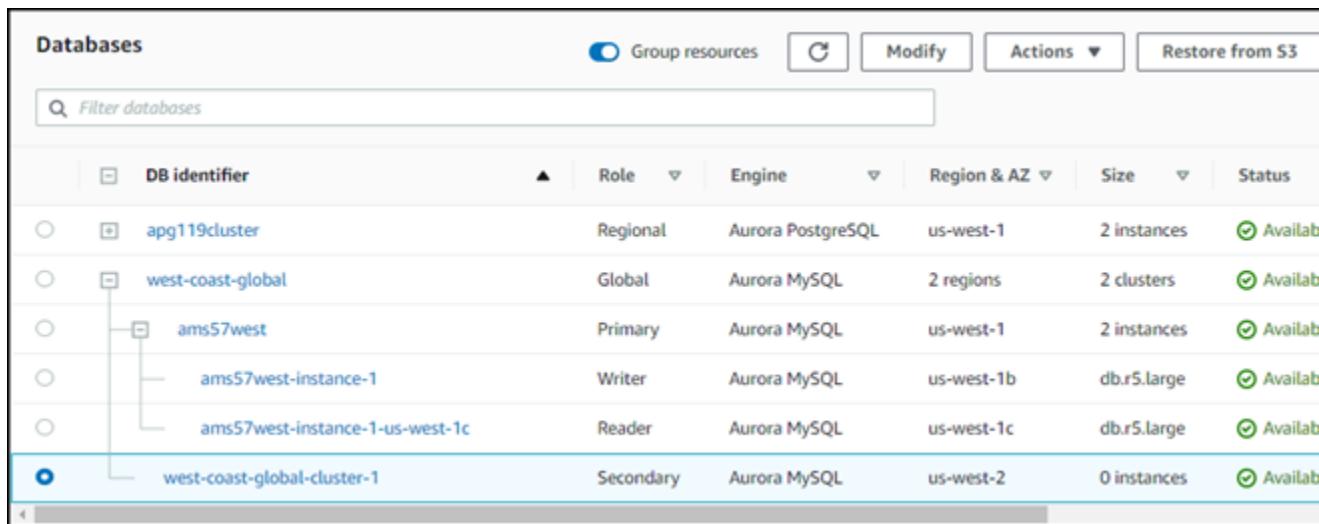
```
$ aws rds describe-db-clusters --db-cluster-identifier secondary-cluster-id --query '*[].[Status]' --output text
```

It can take several minutes for the status of a newly added secondary Aurora DB cluster to change from **creating** to **available**. When the Aurora DB cluster is available, you can delete the reader instance.

- Select the reader instance in the secondary Aurora DB cluster, and then choose **Delete**.

DB identifier	Role	Engine	Region & AZ	Size	Status
west-coast-global	Global	Aurora MySQL	2 regions	2 clusters	Available
ams57west	Primary	Aurora MySQL	us-west-1	2 instances	Available
west-coast-global-cluster-1	Secondary	Aurora MySQL	us-west-2	1 instance	Available
west-coast-global-instance-1	Reader	Aurora MySQL	us-west-2a	db.r5.large	Available

After deleting the reader instance, the secondary cluster remains part of the Aurora global database. It has no instance associated with it, as shown following.



DB identifier	Role	Engine	Region & AZ	Size	Status
apg119cluster	Regional	Aurora PostgreSQL	us-west-1	2 instances	Available
west-coast-global	Global	Aurora MySQL	2 regions	2 clusters	Available
ams57west	Primary	Aurora MySQL	us-west-1	2 instances	Available
ams57west-instance-1	Writer	Aurora MySQL	us-west-1b	db.r5.large	Available
ams57west-instance-1-us-west-1c	Reader	Aurora MySQL	us-west-1c	db.r5.large	Available
west-coast-global-cluster-1	Secondary	Aurora MySQL	us-west-2	0 instances	Available

You can use this headless secondary Aurora DB cluster to [manually recover your Amazon Aurora global database from an unplanned outage in the primary AWS Region \(p. 267\)](#) if such an outage occurs.

Using a snapshot for your Amazon Aurora global database

You can restore a snapshot of an Aurora DB cluster or from an Amazon RDS DB instance to use as the starting point for your Aurora global database. You restore the snapshot and create a new Aurora provisioned DB cluster at the same time. You then add another AWS Region to the restored DB cluster, thus turning it into an Aurora global database. Any Aurora DB cluster that you create using a snapshot in this way becomes the primary cluster of your Aurora global database.

The snapshot that you use can be from a provisioned or from a serverless Aurora DB cluster.

Note

You can't create a provisioned Aurora DB cluster from a snapshot made from an Aurora MySQL 5.6.10a-based global database. A snapshot from an Aurora MySQL 5.6.10a-based global database can only be restored as an Aurora global database.

During the restore process, choose the same DB engine type as the snapshot. For example, suppose that you want to restore a snapshot that was made from an Aurora Serverless v1 DB cluster running Aurora PostgreSQL. In this case, you create an Aurora PostgreSQL DB cluster using that same Aurora DB engine and version.

The restored DB cluster assumes the role of primary cluster for Aurora global database when you add an AWS Regions to it. All data contained in this primary cluster is replicated to any secondary clusters that you add to your Aurora global database.

RDS > Snapshots > Restore snapshot

Restore snapshot

You are creating a new DB Instance from a source DB Instance at a specified time. This new DB Instance will have the default DB Security group and DB Parameter groups.

DB specifications

Engine: Amazon Aurora with MySQL compatibility

Capacity type [Info](#)

- Provisioned
You provision and manage the server instance sizes.
- Serverless
You specify the minimum and maximum amount of resources needed, and Aurora scales the capacity based on database load. This is a good option for intermittent or unpredictable workloads.

► Replication features [Info](#)
Single-master replication is currently selected

Engine version [Info](#)
View the engine versions that support the following database features.

- Show versions that support the global database feature
- Show versions that support the parallel query feature

Version: Aurora (MySQL 5.6) 1.22.1

To see more versions, modify the capacity types. [Info](#)

Managing an Amazon Aurora global database

With the exception of the managed planned failover process, you perform most management operations on the individual clusters that make up an Aurora global database. When you choose **Group related resources** on the **Databases** page in the console, you see the primary cluster and secondary clusters grouped under the associated global database. To find the AWS Regions where a global database's DB clusters are running, its Aurora DB engine and version, and its identifier, use its **Configuration** tab.

The managed planned failover process is available to Aurora global database objects only, not for a single Aurora DB cluster. To learn more, see [Performing managed planned failovers for Amazon Aurora global databases \(p. 268\)](#).

To recover an Aurora global database from an unplanned outage in its primary Region, see [Using failover in an Amazon Aurora global database \(p. 266\)](#).

Topics

- [Modifying an Amazon Aurora global database \(p. 249\)](#)
- [Modifying parameters for an Aurora global database \(p. 250\)](#)
- [Removing a cluster from an Amazon Aurora global database \(p. 251\)](#)
- [Deleting an Amazon Aurora global database \(p. 253\)](#)

Modifying an Amazon Aurora global database

The **Databases** page in the AWS Management Console lists all your Aurora global databases, showing the primary cluster and secondary clusters for each one. The Aurora global database has its own

configuration settings. Specifically, it has AWS Regions associated with its primary and secondary clusters, as shown in the screenshot following.

DB identifier	Role	Engine	Region & AZ	Size	Status
lab-east-west-global	Global	Aurora PostgreSQL	2 regions	2 clusters	Available
lab-sfo-db-cluster	Primary	Aurora PostgreSQL	us-west-1	2 instances	Available
lab-sfo-db-cluster-instance-1	Writer	Aurora PostgreSQL	us-west-1b	db.r4.large	Available
lab-sfo-db-cluster-instance-1-us-west-1c	Reader	Aurora PostgreSQL	us-west-1c	db.r4.large	Available
lab-east-coast-db-cluster	Secondary	Aurora PostgreSQL	us-east-1	2 instances	Available

When you make changes to the Aurora global database, you have a chance to cancel changes, as shown in the following screenshot.

Global database identifier
Enter a name for your global database. The name must be unique across all global databases in your AWS account.
lab-east-west-global-database-01

The global database identifier is case-insensitive, but is stored as all lowercase (as in "mydbinstance"). Constraints: 1 to 60 alphanumeric characters or hyphens. First character must be a letter. Can't contain two consecutive hyphens. Can't end with a hyphen.

Additional configuration

Encryption
Configure encryption keys by modifying member DB clusters.

Cancel **Continue**

When you choose **Continue**, you confirm the changes.

Modifying parameters for an Aurora global database

You can configure the Aurora DB cluster parameter groups independently for each Aurora cluster within the Aurora global database. Most parameters work the same as for other kinds of Aurora clusters. We recommend that you keep settings consistent among all the clusters in a global database. Doing this helps to avoid unexpected behavior changes if you promote a secondary cluster to be the primary.

For example, use the same settings for time zones and character sets to avoid inconsistent behavior if a different cluster takes over as the primary cluster.

The `aurora_enable_repl_bin_log_filtering` and `aurora_enable_replica_log_compression` configuration settings have no effect.

Removing a cluster from an Amazon Aurora global database

You can remove Aurora DB clusters from your Aurora global database for several different reasons. For example, you might want to remove an Aurora DB cluster from an Aurora global database if the primary cluster becomes degraded or isolated. It then becomes a standalone provisioned Aurora DB cluster that could be used to create a new Aurora global database. To learn more, see [Recovering an Amazon Aurora global database from an unplanned outage \(p. 267\)](#).

You also might want to remove Aurora DB clusters because you want to delete an Aurora global database that you no longer need. You can't delete the Aurora global database until after you remove (detach) all associated Aurora DB clusters, leaving the primary for last. For more information, see [Deleting an Amazon Aurora global database \(p. 253\)](#).

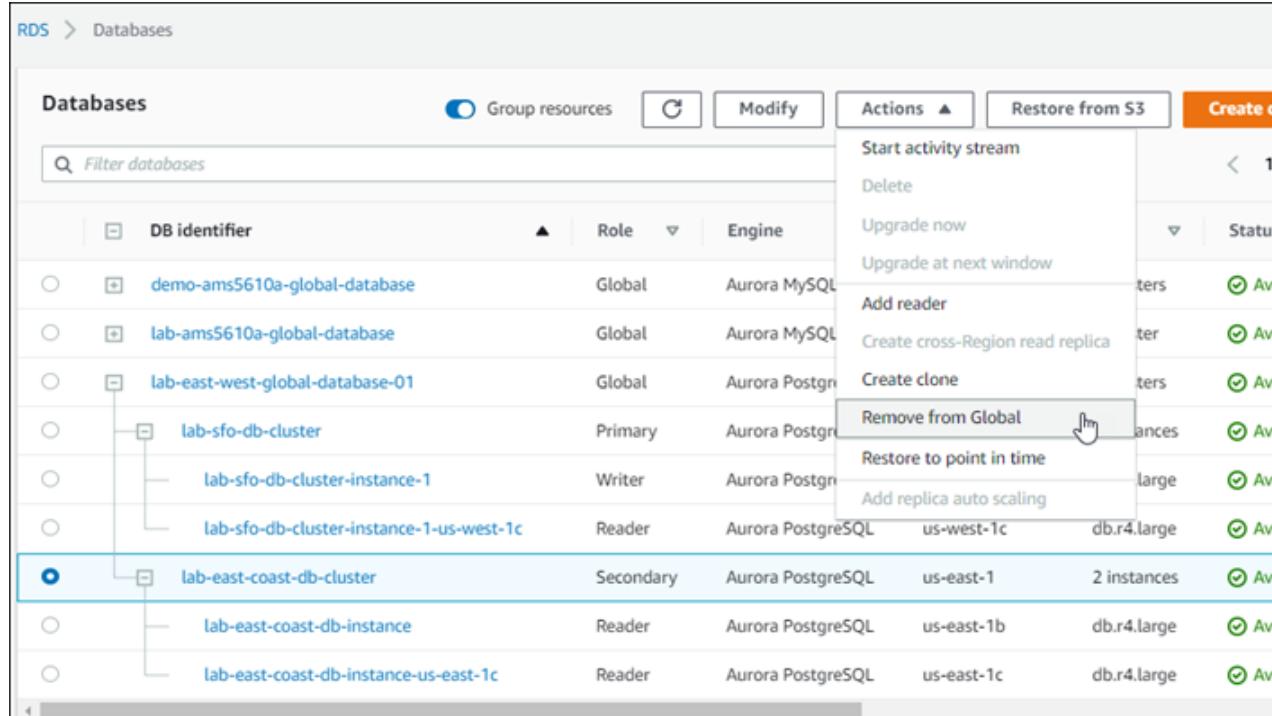
When an Aurora DB cluster is detached from the Aurora global database, it's no longer synchronized with the primary. It becomes a standalone provisioned Aurora DB cluster with full read/write capabilities.

You can remove Aurora DB clusters from your Aurora global database using the AWS Management Console, the AWS CLI, or the RDS API.

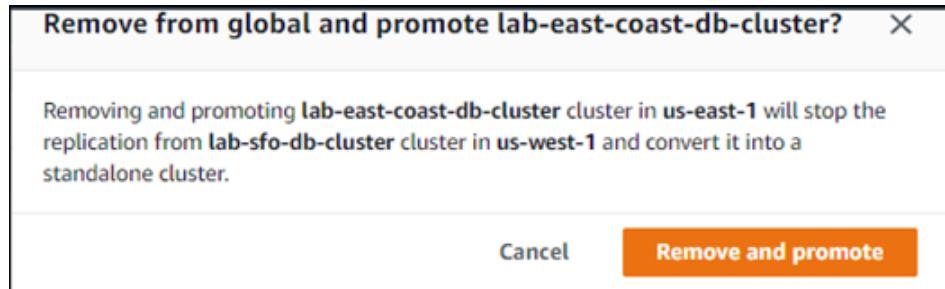
Console

To remove an Aurora cluster from an Aurora global database

1. Sign in to the AWS Management Console and open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. Choose the cluster on the **Databases** page.
3. For **Actions**, choose **Remove from Global**.



You see a prompt to confirm that you want to detach the secondary from the Aurora global database.



- Choose **Remove and promote** to remove the cluster from the global database.

The Aurora DB cluster is no longer serving as a secondary in the Aurora global database, and is no longer synchronized with the primary DB cluster. It is a standalone Aurora DB cluster with full read/write capability.

		Region	Aurora PostgreSQL	us-east-1	2 instances	
○	lab-east-coast-db-cluster	Regional	Aurora PostgreSQL	us-east-1	2 instances	
○	lab-east-coast-db-instance	Writer	Aurora PostgreSQL	us-east-1b	db.r4.large	
○	lab-east-coast-db-instance-us-east-1c	Reader	Aurora PostgreSQL	us-east-1c	db.r4.large	
○	lab-east-west-global-database-01	Global	Aurora PostgreSQL	1 region	1 cluster	
○	lab-sfo-db-cluster	Primary	Aurora PostgreSQL	us-west-1	2 instances	
○	lab-sfo-db-cluster-instance-1	Writer	Aurora PostgreSQL	us-west-1b	db.r4.large	
○	lab-sfo-db-cluster-instance-1-us-west-1c	Reader	Aurora PostgreSQL	us-west-1c	db.r4.large	

After you remove or delete all secondary clusters, then you can remove the primary cluster the same way. You can't detach (remove) the primary Aurora DB cluster from an Aurora global database until after you remove all secondary clusters.

The Aurora global database might remain in the **Databases** list, with zero Regions and AZs. You can delete if you no longer want to use this Aurora global database. For more information, see [Deleting an Amazon Aurora global database \(p. 253\)](#).

AWS CLI

To remove an Aurora cluster from an Aurora global database, run the [remove-from-global-cluster](#) CLI command with the following parameters:

- global-cluster-identifier** – The name (identifier) of your Aurora global database.
- db-cluster-identifier** – The name of each Aurora DB cluster to remove from the Aurora global database. Remove all secondary Aurora DB clusters before removing the primary.

The following examples first remove a secondary cluster and then the primary cluster from an Aurora global database.

For Linux, macOS, or Unix:

```
aws rds --region secondary_region \
    remove-from-global-cluster \
    --db-cluster-identifier secondary_cluster_ARN \
    --global-cluster-identifier global_database_id

aws rds --region primary_region \
```

```
remove-from-global-cluster \
--db-cluster-identifier primary_cluster_ARN \
--global-cluster-identifier global_database_id
```

Repeat the `remove-from-global-cluster --db-cluster-identifier secondary_cluster_ARN` command for each secondary AWS Region in your Aurora global database.

For Windows:

```
aws rds --region secondary_region ^
remove-from-global-cluster ^
--db-cluster-identifier secondary_cluster_ARN ^
--global-cluster-identifier global_database_id

aws rds --region primary_region ^
remove-from-global-cluster ^
--db-cluster-identifier primary_cluster_ARN ^
--global-cluster-identifier global_database_id
```

Repeat the `remove-from-global-cluster --db-cluster-identifier secondary_cluster_ARN` command for each secondary AWS Region in your Aurora global database.

RDS API

To remove an Aurora cluster from an Aurora global database with the RDS API, run the [RemoveFromGlobalCluster](#) action.

Deleting an Amazon Aurora global database

Because an Aurora global database typically holds business-critical data, you can't delete the global database and its associated clusters in a single step. To delete an Aurora global database, do the following:

- Remove all secondary DB clusters from the Aurora global database. Each cluster becomes a standalone Aurora DB cluster. To learn how, see [Removing a cluster from an Amazon Aurora global database \(p. 251\)](#).
- From each standalone Aurora DB cluster, delete all Aurora Replicas.
- Remove the primary DB cluster from the Aurora global database. This becomes a standalone Aurora DB cluster.
- From the Aurora primary DB cluster, first delete all Aurora Replicas, then delete the writer DB instance.

Deleting the writer instance from the newly standalone Aurora DB cluster also typically removes the Aurora DB cluster and the Aurora global database.

For more general information, see [Deleting a DB instance from an Aurora DB cluster \(p. 472\)](#).

To delete an Aurora global database, you can use the AWS Management Console, the AWS CLI, or the RDS API.

Console

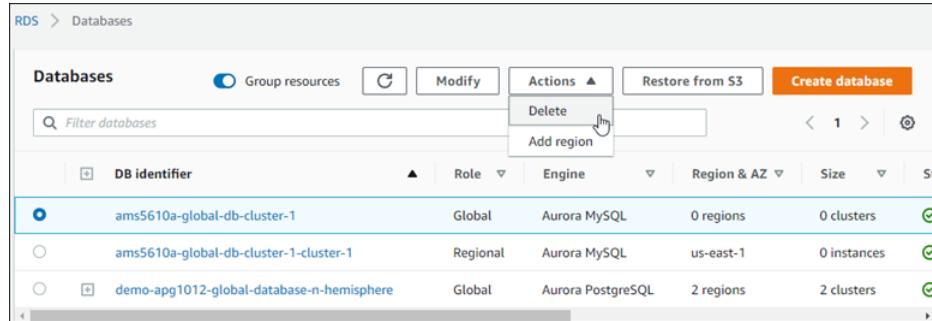
To delete an Aurora global database

1. Sign in to the AWS Management Console and open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. Choose **Databases** and find the Aurora global database you want to delete in the listing.

3. Confirm that all clusters are removed from the Aurora global database. The Aurora global database should show 0 Regions and AZs and a size of 0 clusters.

If the Aurora global database contains any Aurora DB clusters, you can't delete it. If necessary, detach the primary and secondary Aurora DB clusters from the Aurora global database. For more information, see [Removing a cluster from an Amazon Aurora global database \(p. 251\)](#).

4. Choose your Aurora global database in the list, and then choose **Delete** from the Actions menu.



AWS CLI

To delete an Aurora global database, run the [delete-global-cluster](#) CLI command with the name of the AWS Region and the Aurora global database identifier, as shown in the following example.

For Linux, macOS, or Unix:

```
aws rds --region primary_region delete-global-cluster \
--global-cluster-identifier global_database_id
```

For Windows:

```
aws rds --region primary_region delete-global-cluster ^
--global-cluster-identifier global_database_id
```

RDS API

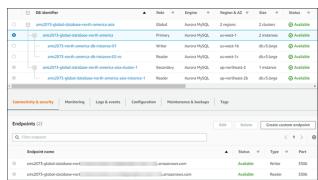
To delete a cluster that is part of an Aurora global database, run the [DeleteGlobalCluster](#) API operation.

Connecting to an Amazon Aurora global database

How you connect to an Aurora global database depends on whether you need to write to the database or read from the database:

- For read-only requests or queries, you connect to the reader endpoint for the Aurora cluster in your AWS Region.
- To run data manipulation language (DML) or data definition language (DDL) statements, you connect to the cluster endpoint for the primary cluster. This endpoint might be in a different AWS Region than your application.

When you view an Aurora global database in the console, you can see all the general-purpose endpoints associated with all of its clusters. The following screenshot shows an example. There is a single cluster endpoint associated with the primary cluster that you use for write operations. The primary cluster and each secondary cluster has a reader endpoint that you use for read-only queries. To minimize latency, choose whichever reader endpoint is in your AWS Region or the AWS Region closest to you. The following shows an Aurora MySQL example.



Using write forwarding in an Amazon Aurora global database

You can reduce the number of endpoints that you need to manage for applications running on your Aurora global database, by using *write forwarding*. This feature of Aurora MySQL lets secondary clusters in an Aurora global database forward SQL statements that perform write operations to the primary cluster. The primary cluster updates the source and then propagates resulting changes back to all secondary AWS Regions.

The write forwarding configuration saves you from implementing your own mechanism to send write operations from a secondary AWS Region to the primary Region. Aurora handles the cross-Region networking setup. Aurora also transmits all necessary session and transactional context for each statement. The data is always changed first on the primary cluster and then replicated to the secondary clusters in the Aurora global database. This way, the primary cluster is the source of truth and always has an up-to-date copy of all your data.

Note

Write forwarding requires Aurora MySQL version 2.08.1 or later.

Topics

- [Enabling write forwarding \(p. 255\)](#)
- [Checking if a secondary cluster has write forwarding enabled \(p. 257\)](#)
- [Application and SQL compatibility with write forwarding \(p. 258\)](#)
- [Isolation and consistency for write forwarding \(p. 259\)](#)
- [Running multipart statements with write forwarding \(p. 262\)](#)
- [Transactions with write forwarding \(p. 262\)](#)
- [Configuration parameters for write forwarding \(p. 262\)](#)
- [Amazon CloudWatch metrics for write forwarding \(p. 263\)](#)

Enabling write forwarding

By default, write forwarding isn't enabled when you add a secondary cluster to an Aurora global database.

To enable write forwarding using the AWS Management Console, choose the **Enable read replica write forwarding** option when you add a Region for a global database. For an existing secondary cluster, modify the cluster to use the **Enable read replica write forwarding** option. To turn off write forwarding, clear the **Enable read replica write forwarding** check box when adding the Region or modifying the secondary cluster.

To enable write forwarding using the AWS CLI, use the `--enable-global-write-forwarding` option. This option works when you create a new secondary cluster using the `create-db-cluster` command. It also works when you modify an existing secondary cluster using the `modify-db-cluster` command. It requires that the global database uses an Aurora version that supports write forwarding. You can turn write forwarding off by using the `--no-enable-global-write-forwarding` option with these same CLI commands.

To enable write forwarding using the Amazon RDS API, set the `EnableGlobalWriteForwarding` parameter to `true`. This parameter works when you create a new secondary cluster using the `CreateDBCluster` operation. It also works when you modify an existing secondary cluster using the `ModifyDBCluster` operation. It requires that the global database uses an Aurora version that supports write forwarding. You can turn write forwarding off by setting the `EnableGlobalWriteForwarding` parameter to `false`.

Note

For a database session to use write forwarding, you also specify a setting for the `aurora_replica_read_consistency` configuration parameter. Do this in every session that uses the write forwarding feature. For information about this parameter, see [Isolation and consistency for write forwarding \(p. 259\)](#).

The following CLI examples show how you can set up an Aurora global database with write forwarding enabled or disabled. The highlighted items represent the commands and options that are important to specify and keep consistent when setting up the infrastructure for an Aurora global database.

The following example creates an Aurora global database, a primary cluster, and a secondary cluster with write forwarding enabled. Substitute your own choices for the user name, password, and primary and secondary AWS Regions.

```
# Create overall global database.  
aws rds create-global-cluster --global-cluster-identifier write-forwarding-test \  
  --engine aurora-mysql --engine-version 5.7.mysql_aurora.2.08.1 \  
  --region us-east-1  
  
# Create primary cluster, in the same AWS Region as the global database.  
aws rds create-db-cluster --global-cluster-identifier write-forwarding-test \  
  --db-cluster-identifier write-forwarding-test-cluster-1 \  
  --engine aurora-mysql --engine-version 5.7.mysql_aurora.2.08.1 \  
  --master-username my_user_name --master-user-password my_password \  
  --region us-east-1  
  
aws rds create-db-instance --db-cluster-identifier write-forwarding-test-cluster-1 \  
  --db-instance-identifier write-forwarding-test-cluster-1-instance-1 \  
  --db-instance-class db.r5.large \  
  --engine aurora-mysql --engine-version 5.7.mysql_aurora.2.08.1 \  
  --region us-east-1  
  
aws rds create-db-instance --db-cluster-identifier write-forwarding-test-cluster-1 \  
  --db-instance-identifier write-forwarding-test-cluster-1-instance-2 \  
  --db-instance-class db.r5.large \  
  --engine aurora-mysql --engine-version 5.7.mysql_aurora.2.08.1 \  
  --region us-east-1  
  
# Create secondary cluster, in a different AWS Region than the global database,  
# with write forwarding enabled.  
aws rds create-db-cluster --global-cluster-identifier write-forwarding-test \  
  --db-cluster-identifier write-forwarding-test-cluster-2 \  
  --engine aurora-mysql --engine-version 5.7.mysql_aurora.2.08.1 \  
  --region us-east-2 \  
  --enable-global-write-forwarding  
  
aws rds create-db-instance --db-cluster-identifier write-forwarding-test-cluster-2 \  
  --db-instance-identifier write-forwarding-test-cluster-2-instance-1 \  
  --db-instance-class db.r5.large \  
  --engine aurora-mysql --engine-version 5.7.mysql_aurora.2.08.1 \  
  --region us-east-2  
  
aws rds create-db-instance --db-cluster-identifier write-forwarding-test-cluster-2 \  
  --db-instance-identifier write-forwarding-test-cluster-2-instance-2 \  
  --db-instance-class db.r5.large \  
  --engine aurora-mysql --engine-version 5.7.mysql_aurora.2.08.1 \
```

```
--region us-east-2
```

The following example continues from the previous one. It creates a secondary cluster without write forwarding enabled, then enables write forwarding. After this example finishes, all secondary clusters in the global database have write forwarding enabled.

```
# Create secondary cluster, in a different AWS Region than the global database,
# without write forwarding enabled.
aws rds create-db-cluster --global-cluster-identifier write-forwarding-test \
--db-cluster-identifier write-forwarding-test-cluster-2 \
--engine aurora-mysql --engine-version 5.7.mysql_aurora.2.08.1 \
--region us-west-1

aws rds create-db-instance --db-cluster-identifier write-forwarding-test-cluster-2 \
--db-instance-identifier write-forwarding-test-cluster-2-instance-1 \
--db-instance-class db.r5.large \
--engine aurora-mysql --engine-version 5.7.mysql_aurora.2.08.1 \
--region us-west-1

aws rds create-db-instance --db-cluster-identifier write-forwarding-test-cluster-2 \
--db-instance-identifier write-forwarding-test-cluster-2-instance-2 \
--db-instance-class db.r5.large \
--engine aurora-mysql --engine-version 5.7.mysql_aurora.2.08.1 \
--region us-west-1

aws rds modify-db-cluster --db-cluster-identifier write-forwarding-test-cluster-2 \
--region us-east-2 \
--enable-global-write-forwarding
```

Checking if a secondary cluster has write forwarding enabled

To determine whether you can use write forwarding from a secondary cluster, you can check whether the cluster has the attribute "GlobalWriteForwardingStatus": "enabled".

In the AWS Management Console, you see Read replica write forwarding on the **Configuration** tab of the details page for the cluster. To see the status of the global write forwarding setting for all of your clusters, run the following AWS CLI command.

A secondary cluster shows the value "enabled" or "disabled" to indicate if write forwarding is turned on or off. A value of null indicates that write forwarding isn't available for that cluster. Either the cluster isn't part of a global database, or is the primary cluster instead of a secondary cluster. The value can also be "enabling" or "disabling" if write forwarding is in the process of being turned on or off.

Example

```
aws rds describe-db-clusters --query '*[].
{DBClusterIdentifier:DBClusterIdentifier,GlobalWriteForwardingStatus:GlobalWriteForwardingStatus}'
[
  {
    "GlobalWriteForwardingStatus": "enabled",
    "DBClusterIdentifier": "aurora-write-forwarding-test-replica-1"
  },
  {
    "GlobalWriteForwardingStatus": "disabled",
    "DBClusterIdentifier": "aurora-write-forwarding-test-replica-2"
  },
  {
    "GlobalWriteForwardingStatus": null,
    "DBClusterIdentifier": "non-global-cluster"
  }
]
```

To find all secondary clusters that have global write forwarding enabled, run the following command. This command also returns the cluster's reader endpoint. You use the secondary cluster's reader endpoint to when you use write forwarding from the secondary to the primary in your Aurora global database.

Example

```
aws rds describe-db-clusters --query 'DBClusters[].  
{DBClusterIdentifier:DBClusterIdentifier,GlobalWriteForwardingStatus:GlobalWriteForwardingStatus,ReaderEndpoint:ReaderEndpoint  
| [?GlobalWriteForwardingStatus == `enabled`]}  
[  
 {  
     "GlobalWriteForwardingStatus": "enabled",  
     "ReaderEndpoint": "aurora-write-forwarding-test-replica-1.cluster-ro-cnpexample.us-west-2.rds.amazonaws.com",  
     "DBClusterIdentifier": "aurora-write-forwarding-test-replica-1"  
 }  
]
```

Application and SQL compatibility with write forwarding

Certain statements aren't allowed or can produce stale results when you use them in a global database with write forwarding. Thus, the `EnableGlobalWriteForwarding` setting is turned off by default for secondary clusters. Before turning it on, check to make sure that your application code isn't affected by any of these restrictions.

You can use the following kinds of SQL statements with write forwarding:

- Data manipulation language (DML) statements, such as `INSERT`, `DELETE`, and `UPDATE`. There are some restrictions on the properties of these statements that you can use with write forwarding, as described following.
- `SELECT ... LOCK IN SHARE MODE` and `SELECT FOR UPDATE` statements.
- `PREPARE` and `EXECUTE` statements.

The following restrictions apply to the SQL statements you use with write forwarding. In some cases, you can use the statements on secondary clusters with write forwarding enabled at the cluster level. This approach works if write forwarding isn't turned on within the session by the `aurora_replica_read_consistency` configuration parameter. Trying to use a statement when it's not allowed because of write forwarding causes an error message with the following format.

```
ERROR 1235 (42000): This version of MySQL doesn't yet support 'operation with write forwarding'.
```

Data definition language (DDL)

Connect to the primary cluster to run DDL statements.

Updating a permanent table using data from a temporary table

You can use temporary tables on secondary clusters with write forwarding enabled. However, you can't use a DML statement to modify a permanent table if the statement refers to a temporary table. For example, you can't use an `INSERT ... SELECT` statement that takes the data from a temporary table. The temporary table exists on the secondary cluster and isn't available when the statement runs on the primary cluster.

XA transactions

You can't use the following statements on a secondary cluster when write forwarding is turned on within the session. You can use these statements on secondary clusters that don't have write forwarding enabled, or within sessions where the `aurora_replica_read_consistency` setting

is empty. Before turning on write forwarding within a session, check if your code uses these statements.

```
XA {START|BEGIN} xid [JOIN|RESUME]  
XA END xid [SUSPEND [FOR MIGRATE]]  
XA PREPARE xid  
XA COMMIT xid [ONE PHASE]  
XA ROLLBACK xid  
XA RECOVER [CONVERT XID]
```

LOAD statements for permanent tables

You can't use the following statements on a secondary cluster with write forwarding enabled.

```
LOAD DATA INFILE 'data.txt' INTO TABLE t1;  
LOAD XML LOCAL INFILE 'test.xml' INTO TABLE t1;
```

You can load data into a temporary table on a secondary cluster. However, make sure that you run any LOAD statements that refer to permanent tables only on the primary cluster.

Plugin statements

You can't use the following statements on a secondary cluster with write forwarding enabled.

```
INSTALL PLUGIN example SONAME 'ha_example.so';  
UNINSTALL PLUGIN example;
```

SAVEPOINT statements

You can't use the following statements on a secondary cluster when write forwarding is turned on within the session. You can use these statements on secondary clusters that don't have write forwarding enabled, or within sessions where the `aurora_replica_read_consistency` setting is blank. Check if your code uses these statements before turning on write forwarding within a session.

```
SAVEPOINT t1_save;  
ROLLBACK TO SAVEPOINT t1_save;  
RELEASE SAVEPOINT t1_save;
```

Isolation and consistency for write forwarding

In sessions that use write forwarding, you can only use the `REPEATABLE READ` isolation level. Although you can also use the `READ COMMITTED` isolation level with read-only clusters in secondary AWS Regions, that isolation level doesn't work with write forwarding. For information about the `REPEATABLE READ` and `READ COMMITTED` isolation levels, see [Aurora MySQL isolation levels \(p. 1070\)](#).

You can control the degree of read consistency on a secondary cluster. The read consistency level determines how much waiting the secondary cluster does before each read operation to ensure that some or all changes are replicated from the primary cluster. You can adjust the read consistency level to ensure that all forwarded write operations from your session are visible in the secondary cluster before any subsequent queries. You can also use this setting to ensure that queries on the secondary cluster always see the most current updates from the primary cluster. This is so even for those submitted by other sessions or other clusters. To specify this type of behavior for your application, you choose a value for the session-level parameter `aurora_replica_read_consistency`.

Important

Always set the `aurora_replica_read_consistency` parameter for any session for which you want to forward writes. If you don't, Aurora doesn't enable write forwarding for that session. This parameter has an empty value by default, so choose a specific value when you use

this parameter. The `aurora_replica_read_consistency` parameter has an effect only on secondary clusters that have write forwarding enabled.

For the `aurora_replica_read_consistency` parameter, you can specify the values `EVENTUAL`, `SESSION`, and `GLOBAL`.

As you increase the consistency level, your application spends more time waiting for changes to be propagated between AWS Regions. You can choose the balance between fast response time and ensuring that changes made in other locations are fully available before your queries run.

With the read consistency set to `EVENTUAL`, queries in a secondary AWS Region that uses write forwarding might see data that is slightly stale due to replication lag. Results of write operations in the same session aren't visible until the write operation is performed on the primary Region and replicated to the current Region. The query doesn't wait for the updated results to be available. Thus, it might retrieve the older data or the updated data, depending on the timing of the statements and the amount of replication lag.

With the read consistency set to `SESSION`, all queries in a secondary AWS Region that uses write forwarding see the results of all changes made in that session. The changes are visible regardless of whether the transaction is committed. If necessary, the query waits for the results of forwarded write operations to be replicated to the current Region. It doesn't wait for updated results from write operations performed in other Regions or in other sessions within the current Region.

With the read consistency set to `GLOBAL`, a session in a secondary AWS Region sees changes made by that session. It also sees all committed changes from both the primary AWS Region and other secondary AWS Regions. Each query might wait for a period that varies depending on the amount of session lag. The query proceeds when the secondary cluster is up-to-date with all committed data from the primary cluster, as of the time that the query began.

For more information about all the parameters involved with write forwarding, see [Configuration parameters for write forwarding \(p. 262\)](#).

Examples of using write forwarding

In the following example, the primary cluster is in the US East (N. Virginia) Region. The secondary cluster is in the US East (Ohio) Region. The example shows the effects of running `INSERT` statements followed by `SELECT` statements. Depending on the value of the `aurora_replica_read_consistency` setting, the results might differ depending on the timing of the statements. To achieve higher consistency, you might wait briefly before issuing the `SELECT` statement. Or Aurora can automatically wait until the results finish replicating before proceeding with `SELECT`.

In this example, there is a read consistency setting of `eventual`. Running an `INSERT` statement immediately followed by a `SELECT` statement still returns the value of `COUNT(*)`. This value reflects the number of rows before the new row is inserted. Running the `SELECT` again a short time later does return the updated row count. The `SELECT` statements don't do any waiting.

```
mysql> set aurora_replica_read_consistency = 'eventual';
mysql> select count(*) from t1;
+-----+
| count(*) |
+-----+
|      5   |
+-----+
1 row in set (0.00 sec)
mysql> insert into t1 values (6); select count(*) from t1;
+-----+
| count(*) |
+-----+
|      5   |
+-----+
1 row in set (0.00 sec)
```

```
mysql> select count(*) from t1;
+-----+
| count(*) |
+-----+
|      6 |
+-----+
1 row in set (0.00 sec)
```

With a read consistency setting of `session`, a `SELECT` statement immediately after an `INSERT` does wait until the changes from the `INSERT` statement are visible. Subsequent `SELECT` statements don't do any waiting.

```
mysql> set aurora_replica_read_consistency = 'session';
mysql> select count(*) from t1;
+-----+
| count(*) |
+-----+
|      6 |
+-----+
1 row in set (0.01 sec)
mysql> insert into t1 values (6); select count(*) from t1; select count(*) from t1;
Query OK, 1 row affected (0.08 sec)
+-----+
| count(*) |
+-----+
|      7 |
+-----+
1 row in set (0.37 sec)
+-----+
| count(*) |
+-----+
|      7 |
+-----+
1 row in set (0.00 sec)
```

With the read consistency setting still set to `session`, introducing a brief wait after performing an `INSERT` statement makes the updated row count available by the time the next `SELECT` statement runs.

```
mysql> insert into t1 values (6); select sleep(2); select count(*) from t1;
Query OK, 1 row affected (0.07 sec)
+-----+
| sleep(2) |
+-----+
|      0 |
+-----+
1 row in set (2.01 sec)
+-----+
| count(*) |
+-----+
|      8 |
+-----+
1 row in set (0.00 sec)
```

With a read consistency setting of `global`, each `SELECT` statement waits to ensure that all data changes as of the start time of the statement are visible before performing the query. The amount of waiting for each `SELECT` statement varies, depending on the amount of replication lag between the primary and secondary clusters.

```
mysql> set aurora_replica_read_consistency = 'global';
mysql> select count(*) from t1;
+-----+
| count(*) |
+-----+
```

```
+-----+
|     8 |
+-----+
1 row in set (0.75 sec)
mysql> select count(*) from t1;
+-----+
| count(*) |
+-----+
|     8 |
+-----+
1 row in set (0.37 sec)
mysql> select count(*) from t1;
+-----+
| count(*) |
+-----+
|     8 |
+-----+
1 row in set (0.66 sec)
```

Running multipart statements with write forwarding

A DML statement might consist of multiple parts, such as a `INSERT ... SELECT` statement or a `DELETE ... WHERE` statement. In this case, the entire statement is forwarded to the primary cluster and run there.

Transactions with write forwarding

Whether the transaction is forwarded to the primary cluster depends on the access mode of the transaction. You can specify the access mode for the transaction by using the `SET TRANSACTION` statement or the `START TRANSACTION` statement. You can also specify the transaction access mode by changing the value of the Aurora MySQL session variable `tx_read_only`. You can only change this session value while you're connected to a secondary cluster that has write forwarding enabled.

If a long-running transaction doesn't issue any statement for a substantial period of time, it might exceed the idle timeout period. This period has a default of one minute. You can increase it up to one day. A transaction that exceeds the idle timeout is canceled by the primary cluster. The next subsequent statement you submit receives a timeout error. Then Aurora rolls back the transaction.

This type of error can occur in other cases when write forwarding becomes unavailable. For example, Aurora cancels any transactions that use write forwarding if you restart the primary cluster or if you turn off the write forwarding configuration setting.

Configuration parameters for write forwarding

The Aurora cluster parameter groups include settings for the write forwarding feature. Because these are cluster parameters, all DB instances in each cluster have the same values for these variables. Details about these parameters are summarized in the following table, with usage notes after the table.

Name	Scope	Type	Default value	Valid values
<code>aurora_fwd_master_idle_timeout</code> (Aurora MySQL version 2)	Global	unsigned integer	60	1–86,400
<code>aurora_fwd_master_max_connections</code> (Aurora MySQL version 2)	Global	unsigned long integer	10	0–90
<code>aurora_fwd_writer_idle_timeout</code> (Aurora MySQL version 3)	Global	unsigned integer	60	1–86,400

Name	Scope	Type	Default value	Valid values
aurora_fwd_writer_max_connections_global (Aurora MySQL version 3)	Global	unsigned long integer	10	0–90
aurora_replica_read_consistency	Session	Enum	"	EVENTUAL, SESSION, GLOBAL

To control incoming write requests from secondary clusters, use these settings on the primary cluster:

- `aurora_fwd_master_idle_timeout`, `aurora_fwd_writer_idle_timeout`: The number of seconds the primary cluster waits for activity on a connection that's forwarded from a secondary cluster before closing it. If the session remains idle beyond this period, Aurora cancels the session.
- `aurora_fwd_master_max_connections_pct`, `aurora_fwd_writer_max_connections_pct`: The upper limit on database connections that can be used on a writer DB instance to handle queries forwarded from readers. It's expressed as a percentage of the `max_connections` setting for the writer DB instance in the primary cluster. For example, if `max_connections` is 800 and `aurora_fwd_master_max_connections_pct` or `aurora_fwd_writer_max_connections_pct` is 10, then the writer allows a maximum of 80 simultaneous forwarded sessions. These connections come from the same connection pool managed by the `max_connections` setting.

This setting applies only on the primary cluster, when one or more secondary clusters have write forwarding enabled. If you decrease the value, existing connections aren't affected. Aurora takes the new value of the setting into account when attempting to create a new connection from a secondary cluster. The default value is 10, representing 10% of the `max_connections` value. If you enable query forwarding on any of the secondary clusters, this setting must have a nonzero value for write operations from secondary clusters to succeed. If the value is zero, the write operations receive the error code `ER_CON_COUNT_ERROR` with the message `Not enough connections on writer to handle your request`.

The `aurora_replica_read_consistency` parameter is a session-level parameter that enables write forwarding. You use it in each session. You can specify `EVENTUAL`, `SESSION`, or `GLOBAL` for read consistency level. To learn more about consistency levels, see [Isolation and consistency for write forwarding \(p. 259\)](#). The following rules apply to this parameter:

- This is a session-level parameter. The default value is " (empty).
- Write forwarding is available in a session only if `aurora_replica_read_consistency` is set to `EVENTUAL` or `SESSION` or `GLOBAL`. This parameter is relevant only in reader instances of secondary clusters that have write forwarding enabled and that are in an Aurora global database.
- You can't set this variable (when empty) or unset (when already set) inside a multistatement transaction. However, you can change it from one valid value (`EVENTUAL`, `SESSION`, or `GLOBAL`) to another valid value (`EVENTUAL`, `SESSION`, or `GLOBAL`) during such a transaction.
- The variable can't be `SET` when write forwarding isn't enabled on the secondary cluster.
- Setting the session variable on a primary cluster doesn't have any effect. If you try to modify this variable on a primary cluster, you receive an error.

Amazon CloudWatch metrics for write forwarding

The following Amazon CloudWatch metrics apply to the primary cluster when you use write forwarding on one or more secondary clusters. These metrics are all measured on the writer DB instance in the primary cluster.

CloudWatch Metric (Aurora MySQL status variable)	Units and description
ForwardingMasterDMLLatency (-)	Milliseconds. Average time to process each forwarded DML statement on the writer DB instance. It doesn't include the time for the secondary cluster to forward the write request. It also doesn't include the time to replicate changes back to the secondary cluster. For Aurora MySQL version 2.
ForwardingMasterOpenSessions (Aurora_fwd_master_open_sessions)	Count. Number of forwarded sessions on the writer DB instance. For Aurora MySQL version 2.
ForwardingMasterDMLThroughput (-)	Count, per second. Number of forwarded DML statements processed each second by this writer DB instance. For Aurora MySQL version 2.
- (Aurora_fwd_master_dml_stmt_duration)	Microseconds. Total duration of DML statements forwarded to this writer DB instance. For Aurora MySQL version 2.
- (Aurora_fwd_master_dml_stmt_count)	Count. Total number of DML statements forwarded to this writer DB instance. For Aurora MySQL version 2.
- (Aurora_fwd_master_select_stmt_duration)	Microseconds. Total duration of SELECT statements forwarded to this writer DB instance. For Aurora MySQL version 2.
- (Aurora_fwd_master_select_stmt_count)	Count. Total number of SELECT statements forwarded to this writer DB instance. For Aurora MySQL version 2.
ForwardingWriterDMLLatency (-)	Milliseconds. Average time to process each forwarded DML statement on the writer DB instance. It doesn't include the time for the secondary cluster to forward the write request. It also doesn't include the time to replicate changes back to the secondary cluster. For Aurora MySQL version 3 and higher.
ForwardingWriterOpenSessions (Aurora_fwd_writer_open_sessions)	Count. Number of forwarded sessions on the writer DB instance. For Aurora MySQL version 3 and higher.
ForwardingWriterDMLThroughput (-)	Count, per second. Number of forwarded DML statements processed each second by this writer DB instance. For Aurora MySQL version 3 and higher.
- (Aurora_fwd_writer_dml_stmt_duration)	Microseconds. Total duration of DML statements forwarded to this writer DB instance.
- (Aurora_fwd_writer_dml_stmt_count)	Count. Total number of DML statements forwarded to this writer DB instance. For Aurora MySQL version 3 and higher.

CloudWatch Metric	Units and description
(Aurora MySQL status variable)	
– (Aurora_fwd_writer_select_stmt_duration)	Microseconds. Total duration of <code>SELECT</code> statements forwarded to this writer DB instance. For Aurora MySQL version 3 and higher.
– (Aurora_fwd_writer_select_stmt_count)	Count. Total number of <code>SELECT</code> statements forwarded to this writer DB instance. For Aurora MySQL version 3 and higher.

The following CloudWatch metrics apply to each secondary cluster. These metrics are measured on each reader DB instance in a secondary cluster with write forwarding enabled.

CloudWatch Metric	Unit and description
(Aurora MySQL status variable)	
ForwardingReplicaDMLLatency (–)	Milliseconds. Average response time in milliseconds of forwarded DMLs on replica.
ForwardingReplicaReadWaitLatency (–)	Milliseconds. Average wait time in milliseconds that a <code>SELECT</code> statement on a reader DB instance waits to catch up to the primary cluster. The degree to which the reader DB instance waits before processing a query depends on the <code>aurora_replica_read_consistency</code> setting.
ForwardingReplicaDMLThroughput (–)	Count (per second). Number of forwarded DML statements processed each second.
ForwardingReplicaReadWaitThroughput (–)	Count (<code>SELECT</code> statements per second). Total number of <code>SELECT</code> statements processed each second in all sessions that are forwarding writes.
ForwardingReplicaOpenSessions (Aurora_fwd_replica_open_sessions)	Count. The number of sessions that are using write forwarding on a reader DB instance.
ForwardingReplicaSelectLatency (–)	Milliseconds. Forwarded <code>SELECT</code> latency, average over all forwarded <code>SELECT</code> statements within the monitoring period.
ForwardingReplicaSelectThroughput (–)	Count per second. Forwarded <code>SELECT</code> throughput, per second average within the monitoring period.
– (Aurora_fwd_replica_dml_stmt_count)	Count. Total number of DML statements forwarded from this reader DB instance.
– (Aurora_fwd_replica_dml_stmt_duration)	Microseconds. Total duration of all DML statements forwarded from this reader DB instance.

CloudWatch Metric (Aurora MySQL status variable)	Unit and description
– (Aurora_fwd_replica_select_stmt_duration)	Microseconds. Total duration of SELECT statements forwarded from this reader DB instance.
– (Aurora_fwd_replica_select_stmt_count)	Count. Total number of SELECT statements forwarded from this reader DB instance.
– (Aurora_fwd_replica_read_wait_duration)	Microseconds. Total duration of waits due to the read consistency setting on this reader DB instance.
– (Aurora_fwd_replica_read_wait_count)	Count. Total number of read-after-write waits on this reader DB instance.
– (Aurora_fwd_replica_errors_session_limit)	Count. Number of sessions rejected by the primary cluster due to the error conditions writer full or Too many forwarded statements in progress.

Using failover in an Amazon Aurora global database

An Aurora global database provides more comprehensive failover capabilities than the [failover provided by a default Aurora DB cluster \(p. 68\)](#). By using an Aurora global database, you can plan for and recover from disaster fairly quickly. Recovery from disaster is typically measured using values for RTO and RPO.

- **Recovery time objective (RTO)** – The time it takes a system to return to a working state after a disaster. In other words, RTO measures downtime. For an Aurora global database, RTO can be in the order of minutes.
- **Recovery point objective (RPO)** – The amount of data that can be lost (measured in time). For an Aurora global database, RPO is typically measured in seconds. With an Aurora PostgreSQL-based global database, you can use the `rds.global_db_rpo` parameter to set and track the upper bound on RPO, but doing so might affect transaction processing on the primary cluster's writer node. For more information, see [Managing RPOs for Aurora PostgreSQL-based global databases \(p. 272\)](#).

With an Aurora global database, there are two different approaches to failover depending on the scenario.

- **Manual unplanned failover ("detach and promote")** – To recover from an unplanned outage or to do disaster recovery (DR) testing, perform a cross-Region failover to one of the secondaries in your Aurora global database. The RTO for this manual process depends on how quickly you can perform the tasks listed in [Recovering an Amazon Aurora global database from an unplanned outage \(p. 267\)](#). The RPO is typically measured in seconds, but this depends on the Aurora storage replication lag across the network at the time of the failure.
- **Managed planned failover** – This feature is intended for controlled environments, such as operational maintenance and other planned operational procedures. By using managed planned failover, you can relocate the primary DB cluster of your Aurora global database to one of the secondary Regions. Because this feature synchronizes secondary DB clusters with the primary before making any other changes, RPO is 0 (no data loss). To learn more, see [Performing managed planned failovers for Amazon Aurora global databases \(p. 268\)](#).

Topics

- [Recovering an Amazon Aurora global database from an unplanned outage \(p. 267\)](#)
- [Performing managed planned failovers for Amazon Aurora global databases \(p. 268\)](#)
- [Managing RPOs for Aurora PostgreSQL-based global databases \(p. 272\)](#)

Recovering an Amazon Aurora global database from an unplanned outage

On very rare occasions, your Aurora global database might experience an unexpected outage in its primary AWS Region. If this happens, your primary Aurora DB cluster and its writer node aren't available, and the replication between the primary cluster and the secondaries ceases. To minimize both downtime (RTO) and data loss (RPO), you can work quickly to perform a cross-Region failover and reconstruct your Aurora global database.

Tip

We recommend that you understand this process before using it. Have a plan ready to quickly proceed at the first sign of a Region-wide issue. Be ready to identify the secondary Region with the least lag time. Use Amazon CloudWatch regularly to track lag times for the secondary clusters. Make sure to test your plan to check that your procedures are complete and accurate, and that staff are trained to perform a DR failover before it really happens.

To fail over to a secondary cluster after an unplanned outage in the primary Region

1. Stop issuing DML statements and other write operations to the primary Aurora DB cluster in the AWS Region with the outage.
2. Identify an Aurora DB cluster from a secondary AWS Region to use as a new primary DB cluster. If you have two or more secondary AWS Regions in your Aurora global database, choose the secondary cluster that has the least lag time.
3. Detach your chosen secondary DB cluster from the Aurora global database.

Removing a secondary DB cluster from an Aurora global database immediately stops the replication from the primary to this secondary and promotes it to a standalone provisioned Aurora DB cluster with full read/write capabilities. Any other secondary Aurora DB clusters associated with the primary cluster in the Region with the outage are still available and can accept calls from your application. They also consume resources. Because you're recreating the Aurora global database, remove the other secondary DB clusters before creating the new Aurora global database in the following steps. Doing this avoids data inconsistencies among the DB clusters in the Aurora global database (*split-brain* issues).

For detailed steps for detaching, see [Removing a cluster from an Amazon Aurora global database \(p. 251\)](#).

4. Reconfigure your application to send all write operations to this now standalone Aurora DB cluster using its new endpoint. If you accepted the provided names when you created the Aurora global database, you can change the endpoint by removing the `-ro` from the cluster's endpoint string in your application.

For example, the secondary cluster's endpoint `my-global.cluster-ro-aaaaaabbbbb.us-west-1.rds.amazonaws.com` becomes `my-global.cluster-aaaaaabbbbb.us-west-1.rds.amazonaws.com` when that cluster is detached from the Aurora global database.

This Aurora DB cluster becomes the primary cluster of a new Aurora global database when you start adding Regions to it in the next step.

5. Add an AWS Region to the DB cluster. When you do this, the replication process from primary to secondary begins. For detailed steps to add a Region, see [Adding an AWS Region to an Amazon Aurora global database \(p. 242\)](#).

6. Add more AWS Regions as needed to recreate the topology needed to support your application.

Make sure that application writes are sent to the correct Aurora DB cluster before, during, and after making these changes. Doing this avoids data inconsistencies among the DB clusters in the Aurora global database (*split-brain* issues).

If you reconfigured in response to an outage in an AWS Region, you might be able to return your Aurora global database to its original primary AWS Region after the outage is resolved. In this case, you use the managed planned failover process. Your Aurora global database must use a version of Aurora PostgreSQL or Aurora MySQL that supports managed planned failovers. For more information, see [Performing managed planned failovers for Amazon Aurora global databases \(p. 268\)](#).

Performing managed planned failovers for Amazon Aurora global databases

By using managed planned failovers, you can relocate the primary cluster of your Aurora global database to a different AWS Region on a routine basis. This approach is intended for controlled environments, such as operational maintenance and other planned operational procedures.

As an example, say a financial institution headquartered in New York has branch offices located in San Francisco, the UK, and Europe. The organization's core business applications use an Aurora global database. Its primary cluster runs in the US East (Ohio) Region. It has secondary clusters running in the US West (N. California) Region, Europe (London) Region, and the Europe (Frankfurt) Region. Every quarter, it relocates the primary cluster from the (current) primary AWS Region to the secondary Region designated for that rotation.

Note

Managed *planned* failover is designed to be used on a healthy Aurora global database. To recover from an unplanned outage or to do disaster recovery (DR) testing, follow the "detach and promote" process detailed in [Recovering an Amazon Aurora global database from an unplanned outage \(p. 267\)](#).

During a managed planned failover, your primary cluster is failed over to your choice of secondary Region while your Aurora global database's existing replication topology is maintained. Before the managed planned failover process begins, Aurora global database synchronizes all secondary clusters with its primary cluster. After ensuring that all clusters are synchronized, the managed planned failover begins. The DB cluster in the primary Region becomes read-only. The chosen secondary cluster promotes one of its read-only nodes to full writer status, thus allowing the cluster to assume the role of primary cluster. Because all secondary clusters were synchronized with the primary at the beginning of the process, the new primary continues operations for the Aurora global database without losing any data. Your database is unavailable for a short time while the primary and selected secondary clusters are assuming their new roles.

To optimize application availability, we recommend that you do the following before using this feature:

- Perform this operation during nonpeak hours or at another time when writes to the primary DB cluster are minimal.
- Take applications offline to prevent writes from being sent to the primary cluster of Aurora global database.
- Check lag times for all secondary Aurora DB clusters in the Aurora global database. Choose the secondary with the least overall lag time for the managed planned failover. Use Amazon CloudWatch to view the `AuroraGlobalDBReplicationLag` metric for all secondaries. This metric tells you how far behind (in milliseconds) a secondary is to the primary DB cluster. Its value is directly proportional to the time it'll take for Aurora to complete failover. In other words, the larger the lag value, the longer the outage, so choose the Region with the least lag.

For more information about CloudWatch metrics for Aurora, see [Cluster-level metrics for Amazon Aurora \(p. 633\)](#).

During a managed planned failover, the chosen secondary DB cluster is promoted to its new role as primary. However, it doesn't inherit the various configuration options of the primary DB cluster. A mismatch in configuration can lead to performance issues, workload incompatibilities, and other anomalous behavior. To avoid such issues, we recommend that you resolve differences between your Aurora global database clusters for the following:

- **Configure Aurora DB cluster parameter group for the new primary, if necessary** – You can configure your Aurora DB cluster parameter groups independently for each Aurora cluster in your Aurora global database. That means that when you promote a secondary DB cluster to take over the primary role, the parameter group from the secondary might be configured differently than for the primary. If so, modify the promoted secondary DB cluster's parameter group to conform to your primary cluster's settings. To learn how, see [Modifying parameters for an Aurora global database \(p. 250\)](#).
- **Configure monitoring tools and options, such as Amazon CloudWatch Events and alarms** – Configure the promoted DB cluster with the same logging ability, alarms, and so on as needed for the global database. As with parameter groups, configuration for these features isn't inherited from the primary during the failover process. For more information about Aurora DB clusters and monitoring, see [Overview of monitoring Amazon Aurora](#).
- **Configure integrations with other AWS services** – If your Aurora global database integrates with AWS services, such as AWS Secrets Manager, AWS Identity and Access Management, Amazon S3, and AWS Lambda, you need to make sure these are configured as needed. For more information about integrating Aurora global databases with IAM, Amazon S3 and Lambda, see [Using Amazon Aurora global databases with other AWS services \(p. 279\)](#). To learn more about Secrets Manager, see [How to automate replication of secrets in AWS Secrets Manager across AWS Regions](#).

When the failover process completes, the promoted Aurora DB cluster can handle write operations for the Aurora global database. Make sure to change the endpoint for your application to use the new endpoint. If you accepted the provided names when you created the Aurora global database, you can change the endpoint by removing the `-ro` from the promoted cluster's endpoint string in your application.

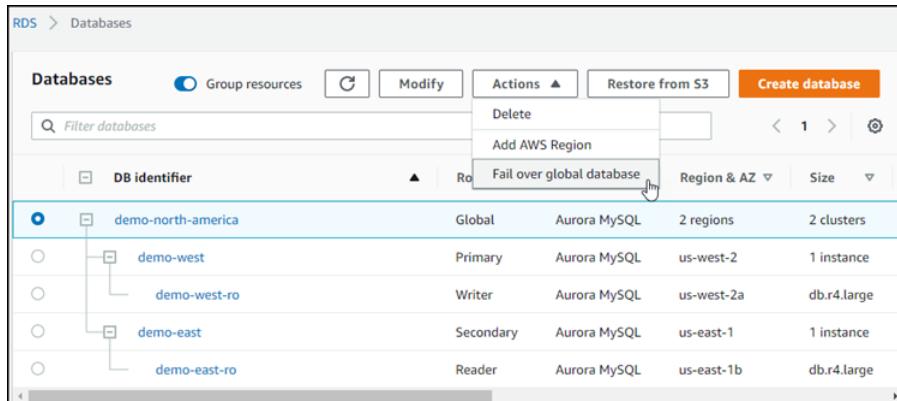
For example, the secondary cluster's endpoint `my-global.cluster-ro-aaaaaabbbbb.us-west-1.rds.amazonaws.com` becomes `my-global.cluster-aaaaaabbbbb.us-west-1.rds.amazonaws.com` when that cluster is promoted to primary.

You can fail over your Aurora global database using the AWS Management Console, the AWS CLI, or the RDS API.

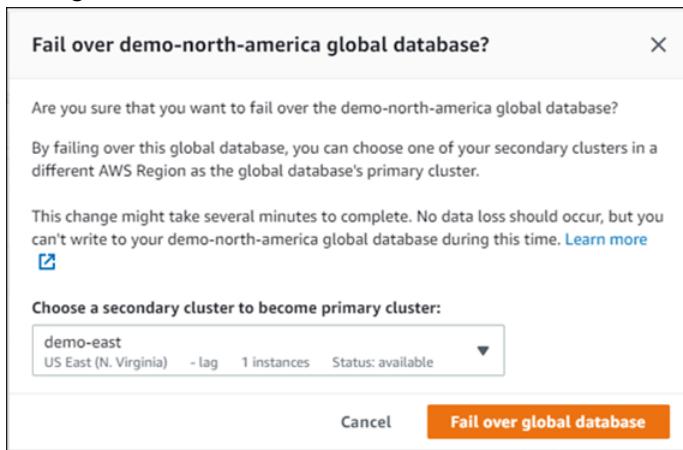
Console

To start the failover process on your Aurora global database

1. Sign in to the AWS Management Console and open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. Choose **Databases** and find the Aurora global database you want to fail over.
3. Choose **Fail over global database** from Actions menu. The failover process doesn't begin until after you choose the failover target in the next step. At this point, the failover is pending.



- Choose the secondary Aurora DB cluster that you want to promote to primary. The secondary DB cluster must be **available**. If you have more than one secondary DB cluster, you can compare the **lag** amount for all secondaries and choose the one with the smallest amount of lag.



- Choose **Fail over global database** to confirm your choice of secondary DB cluster and begin the failover process.

Tip

The failover process can take some time to complete. You can cancel once the process is underway, but it can take some time to return your Aurora global database to its original configuration.

DB identifier	Role	Engine	Region & AZ	Size
demo-north-america	Global	Aurora MySQL	2 regions	2 clusters
demo-west	Pending	Aurora MySQL	us-west-2	1 instance
demo-west-ro	Writer	Aurora MySQL	us-west-2a	db.r4.large
demo-east	Pending	Aurora MySQL	us-east-1	1 instance
demo-east-ro	Reader	Aurora MySQL	us-east-1b	db.r4.large

The **Status** column of the Databases list shows the state of each Aurora DB instance and Aurora DB cluster during the failover process.

DB identifier	Role	Engine	Region & AZ	Size	Status
demo-north-america	Global	Aurora MySQL	2 regions	2 clusters	Failing-over
demo-east	Pending	Aurora MySQL	us-east-1	1 instance	Modifying
demo-east-ro	Reader	Aurora MySQL	us-east-1b	db.r4.large	Modifying
demo-west	Pending	Aurora MySQL	us-west-2	1 instance	Modifying
demo-west-ro	Writer	Aurora MySQL	us-west-2a	db.r4.large	Modifying

The status bar at the top of the Console displays progress and provides a **Cancel failover** option. If you choose **Cancel failover**, you're given the option to proceed with the failover or to cancel the failover process.

Are you sure that you want to cancel the failover of the global database lab-operations?

If you do, these actions occur:

- The lab-test-operations cluster reverts to the primary cluster.
- The lab-operations-cluster-1 cluster reverts to a secondary cluster.

Close **Cancel failover**

Choose **Cancel failover** if you want to cancel the failover.

- Choose **Close** to continue failing over and dismiss the prompt.

When the failover completes, you can see the Aurora DB clusters and their current state in the **Databases** list, as shown following.

Databases					
	DB Identifier	Role	Engine	Region & AZ	Size
...	demo-north-america	Global	Aurora MySQL	2 regions	2 clusters
...	demo-east	Primary	Aurora MySQL	us-east-1	1 instance
...	demo-east-ro	Writer	Aurora MySQL	us-east-1b	db.r4.large
...	demo-west	Secondary	Aurora MySQL	us-west-2	1 instance
...	demo-west-ro	Reader	Aurora MySQL	us-west-2a	db.r4.large

AWS CLI

To fail over an Aurora global database

Use the [failover-global-cluster](#) CLI command to fail over your Aurora global database. With the command, pass values for the following parameters.

- **--region** – Specify the AWS Region where the primary DB cluster of the Aurora global database is running.
- **--global-cluster-identifier** – Specify the name of your Aurora global database.
- **--target-db-cluster-identifier** – Specify the Amazon Resource Name (ARN) of the Aurora DB cluster that you want to promote to be the primary for the Aurora global database.

For Linux, macOS, or Unix:

```
aws rds --region aws-Region \
    failover-global-cluster --global-cluster-identifier global_database_id \
    --target-db-cluster-identifier ARN-of-secondary-to-promote
```

For Windows:

```
aws rds --region aws-Region ^
    failover-global-cluster --global-cluster-identifier global_database_id ^
    --target-db-cluster-identifier ARN-of-secondary-to-promote
```

RDS API

To fail over an Aurora global database, run the [FailoverGlobalCluster](#) API operation.

Managing RPOs for Aurora PostgreSQL-based global databases

With an Aurora PostgreSQL-based global database, you can manage the recovery point objective (RPO) for your Aurora global database by using PostgreSQL's `rds.global_db_rpo` parameter. RPO represents maximum amount of data that can be lost in the event of an outage.

When you set an RPO for your Aurora PostgreSQL-based global database, Aurora monitors the *RPO lag time* of all secondary clusters to make sure that at least one secondary cluster stays within the target RPO window. RPO lag time is another time-based metric.

The RPO is used when your database resumes operations in a new AWS Region after a failover. Aurora evaluates RPO and RPO lag times to commit (or block) transactions on the primary as follows:

- Commits the transaction if at least one secondary DB cluster has an RPO lag time less than the RPO.

- Blocks the transaction if all secondary DB clusters have RPO lag times that are larger than the RPO. It also logs the event to the PostgreSQL log file and emits "wait" events that show the blocked sessions.

In other words, if all secondary clusters are behind the target RPO, Aurora pauses transactions on the primary cluster until at least one of the secondary clusters catches up. Paused transactions are resumed and committed as soon as the lag time of at least one secondary DB cluster becomes less than the RPO. The result is that no transactions can commit until the RPO is met.

If you set this parameter as outlined in the following, you can then also monitor the metrics that it generates. You can do so by using `psql` or another tool to query the Aurora global database's primary DB cluster and obtain detailed information about your Aurora PostgreSQL-based global database's operations. To learn how, see [Monitoring Aurora PostgreSQL-based Aurora global databases \(p. 277\)](#).

Topics

- [Setting the recovery point objective \(p. 273\)](#)
- [Viewing the recovery point objective \(p. 274\)](#)
- [Disabling the recovery point objective \(p. 275\)](#)

Setting the recovery point objective

The `rds.global_db_rpo` parameter controls the RPO setting for a PostgreSQL database. This parameter is supported by Aurora PostgreSQL. Valid values for `rds.global_db_rpo` range from 20 seconds to 2,147,483,647 seconds (68 years). Choose a realistic value to meet your business need and use case. For example, you might want to allow up to 10 minutes for your RPO, in which case you set the value to 600.

You can set this value for your Aurora PostgreSQL-based global database by using the AWS Management Console, the AWS CLI, or the RDS API.

Console

To set the RPO

1. Sign in to the AWS Management Console and open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. Choose the primary cluster of your Aurora global database and open the **Configuration** tab to find its DB cluster parameter group. For example, the default parameter group for a primary DB cluster running Aurora PostgreSQL 11.7 is `default.aurora-postgresql11`.

Parameter groups can't be edited directly. Instead, you do the following:

- Create a custom DB cluster parameter group using the appropriate default parameter group as the starting point. For example, create a custom DB cluster parameter group based on the `default.aurora-postgresql11`.
- On your custom DB parameter group, set the value of the `rds.global_db_rpo` parameter to meet your use case. Valid values range from 20 seconds up to the maximum integer value of 2,147,483,647 (68 years).
- Apply the modified DB cluster parameter group to your Aurora DB cluster.

For more information, see [Modifying parameters in a DB cluster parameter group \(p. 349\)](#).

AWS CLI

To set the `rds.global_db_rpo` parameter, use the `modify-db-cluster-parameter-group` CLI command. In the command, specify the name of your primary cluster's parameter group and values for RPO parameter.

The following example sets the RPO to 600 seconds (10 minutes) for the primary DB cluster's parameter group named `my_custom_global_parameter_group`.

For Linux, macOS, or Unix:

```
aws rds modify-db-cluster-parameter-group \
--db-cluster-parameter-group-name my_custom_global_parameter_group \
--parameters "ParameterName=rds.global_db_rpo,ParameterValue=600,ApplyMethod=immediate"
```

For Windows:

```
aws rds modify-db-cluster-parameter-group ^
--db-cluster-parameter-group-name my_custom_global_parameter_group ^
--parameters "ParameterName=rds.global_db_rpo,ParameterValue=600,ApplyMethod=immediate"
```

RDS API

To modify the `rds.global_db_rpo` parameter, use the Amazon RDS [ModifyDBClusterParameterGroup](#) API operation.

Viewing the recovery point objective

The recovery point objective (RPO) of a global database is stored in the `rds.global_db_rpo` parameter for each DB cluster. You can connect to the endpoint for the secondary cluster you want to view and use `psql` to query the instance for this value.

```
db-name=>show rds.global_db_rpo;
```

If this parameter isn't set, the query returns the following:

```
rds.global_db_rpo
-----
-1
(1 row)
```

This next response is from a secondary DB cluster that has 1 minute RPO setting.

```
rds.global_db_rpo
-----
60
(1 row)
```

You can also use the CLI to get values for find out if `rds.global_db_rpo` is active on any of the Aurora DB clusters by using the CLI to get values of all user parameters for the cluster.

For Linux, macOS, or Unix:

```
aws rds describe-db-cluster-parameters \
--db-cluster-parameter-group-name lab-test-apg-global \
--source user
```

For Windows:

```
aws rds describe-db-cluster-parameters ^
--db-cluster-parameter-group-name lab-test-apg-global *
```

```
--source user
```

The command returns output similar to the following for all `user` parameters that aren't default-engine or system DB cluster parameters.

```
{  
    "Parameters": [  
        {  
            "ParameterName": "rds.global_db_rpo",  
            "ParameterValue": "60",  
            "Description": "(s) Recovery point objective threshold, in seconds, that blocks user commits when it is violated.",  
            "Source": "user",  
            "ApplyType": "dynamic",  
            "DataType": "integer",  
            "AllowedValues": "20-2147483647",  
            "IsModifiable": true,  
            "ApplyMethod": "immediate",  
            "SupportedEngineModes": [  
                "provisioned"  
            ]  
        }  
    ]  
}
```

To learn more about viewing parameters of the cluster parameter group, see [Viewing parameter values for a DB cluster parameter group \(p. 360\)](#).

Disabling the recovery point objective

To disable the RPO, reset the `rds.global_db_rpo` parameter. You can reset parameters using the AWS Management Console, the AWS CLI, or the RDS API.

Console

To disable the RPO

1. Sign in to the AWS Management Console and open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. In the navigation pane, choose **Parameter groups**.
3. In the list, choose your primary DB cluster parameter group.
4. Choose **Edit parameters**.
5. Choose the box next to the `rds.global_db_rpo` parameter.
6. Choose **Reset**.
7. When the screen shows **Reset parameters in DB parameter group**, choose **Reset parameters**.

For more information on how to reset a parameter with the console, see [Modifying parameters in a DB cluster parameter group \(p. 349\)](#).

AWS CLI

To reset the `rds.global_db_rpo` parameter, use the `reset-db-cluster-parameter-group` command.

For Linux, macOS, or Unix:

```
aws rds reset-db-cluster-parameter-group \  
  --db-cluster-parameter-group-name global_db_cluster_parameter_group \  
  \
```

```
--parameters "ParameterName=rds.global_db_rpo,ApplyMethod=immediate"
```

For Windows:

```
aws rds reset-db-cluster-parameter-group ^
--db-cluster-parameter-group-name global_db_cluster_parameter_group ^
--parameters "ParameterName=rds.global_db_rpo,ApplyMethod=immediate"
```

RDS API

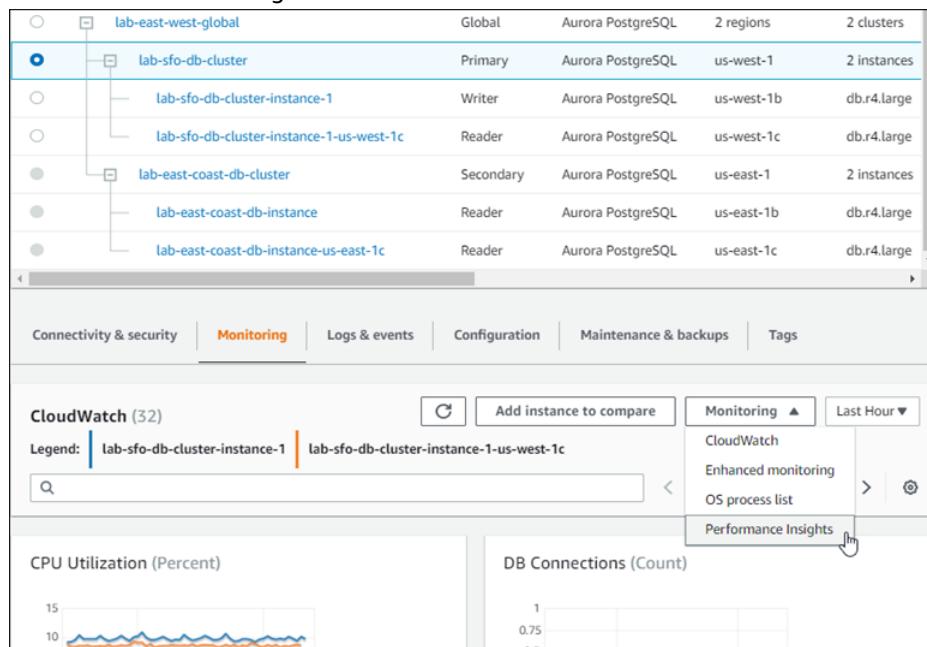
To reset the `rds.global_db_rpo` parameter, use the Amazon RDS API [ResetDBClusterParameterGroup](#) operation.

Monitoring an Amazon Aurora global database

When you create the Aurora DB clusters that make up your Aurora global database, you can choose many options that let you monitor your DB cluster's performance. These options include the following:

- Amazon RDS Performance Insights – Enables performance schema in the underlying Aurora database engine. To learn more about Performance Insights and Aurora global databases, see [Monitoring an Amazon Aurora global database with Amazon RDS Performance Insights \(p. 277\)](#).
- Enhanced monitoring – Generates metrics for process or thread utilization on the CPU.
- Amazon CloudWatch Logs – Publishes specified log types to CloudWatch Logs. Error logs are published by default, but you can choose other logs specific to your Aurora database engine.
 - For Aurora MySQL-based Aurora DB clusters, you can export the audit log, general log, and slow query log.
 - For Aurora PostgreSQL-based Aurora DB clusters, you can export the Postgresql log.
- For Aurora PostgreSQL-based global databases, you can use certain functions to check status of your Aurora global database and its instances. To learn how, see [Monitoring Aurora PostgreSQL-based Aurora global databases \(p. 277\)](#).

The following screenshot shows some of the options available on the Monitoring tab of a primary Aurora DB cluster in an Aurora global database.



For more information, see [Monitoring metrics in an Amazon Aurora cluster \(p. 541\)](#).

Monitoring an Amazon Aurora global database with Amazon RDS Performance Insights

You can use Amazon RDS Performance Insights for your Aurora global databases. You enable this feature individually, for each Aurora DB cluster in your Aurora global database. To do so, you choose **Enable Performance Insights** in the **Additional configuration** section of the Create database page. Or you can modify your Aurora DB clusters to use this feature after they are up and running. You can enable or turn off Performance Insights for each cluster that's part of your Aurora global database.

The reports created by Performance Insights apply to each cluster in the global database. When you add a new secondary AWS Region to an Aurora global database that's already using Performance Insights, be sure that you enable Performance Insights in the newly added cluster. It doesn't inherit the Performance Insights setting from the existing global database.

You can switch AWS Regions while viewing the Performance Insights page for a DB instance that's attached to a global database. However, you might not see performance information immediately after switching AWS Regions. Although the DB instances might have identical names in each AWS Region, the associated Performance Insights URL is different for each DB instance. After switching AWS Regions, choose the name of the DB instance again in the Performance Insights navigation pane.

For DB instances associated with a global database, the factors affecting performance might be different in each AWS Region. For example, the DB instances in each AWS Region might have different capacity.

To learn more about using Performance Insights, see [Monitoring DB load with Performance Insights on Amazon Aurora \(p. 573\)](#).

Monitoring Aurora PostgreSQL-based Aurora global databases

To view the status of a global database, use the `aurora_global_db_status` and `aurora_global_db_instance_status` functions.

Note

Only Aurora PostgreSQL supports the `aurora_global_db_status` and `aurora_global_db_instance_status` functions.

To monitor an Aurora PostgreSQL-based global database

1. Connect to the global database primary cluster endpoint using a PostgreSQL utility such as `psql`. For more information about how to connect, see [Connecting to an Amazon Aurora global database \(p. 254\)](#).
2. Use the `aurora_global_db_status` function in a `psql` command to list the primary and secondary volumes. This shows the lag times of the global database secondary DB clusters.

```
postgres=> select * from aurora_global_db_status();
```

aws_region	highest_lsn_written	durability_lag_in_msec	rpo_lag_in_msec	last_lag_calculation_time	feedback_epoch	feedback_xmin
us-east-1	93763984222	-1	-1	1970-01-01 00:00:00+00	0	0
us-west-2	93763984222	900	1090	2020-05-12 22:49:14.328+00	2	3315479243

(2 rows)

The output includes a row for each DB cluster of the global database containing the following columns:

- **aws_region** – The AWS Region that this DB cluster is in. For tables listing AWS Regions by engine, see [Regions and Availability Zones](#).
- **highest_lsn_written** – The highest log sequence number (LSN) currently written on this DB cluster.

A *log sequence number (LSN)* is a unique sequential number that identifies a record in the database transaction log. LSNs are ordered such that a larger LSN represents a later transaction.

- **durability_lag_in_msec** – The timestamp difference between the highest log sequence number written on a secondary DB cluster (**highest_lsn_written**) and the **highest_lsn_written** on the primary DB cluster.
- **rpo_lag_in_msec** – The recovery point objective (RPO) lag. This lag is the time difference between the most recent user transaction commit stored on a secondary DB cluster and the most recent user transaction commit stored on the primary DB cluster.
- **last_lag_calculation_time** – The timestamp when values were last calculated for **durability_lag_in_msec** and **rpo_lag_in_msec**.
- **feedback_epoch** – The epoch the secondary DB cluster uses when it generates hot standby information.

Hot standby is when a DB cluster can connect and query while the server is in recovery or standby mode. Hot standby feedback is information about the DB cluster when it's in hot standby. For more information, see [Hot standby](#) in the PostgreSQL documentation.

- **feedback_xmin** – The minimum (oldest) active transaction ID used by the secondary DB cluster.

3. Use the `aurora_global_db_instance_status` function to list all secondary DB instances for both the primary DB cluster and secondary DB clusters.

```
postgres=> select * from aurora_global_db_instance_status();
```

server_id	aws_region	durable_lsn	highest_lsn_rcvd	feedback_epoch	feedback_xmin	oldest_read_view_lsn	visibility_lag_in_msec	session_id
apg-global-db-rpo-mammothrw-elephantro-1-n1	us-east-1	93763985102						MASTER_SESSION_ID
apg-global-db-rpo-mammothrw-elephantro-1-n2	us-east-1	93763985099	93763985102		2		10	f38430cf-6576-479a-b296-dc06b1b1964a
apg-global-db-rpo-elephantro-mammothrw-n1	us-west-2	93763985095	93763985099		2		1017	0d9f1d98-04ad-4aa4-8fdd-e08674cbbbfe
								3315479243

(3 rows)

The output includes a row for each DB instance of the global database containing the following columns:

- **server_id** – The server identifier for the DB instance.
- **session_id** – A unique identifier for the current session.
- **aws_region** – The AWS Region that this DB instance is in. For tables listing AWS Regions by engine, see [Regions and Availability Zones](#).
- **durable_lsn** – The LSN made durable in storage.

- **highest_lsn_rcvd** – The highest LSN received by the DB Instance from the writer DB Instance.
- **feedback_epoch** – The epoch the DB instance uses when it generates hot standby information.

Hot standby is when a DB instance can connect and query while the server is in recovery or standby mode. Hot standby feedback is information about the DB instance when it's in hot standby. For more information, see the PostgreSQL documentation on [Hot standby](#).

- **feedback_xmin** – The minimum (oldest) active transaction ID used by the DB instance.
- **oldest_read_view_lsn** – The oldest LSN used by the DB instance to read from storage.
- **visibility_lag_in_msec** – How far this DB instance is lagging behind the writer DB instance.

To see how these values change over time, consider the following transaction block where a table insert takes an hour.

```
psql> BEGIN;
psql> INSERT INTO table1 SELECT Large_Data_That_Takes_1_Hr_To_Insert;
psql> COMMIT;
```

In some cases, there might be a network disconnect between the primary DB cluster and the secondary DB cluster after the `BEGIN` statement. If so, the secondary DB cluster's `replication_lag_in_msec` value starts increasing. At the end of the `INSERT` statement, the `replication_lag_in_msec` value is 1 hour. However, the `rpo_lag_in_msec` value is 0 because all the user data committed between the primary DB cluster and secondary DB cluster are still the same. As soon as the `COMMIT` statement completes, the `rpo_lag_in_msec` value increases.

Using Amazon Aurora global databases with other AWS services

You can use your Aurora global databases with other AWS services, such as Amazon S3 and AWS Lambda. Doing so requires that all Aurora DB clusters in your global database have the same privileges, external functions, and so on in the respective AWS Regions. Because a read-only Aurora secondary DB cluster in an Aurora global database can be promoted to the role of primary, we recommend that you set up write privileges ahead of time, on all Aurora DB clusters for any services you plan to use with your Aurora global database.

The following procedures summarize the actions to take for each AWS service.

To invoke AWS Lambda functions from an Aurora global database

1. For all the Aurora clusters that make up the Aurora global database, perform the procedures in [Invoking a Lambda function from an Amazon Aurora MySQL DB cluster \(p. 1010\)](#).
2. For each cluster in the Aurora global database, set the (ARN) of the new IAM (IAM) role.
3. To permit database users in an Aurora global database to invoke Lambda functions, associate the role that you created in [Creating an IAM role to allow Amazon Aurora to access AWS services \(p. 991\)](#) with each cluster in the Aurora global database.
4. Configure each cluster in the Aurora global database to allow outbound connections to Lambda. For instructions, see [Enabling network communication from Amazon Aurora MySQL to other AWS services \(p. 996\)](#).

To load data from Amazon S3

1. For all the Aurora clusters that make up the Aurora global database, perform the procedures in [Loading data into an Amazon Aurora MySQL DB cluster from text files in an Amazon S3 bucket \(p. 997\)](#).

2. For each Aurora cluster in the global database, set either the `aurora_load_from_s3_role` or `aws_default_s3_role` DB cluster parameter to the Amazon Resource Name (ARN) of the new IAM role. If an IAM role isn't specified for `aurora_load_from_s3_role`, Aurora uses the IAM role specified in `aws_default_s3_role`.
3. To permit database users in an Aurora global database to access S3, associate the role that you created in [Creating an IAM role to allow Amazon Aurora to access AWS services \(p. 991\)](#) with each Aurora cluster in the global database.
4. Configure each Aurora cluster in the global database to allow outbound connections to S3. For instructions, see [Enabling network communication from Amazon Aurora MySQL to other AWS services \(p. 996\)](#).

To save queried data to Amazon S3

1. For all the Aurora clusters that make up the Aurora global database, perform the procedures in [Saving data from an Amazon Aurora MySQL DB cluster into text files in an Amazon S3 bucket \(p. 1004\)](#).
2. For each Aurora cluster in the global database, set either the `aurora_select_into_s3_role` or `aws_default_s3_role` DB cluster parameter to the Amazon Resource Name (ARN) of the new IAM role. If an IAM role isn't specified for `aurora_select_into_s3_role`, Aurora uses the IAM role specified in `aws_default_s3_role`.
3. To permit database users in an Aurora global database to access S3, associate the role that you created in [Creating an IAM role to allow Amazon Aurora to access AWS services \(p. 991\)](#) with each Aurora cluster in the global database.
4. Configure each Aurora cluster in the global database to allow outbound connections to S3. For instructions, see [Enabling network communication from Amazon Aurora MySQL to other AWS services \(p. 996\)](#).

Upgrading an Amazon Aurora global database

Upgrading an Aurora global database follows the same procedures as upgrading Aurora DB clusters. However, following are some important differences to take note of before you start the process.

Major version upgrades

When you perform a major version upgrade of an Amazon Aurora global database, you upgrade the global database cluster instead the individual clusters that it contains.

To learn how to upgrade an Aurora PostgreSQL global database to a higher major version, see [In-place major upgrades for global databases \(p. 1687\)](#). To learn how to upgrade an Aurora MySQL global database to a higher major version, see [In-place major upgrades for global databases \(p. 1101\)](#).

Note

With an Aurora global database based on Aurora PostgreSQL, you can't perform a major version upgrade of the Aurora DB engine if the recovery point objective (RPO) feature is turned on.

Minor version upgrades

For a minor upgrade on an Aurora global database, you upgrade all of the secondary clusters before you upgrade the primary cluster.

To learn how to upgrade an Aurora PostgreSQL global database to a higher minor version, see [Manually upgrading the Aurora PostgreSQL engine \(p. 1686\)](#). To learn how to upgrade an Aurora MySQL global database to a higher minor version, see [Upgrading Aurora MySQL by modifying the engine version \(p. 1088\)](#).

Connecting to an Amazon Aurora DB cluster

You can connect to an Aurora DB cluster using the same tools that you use to connect to a MySQL or PostgreSQL database. You specify a connection string with any script, utility, or application that connects to a MySQL or PostgreSQL DB instance. You use the same public key for Secure Sockets Layer (SSL) connections.

In the connection string, you typically use the host and port information from special endpoints associated with the DB cluster. With these endpoints, you can use the same connection parameters regardless of how many DB instances are in the cluster. You also use the host and port information from a specific DB instance in your Aurora DB cluster for specialized tasks, such as troubleshooting.

Note

For Aurora Serverless v1 DB clusters, you connect to the database endpoint rather than to the DB instance. You can find the database endpoint for an Aurora Serverless v1 DB cluster on the **Connectivity & security** tab of the AWS Management Console. For more information, see [Using Amazon Aurora Serverless v1 \(p. 147\)](#).

Regardless of the Aurora DB engine and specific tools you use to work with the DB cluster or instance, the endpoint must be accessible. An Amazon Aurora DB cluster can be created only in a virtual private cloud (VPC) based on the Amazon VPC service. That means that you access the endpoint from either inside the VPC or outside the VPC using one of the following approaches.

- **Access the Amazon Aurora DB cluster inside the VPC** – Enable access to the Amazon Aurora DB cluster through the VPC. You do so by editing the Inbound rules on the Security group for the VPC to allow access to your specific Aurora DB cluster. To learn more, including how to configure your VPC for different Aurora DB cluster scenarios, see [Amazon Virtual Private Cloud VPCs and Amazon Aurora](#).
- **Access the Amazon Aurora DB cluster outside the VPC** – To access an Amazon Aurora DB cluster from outside the VPC, use the public endpoint address of the Amazon Aurora DB cluster. You can also connect to an Amazon Aurora DB cluster that's inside a VPC from an Amazon EC2 instance that's not in the VPC by using ClassicLink. For more information, see [A DB instance in a VPC accessed by an EC2 instance not in a VPC \(p. 1803\)](#).

For more information, see [Troubleshooting Aurora connection failures \(p. 287\)](#).

Topics

- [Connecting to an Amazon Aurora MySQL DB cluster \(p. 281\)](#)
- [Connecting to an Amazon Aurora PostgreSQL DB cluster \(p. 285\)](#)
- [Troubleshooting Aurora connection failures \(p. 287\)](#)

Connecting to an Amazon Aurora MySQL DB cluster

To authenticate to your Aurora MySQL DB cluster, you can use either MySQL user name and password authentication or AWS Identity and Access Management (IAM) database authentication. For more information on using MySQL user name and password authentication, see [Access control and account management](#) in the MySQL documentation. For more information on using IAM database authentication, see [IAM database authentication \(p. 1743\)](#).

When you have a connection to your Amazon Aurora DB cluster with MySQL 8.0 compatibility, you can run SQL commands that are compatible with MySQL version 8.0. The minimum compatible version is MySQL 8.0.23. For more information about MySQL 8.0 SQL syntax, see the [MySQL 8.0 reference manual](#). For information about limitations that apply to Aurora MySQL version 3, see [Comparison of Aurora MySQL version 3 and community MySQL 8.0 \(p. 756\)](#).

When you have a connection to your Amazon Aurora DB cluster with MySQL 5.7 compatibility, you can run SQL commands that are compatible with MySQL version 5.7. For more information about MySQL 5.7 SQL syntax, see the [MySQL 5.7 reference manual](#). For information about limitations that apply to Aurora MySQL 5.7, see [Aurora MySQL version 2 compatible with MySQL 5.7 \(p. 773\)](#).

When you have a connection to your Amazon Aurora DB cluster with MySQL 5.6 compatibility, you can run SQL commands that are compatible with MySQL version 5.6. For more information about MySQL 5.6 SQL syntax, see the [MySQL 5.6 reference manual](#).

Note

For a helpful and detailed guide on connecting to an Amazon Aurora MySQL DB cluster, you can see the [Aurora connection management](#) handbook.

In the details view for your DB cluster, you can find the cluster endpoint, which you can use in your MySQL connection string. The endpoint is made up of the domain name and port for your DB cluster. For example, if an endpoint value is `mycluster.cluster-123456789012.us-east-1.rds.amazonaws.com:3306`, then you specify the following values in a MySQL connection string:

- For host or host name, specify `mycluster.cluster-123456789012.us-east-1.rds.amazonaws.com`
- For port, specify 3306 or the port value you used when you created the DB cluster

The cluster endpoint connects you to the primary instance for the DB cluster. You can perform both read and write operations using the cluster endpoint. Your DB cluster can also have up to 15 Aurora Replicas that support read-only access to the data in your DB cluster. The primary instance and each Aurora Replica has a unique endpoint that is independent of the cluster endpoint and allows you to connect to a specific DB instance in the cluster directly. The cluster endpoint always points to the primary instance. If the primary instance fails and is replaced, then the cluster endpoint points to the new primary instance.

To view the cluster endpoint (writer endpoint), choose **Databases** on the Amazon RDS console and choose the name of the DB cluster to show the DB cluster details.

The screenshot shows the AWS RDS console for an Aurora MySQL cluster named "aurora-cl-mysql". The cluster has three instances: dbinstance4 (Writer), dbinstance1 (Reader), and dbinstance2 (Reader). The "Connectivity & security" tab is selected, showing two endpoints: "aurora-cl-mysql.cluster-ro" (Available, Reader) and "aurora-cl-mysql.cluster" (Available, Writer, port 3306), which is highlighted with a red border.

Endpoint name	Status	Type	Port
aurora-cl-mysql.cluster-ro.us-east-1.rds.amazonaws.com	Available	Reader	3306
aurora-cl-mysql.cluster.us-east-1.rds.amazonaws.com	Available	Writer	3306

Connection utilities for Aurora MySQL

Some connection utilities you can use are the following:

- **Command line** – You can connect to an Amazon Aurora DB cluster by using tools like the MySQL command line utility. For more information on using the MySQL utility, see [mysql - the MySQL command line tool](#) in the MySQL documentation.
- **GUI** – You can use the MySQL Workbench utility to connect by using a UI interface. For more information, see the [Download MySQL workbench](#) page.
- **Applications** – You can use the MariaDB Connector/J utility to connect your applications to your Aurora DB cluster. For more information, see the [MariaDB Connector/J download](#) page.

Note

If you use the MariaDB Connector/J utility with an Aurora Serverless v1 DB cluster, use the prefix `jdbc:mariadb:aurora://` in your connection string. The `mariadb:aurora` parameter avoids the automatic DNS scan for failover targets. That scanning is not needed with Aurora Serverless v1 DB clusters and causes a delay in establishing the connection.

You can use SSL encryption on connections to an Aurora MySQL DB instance. For information, see [Using SSL/TLS with Aurora MySQL DB clusters \(p. 775\)](#).

Connecting with SSL for Aurora MySQL

To connect using SSL, use the MySQL utility as described in the following procedure. If you are using IAM database authentication, you must use an SSL connection. For information, see [IAM database authentication \(p. 1743\)](#).

Note

To connect to the cluster endpoint using SSL, your client connection utility must support Subject Alternative Names (SAN). If your client connection utility doesn't support SAN, you can connect directly to the instances in your Aurora DB cluster. For more information on Aurora endpoints, see [Amazon Aurora connection management \(p. 32\)](#).

To connect to a DB cluster with SSL using the MySQL utility

1. Download the public key for the Amazon RDS signing certificate.

For information about downloading certificates, see [Using SSL/TLS to encrypt a connection to a DB cluster \(p. 1714\)](#).

2. Type the following command at a command prompt to connect to the primary instance of a DB cluster with SSL using the MySQL utility. For the `-h` parameter, substitute the endpoint DNS name for your primary instance. For the `-u` parameter, substitute the user ID of a database user account. For the `--ssl-ca` parameter, substitute the SSL certificate file name as appropriate. Type the master user password when prompted.

```
mysql -h mycluster-primary.123456789012.us-east-1.rds.amazonaws.com -u
admin_user -p --ssl-ca=[full path]rds-combined-ca-bundle.pem --ssl-verify-
server-cert
```

You should see output similar to the following.

```
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 350
Server version: 5.6.10-log MySQL Community Server (GPL)

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql>
```

For general instructions on constructing RDS for MySQL connection strings and finding the public key for SSL connections, see [Connecting to a DB instance running the MySQL database engine](#).

Connecting with the Amazon Web Services JDBC Driver for MySQL (preview)

This is preview documentation for Amazon Web Services JDBC Driver for MySQL. It is subject to change.

The AWS JDBC Driver for MySQL (preview) is a client driver designed for the high availability of Aurora MySQL. The AWS JDBC Driver for MySQL is drop-in compatible with the MySQL Connector/J driver.

The AWS JDBC Driver for MySQL takes full advantage of the failover capabilities of Aurora MySQL. The AWS JDBC Driver for MySQL fully maintains a cache of the DB cluster topology and each DB instance's

role, either primary DB instance or Aurora Replica. It uses this topology to bypass the delays caused by DNS resolution so that a connection to the new primary DB instance is established as fast as possible.

For more information about the AWS JDBC Driver for MySQL and complete instructions for using it, see the [AWS JDBC Driver for MySQL GitHub repository](#).

Connecting to an Amazon Aurora PostgreSQL DB cluster

You can connect to a DB instance in your Amazon Aurora PostgreSQL DB cluster using the same tools that you use to connect to a PostgreSQL database. As part of this, you use the same public key for Secure Sockets Layer (SSL) connections. You can use the endpoint and port information from the primary instance or Aurora Replicas in your Aurora PostgreSQL DB cluster in the connection string of any script, utility, or application that connects to a PostgreSQL DB instance. In the connection string, specify the DNS address from the primary instance or Aurora Replica endpoint as the host parameter. Specify the port number from the endpoint as the port parameter.

When you have a connection to a DB instance in your Amazon Aurora PostgreSQL DB cluster, you can run any SQL command that is compatible with PostgreSQL.

In the details view for your Aurora PostgreSQL DB cluster you can find the cluster endpoint name, status, type, and port number. You use the endpoint and port number in your PostgreSQL connection string. For example, if an endpoint value is `mycluster.cluster-123456789012.us-east-1.rds.amazonaws.com`, then you specify the following values in a PostgreSQL connection string:

- For host or host name, specify `mycluster.cluster-123456789012.us-east-1.rds.amazonaws.com`
- For port, specify 5432 or the port value you used when you created the DB cluster

The cluster endpoint connects you to the primary instance for the DB cluster. You can perform both read and write operations using the cluster endpoint. Your DB cluster can also have up to 15 Aurora Replicas that support read-only access to the data in your DB cluster. Each DB instance in the Aurora cluster (that is, the primary instance and each Aurora Replica) has a unique endpoint that is independent of the cluster endpoint. This unique endpoint allows you to connect to a specific DB instance in the cluster directly. The cluster endpoint always points to the primary instance. If the primary instance fails and is replaced, the cluster endpoint points to the new primary instance.

To view the cluster endpoint (writer endpoint), choose **Databases** on the Amazon RDS console and choose the name of the DB cluster to show the DB cluster details.

The screenshot shows the AWS RDS console interface for managing databases. The top navigation bar includes 'RDS' and 'Databases' with the specific database name 'aurora-cl-postgresql'. Below the navigation is the database name 'aurora-cl-postgresql' with 'Modify' and 'Actions' buttons. A 'Related' section with a search bar follows. The main content area displays a table for 'DB identifier' with columns: DB identifier, Role, Engine, Region & AZ, and Size. It lists three entries: 'aurora-cl-postgresql' (Regional, Aurora PostgreSQL, us-east-1, 2 instances), 'aurora-cl-postgresql-instance-1' (Writer, Aurora PostgreSQL, us-east-1a, db.r5.large), and 'aurora-cl-postgresql-instance-1-us-east-1b' (Reader, Aurora PostgreSQL, us-east-1b, db.r5.large). Below the table are tabs for 'Connectivity & security' (selected), 'Monitoring', 'Logs & events', 'Configuration', 'Maintenance & backups', and 'Tags'. The 'Endpoints (2)' section shows two endpoints: 'aurora-cl-postgresql.cluster-ro...' (us-east-1.rds.amazonaws.com) and 'aurora-cl-postgresql.cluster...' (us-east-1.rds.amazonaws.com). The second endpoint is highlighted with a red border. Buttons for 'Edit', 'Delete', and 'Create custom endpoint' are available. A 'Manage IAM roles' button is also present.

DB identifier	Role	Engine	Region & AZ	Size
aurora-cl-postgresql	Regional	Aurora PostgreSQL	us-east-1	2 instances
aurora-cl-postgresql-instance-1	Writer	Aurora PostgreSQL	us-east-1a	db.r5.large
aurora-cl-postgresql-instance-1-us-east-1b	Reader	Aurora PostgreSQL	us-east-1b	db.r5.large

Connection utilities for Aurora PostgreSQL

Some connection utilities you can use are the following:

- **Command line** – You can connect to an Amazon Aurora PostgreSQL DB instance by using tools like `psql`, the PostgreSQL interactive terminal. For more information on using the PostgreSQL interactive terminal, see [psql](#) in the PostgreSQL documentation.
- **GUI** – You can use the pgAdmin utility to connect to a PostgreSQL DB instance by using a UI interface. For more information, see the [Download](#) page from the pgAdmin website.
- **Applications** – You can use the PostgreSQL JDBC driver to connect your applications to your PostgreSQL DB instance. For more information, see the [Download](#) page from the PostgreSQL JDBC driver website.

Connecting with the Amazon Web Services JDBC Driver for PostgreSQL (preview)

This is preview documentation for Amazon Web Services JDBC Driver for PostgreSQL. It is subject to change.

The AWS JDBC Driver for PostgreSQL (preview) is a client driver designed for the high availability of Aurora PostgreSQL. The AWS JDBC Driver for PostgreSQL is drop-in compatible with the PostgreSQL JDBC Driver.

The AWS JDBC Driver for PostgreSQL takes full advantage of the failover capabilities of Aurora PostgreSQL. The AWS JDBC Driver for PostgreSQL fully maintains a cache of the DB cluster topology and each DB instance's role, either primary DB instance or Aurora Replica. It uses this topology to bypass the delays caused by DNS resolution so that a connection to the new primary DB instance is established as fast as possible.

For more information about the AWS JDBC Driver for PostgreSQL and complete instructions for using it, see the [AWS JDBC Driver for PostgreSQL GitHub repository](#).

Troubleshooting Aurora connection failures

Common causes of connection failures to a new Aurora DB cluster include the following:

- **Security group in the VPC doesn't allow access** – Your VPC needs to allow connections from your device or from an Amazon EC2 instance by proper configuration of the Security group in the VPC. To resolve, modify your VPC's Security group Inbound rules to allow connections. For an example, see [Create a VPC and subnets \(p. 1793\)](#).
- **Port blocked by firewall rules** – Check the value of the port configured for your Aurora DB cluster. If a firewall rule blocks that port, you can re-create the instance using a different port.
- **Incomplete or incorrect IAM configuration** – If you created your Aurora DB instance to use IAM-based authentication, make sure that it's properly configured. For more information, see [IAM database authentication \(p. 1743\)](#).

For more information about troubleshooting Aurora DB connection issues, see [Can't connect to Amazon RDS DB instance \(p. 1814\)](#).

Using Amazon RDS Proxy

By using Amazon RDS Proxy, you can allow your applications to pool and share database connections to improve their ability to scale. RDS Proxy makes applications more resilient to database failures by automatically connecting to a standby DB instance while preserving application connections. By using RDS Proxy, you can also enforce AWS Identity and Access Management (IAM) authentication for databases, and securely store credentials in AWS Secrets Manager.

Note

RDS Proxy is fully compatible with MySQL and PostgreSQL. You can enable RDS Proxy for most applications with no code changes.

Using RDS Proxy, you can handle unpredictable surges in database traffic that otherwise might cause issues due to oversubscribing connections or creating new connections at a fast rate. RDS Proxy establishes a database connection pool and reuses connections in this pool without the memory and CPU overhead of opening a new database connection each time. To protect the database against oversubscription, you can control the number of database connections that are created.

RDS Proxy queues or throttles application connections that can't be served immediately from the pool of connections. Although latencies might increase, your application can continue to scale without abruptly failing or overwhelming the database. If connection requests exceed the limits you specify, RDS Proxy rejects application connections (that is, it sheds load). At the same time, it maintains predictable performance for the load that can be served with the available capacity.

You can reduce the overhead to process credentials and establish a secure connection for each new connection. RDS Proxy can handle some of that work on behalf of the database.

Topics

- [Supported engines and Region availability for RDS Proxy \(p. 288\)](#)
- [Quotas and limitations for RDS Proxy \(p. 288\)](#)
- [Planning where to use RDS Proxy \(p. 290\)](#)
- [RDS Proxy concepts and terminology \(p. 290\)](#)
- [Getting started with RDS Proxy \(p. 295\)](#)
- [Managing an RDS Proxy \(p. 306\)](#)
- [Working with Amazon RDS Proxy endpoints \(p. 315\)](#)
- [Monitoring RDS Proxy metrics with Amazon CloudWatch \(p. 324\)](#)
- [Working with RDS Proxy events \(p. 329\)](#)
- [RDS Proxy command-line examples \(p. 330\)](#)
- [Troubleshooting for RDS Proxy \(p. 332\)](#)
- [Using RDS Proxy with AWS CloudFormation \(p. 337\)](#)

Supported engines and Region availability for RDS Proxy

For information about database engine version support and availability of RDS Proxy in a given AWS Region, see [Amazon RDS Proxy](#).

Quotas and limitations for RDS Proxy

The following quotas and limitations apply to RDS Proxy:

- You can have up to 20 proxies for each AWS account ID. If your application requires more proxies, you can request additional proxies by opening a ticket with the AWS Support organization.
- Each proxy can have up to 200 associated Secrets Manager secrets. Thus, each proxy can connect to up to 200 different user accounts at any given time.
- You can create, view, modify, and delete up to 20 endpoints for each proxy. These endpoints are in addition to the default endpoint that's automatically created for each proxy.
- In an Aurora cluster, all of the connections using the default proxy endpoint are handled by the Aurora writer instance. To perform load balancing for read-intensive workloads, you can create a read-only endpoint for a proxy. That endpoint passes connections to the reader endpoint of the cluster. That way, your proxy connections can take advantage of Aurora read scalability. For more information, see [Overview of proxy endpoints \(p. 315\)](#).

For RDS DB instances in replication configurations, you can associate a proxy only with the writer DB instance, not a read replica.

- You can't use RDS Proxy with Aurora Serverless clusters.
- Using RDS Proxy with Aurora clusters that are part of an Aurora global database isn't currently supported.
- Your RDS Proxy must be in the same virtual private cloud (VPC) as the database. The proxy can't be publicly accessible, although the database can be. For example, if you're prototyping on a local host, you can't connect to your RDS Proxy unless you set up dedicated networking. This is the case because your local host is outside of the proxy's VPC.

Note

For Aurora DB clusters, you can turn on cross-VPC access. To do this, create an additional endpoint for a proxy and specify a different VPC, subnets, and security groups with that endpoint. For more information, see [Accessing Aurora and RDS databases across VPCs \(p. 319\)](#).

- You can't use RDS Proxy with a VPC that has its tenancy set to dedicated.
- If you use RDS Proxy with an RDS DB instance or Aurora DB cluster that has IAM authentication enabled, make sure that all users who connect through a proxy authenticate through user names and passwords. See [Setting up AWS Identity and Access Management \(IAM\) policies \(p. 297\)](#) for details about IAM support in RDS Proxy.
- You can't use RDS Proxy with custom DNS.
- RDS Proxy is available for the MySQL and PostgreSQL engine families.
- Each proxy can be associated with a single target DB instance or cluster. However, you can associate multiple proxies with the same DB instance or cluster.

The following RDS Proxy limitations apply to MySQL:

- RDS Proxy doesn't support the MySQL sha256_password and caching_sha2_password authentication plugins. These plugins implement SHA-256 hashing for user account passwords.
- Currently, all proxies listen on port 3306 for MySQL. The proxies still connect to your database using the port that you specified in the database settings.
- You can't use RDS Proxy with self-managed MySQL databases in EC2 instances.
- You can't use RDS Proxy with an RDS for MySQL DB instance that has the read_only parameter in its DB parameter group set to 1.
- Proxies don't support MySQL compressed mode. For example, they don't support the compression used by the --compress or -C options of the mysql command.
- Some SQL statements and functions can change the connection state without causing pinning. For the most current pinning behavior, see [Avoiding pinning \(p. 312\)](#).

The following RDS Proxy limitations apply to PostgreSQL:

- Currently, all proxies listen on port 5432 for PostgreSQL.
- For PostgreSQL, RDS Proxy doesn't currently support canceling a query from a client by issuing a `CancelRequest`. This is the case for example, when you cancel a long-running query in an interactive `psql` session by using `Ctrl+C`.
- The results of the PostgreSQL function `lastval` aren't always accurate. As a work-around, use the `INSERT` statement with the `RETURNING` clause.
- RDS Proxy doesn't multiplex connections when your client application drivers use the PostgreSQL extended query protocol.

Planning where to use RDS Proxy

You can determine which of your DB instances, clusters, and applications might benefit the most from using RDS Proxy. To do so, consider these factors:

- Any DB instance or cluster that encounters "too many connections" errors is a good candidate for associating with a proxy. The proxy enables applications to open many client connections, while the proxy manages a smaller number of long-lived connections to the DB instance or cluster.
- For DB instances or clusters that use smaller AWS instance classes, such as T2 or T3, using a proxy can help avoid out-of-memory conditions. It can also help reduce the CPU overhead for establishing connections. These conditions can occur when dealing with large numbers of connections.
- You can monitor certain Amazon CloudWatch metrics to determine whether a DB instance or cluster is approaching certain types of limit. These limits are for the number of connections and the memory associated with connection management. You can also monitor certain CloudWatch metrics to determine whether a DB instance or cluster is handling many short-lived connections. Opening and closing such connections can impose performance overhead on your database. For information about the metrics to monitor, see [Monitoring RDS Proxy metrics with Amazon CloudWatch \(p. 324\)](#).
- AWS Lambda functions can also be good candidates for using a proxy. These functions make frequent short database connections that benefit from connection pooling offered by RDS Proxy. You can take advantage of any IAM authentication you already have for Lambda functions, instead of managing database credentials in your Lambda application code.
- Applications that use languages and frameworks such as PHP and Ruby on Rails are typically good candidates for using a proxy. Such applications typically open and close large numbers of database connections, and don't have built-in connection pooling mechanisms.
- Applications that keep a large number of connections open for long periods are typically good candidates for using a proxy. Applications in industries such as software as a service (SaaS) or ecommerce often minimize the latency for database requests by leaving connections open. With RDS Proxy, an application can keep more connections open than it can when connecting directly to the DB instance or cluster.
- You might not have adopted IAM authentication and Secrets Manager due to the complexity of setting up such authentication for all DB instances and clusters. If so, you can leave the existing authentication methods in place and delegate the authentication to a proxy. The proxy can enforce the authentication policies for client connections for particular applications. You can take advantage of any IAM authentication you already have for Lambda functions, instead of managing database credentials in your Lambda application code.
- RDS Proxy is highly available and deployed over multiple Availability Zones (AZs). To ensure overall high availability for your database, deploy your Amazon RDS DB instance or Aurora cluster in a Multi-AZ configuration.

RDS Proxy concepts and terminology

You can simplify connection management for your Amazon RDS DB instances and Amazon Aurora DB clusters by using RDS Proxy.

RDS Proxy handles the network traffic between the client application and the database. It does so in an active way first by understanding the database protocol. It then adjusts its behavior based on the SQL operations from your application and the result sets from the database.

RDS Proxy reduces the memory and CPU overhead for connection management on your database. The database needs less memory and CPU resources when applications open many simultaneous connections. It also doesn't require logic in your applications to close and reopen connections that stay idle for a long time. Similarly, it requires less application logic to reestablish connections in case of a database problem.

The infrastructure for RDS Proxy is highly available and deployed over multiple Availability Zones (AZs). The computation, memory, and storage for RDS Proxy are independent of your RDS DB instances and Aurora DB clusters. This separation helps lower overhead on your database servers, so that they can devote their resources to serving database workloads. The RDS Proxy compute resources are serverless, automatically scaling based on your database workload.

Topics

- [Overview of RDS Proxy concepts \(p. 291\)](#)
- [Connection pooling \(p. 292\)](#)
- [RDS Proxy security \(p. 292\)](#)
- [Failover \(p. 293\)](#)
- [Transactions \(p. 294\)](#)

Overview of RDS Proxy concepts

RDS Proxy handles the infrastructure to perform connection pooling and the other features described in the sections that follow. You see the proxies represented in the RDS console on the [Proxies](#) page.

Each proxy handles connections to a single RDS DB instance or Aurora DB cluster. The proxy automatically determines the current writer instance for RDS Multi-AZ DB instances and Aurora provisioned clusters. For Aurora multi-master clusters, the proxy connects to one of the writer instances and uses the other writer instances as hot standby targets.

The connections that a proxy keeps open and available for your database application to use make up the *connection pool*.

By default, RDS Proxy can reuse a connection after each transaction in your session. This transaction-level reuse is called *multiplexing*. When RDS Proxy temporarily removes a connection from the connection pool to reuse it, that operation is called *borrowing* the connection. When it's safe to do so, RDS Proxy returns that connection to the connection pool.

In some cases, RDS Proxy can't be sure that it's safe to reuse a database connection outside of the current session. In these cases, it keeps the session on the same connection until the session ends. This fallback behavior is called *pinning*.

A proxy has a default endpoint. You connect to this endpoint when you work with an RDS DB instance or Aurora DB cluster, instead of connecting to the read/write endpoint that connects directly to the instance or cluster. The special-purpose endpoints for an Aurora cluster remain available for you to use. For Aurora DB clusters, you can also create additional read/write and read-only endpoints. For more information, see [Overview of proxy endpoints \(p. 315\)](#).

For example, you can still connect to the cluster endpoint for read/write connections without connection pooling. You can still connect to the reader endpoint for load-balanced read-only connections. You can still connect to the instance endpoints for diagnosis and troubleshooting of specific DB instances within an Aurora cluster. If you are using other AWS services such as AWS Lambda to connect to RDS databases, you change their connection settings to use the proxy endpoint. For example, you specify the proxy endpoint to allow Lambda functions to access your database while taking advantage of RDS Proxy functionality.

Each proxy contains a target group. This *target group* embodies the RDS DB instance or Aurora DB cluster that the proxy can connect to. For an Aurora cluster, by default the target group is associated with all the DB instances in that cluster. That way, the proxy can connect to whichever Aurora DB instance is promoted to be the writer instance in the cluster. The RDS DB instance associated with a proxy, or the Aurora DB cluster and its instances, are called the *targets* of that proxy. For convenience, when you create a proxy through the console, RDS Proxy also creates the corresponding target group and registers the associated targets automatically.

An *engine family* is a related set of database engines that use the same DB protocol. You choose the engine family for each proxy that you create.

Connection pooling

Each proxy performs connection pooling for the writer instance of its associated RDS or Aurora database. *Connection pooling* is an optimization that reduces the overhead associated with opening and closing connections and with keeping many connections open simultaneously. This overhead includes memory needed to handle each new connection. It also involves CPU overhead to close each connection and open a new one, such as Transport Layer Security/Secure Sockets Layer (TLS/SSL) handshaking, authentication, negotiating capabilities, and so on. Connection pooling simplifies your application logic. You don't need to write application code to minimize the number of simultaneous open connections.

Each proxy also performs connection multiplexing, also known as connection reuse. With *multiplexing*, RDS Proxy performs all the operations for a transaction using one underlying database connection, then can use a different connection for the next transaction. You can open many simultaneous connections to the proxy, and the proxy keeps a smaller number of connections open to the DB instance or cluster. Doing so further minimizes the memory overhead for connections on the database server. This technique also reduces the chance of "too many connections" errors.

RDS Proxy security

RDS Proxy uses the existing RDS security mechanisms such as TLS/SSL and AWS Identity and Access Management (IAM). For general information about those security features, see [Security in Amazon Aurora \(p. 1706\)](#). If you aren't familiar with how RDS and Aurora work with authentication, authorization, and other areas of security, make sure to familiarize yourself with how RDS and Aurora work with those areas first.

RDS Proxy can act as an additional layer of security between client applications and the underlying database. For example, you can connect to the proxy using TLS 1.2, even if the underlying DB instance supports only TLS 1.0 or 1.1. You can connect to the proxy using an IAM role, even if the proxy connects to the database using the native user and password authentication method. By using this technique, you can enforce strong authentication requirements for database applications without a costly migration effort for the DB instances themselves.

You store the database credentials used by RDS Proxy in AWS Secrets Manager. Each database user for the RDS DB instance or Aurora DB cluster accessed by a proxy must have a corresponding secret in Secrets Manager. You can also set up IAM authentication for users of RDS Proxy. By doing so, you can enforce IAM authentication for database access even if the databases use native password authentication. We recommend using these security features instead of embedding database credentials in your application code.

Using TLS/SSL with RDS Proxy

You can connect to RDS Proxy using the TLS/SSL protocol.

Note

RDS Proxy uses certificates from the AWS Certificate Manager (ACM). If you use RDS Proxy, when you rotate your TLS/SSL certificate you don't need to update applications that use RDS Proxy connections.

To enforce TLS for all connections between the proxy and your database, you can specify a setting **Require Transport Layer Security** when you create or modify a proxy.

RDS Proxy can also ensure that your session uses TLS/SSL between your client and the RDS Proxy endpoint. To have RDS Proxy do so, specify the requirement on the client side. SSL session variables are not set for SSL connections to a database using RDS Proxy.

- For RDS for MySQL and Aurora MySQL, specify the requirement on the client side with the `--ssl-mode` parameter when you run the `mysql` command.
- For Amazon RDS PostgreSQL and Aurora PostgreSQL, specify `sslmode=require` as part of the `conninfo` string when you run the `psql` command.

RDS Proxy supports TLS protocol version 1.0, 1.1, and 1.2. You can connect to the proxy using a higher version of TLS than you use in the underlying database.

By default, client programs establish an encrypted connection with RDS Proxy, with further control available through the `--ssl-mode` option. From the client side, RDS Proxy supports all SSL modes.

For the client, the SSL modes are the following:

PREFERRED

SSL is the first choice, but it isn't required.

DISABLED

No SSL is allowed.

REQUIRED

Enforce SSL.

VERIFY_CA

Enforce SSL and verify the certificate authority (CA).

VERIFY_IDENTITY

Enforce SSL and verify the CA and CA hostname.

Note

You can use the SSL mode `VERIFY_IDENTITY` when connecting to the default proxy endpoint. You can't use that SSL mode when you connect to proxy endpoints that you create.

When using a client with `--ssl-mode VERIFY_CA` or `VERIFY_IDENTITY`, specify the `--ssl-ca` option pointing to a CA in `.pem` format. For a `.pem` file that you can use, download the [Amazon root CA 1 trust store](#) from Amazon Trust Services.

RDS Proxy uses wildcard certificates, which apply to both a domain and its subdomains. If you use the `mysql` client to connect with SSL mode `VERIFY_IDENTITY`, currently you must use the MySQL 8.0-compatible `mysql` command.

Failover

Failover is a high-availability feature that replaces a database instance with another one when the original instance becomes unavailable. A failover might happen because of a problem with a database instance. It might also be part of normal maintenance procedures, such as during a database upgrade. Failover applies to RDS DB instances in a Multi-AZ configuration, and Aurora DB clusters with one or more reader instances in addition to the writer instance.

Connecting through a proxy makes your application more resilient to database failovers. When the original DB instance becomes unavailable, RDS Proxy connects to the standby database without dropping idle application connections. Doing so helps to speed up and simplify the failover process. The result is faster failover that's less disruptive to your application than a typical reboot or database problem.

Without RDS Proxy, a failover involves a brief outage. During the outage, you can't perform write operations on that database. Any existing database connections are disrupted and your application must reopen them. The database becomes available for new connections and write operations when a read-only DB instance is promoted to take the place of the one that's unavailable.

During DB failovers, RDS Proxy continues to accept connections at the same IP address and automatically directs connections to the new primary DB instance. Clients connecting through RDS Proxy are not susceptible to the following:

- Domain Name System (DNS) propagation delays on failover.
- Local DNS caching.
- Connection timeouts.
- Uncertainty about which DB instance is the current writer.
- Waiting for a query response from a former writer that became unavailable without closing connections.

For applications that maintain their own connection pool, going through RDS Proxy means that most connections stay alive during failovers or other disruptions. Only connections that are in the middle of a transaction or SQL statement are canceled. RDS Proxy immediately accepts new connections. When the database writer is unavailable, RDS Proxy queues up incoming requests.

For applications that don't maintain their own connection pools, RDS Proxy offers faster connection rates and more open connections. It offloads the expensive overhead of frequent reconnects from the database. It does so by reusing database connections maintained in the RDS Proxy connection pool. This approach is particularly important for TLS connections, where setup costs are significant.

Transactions

All the statements within a single transaction always use the same underlying database connection. The connection becomes available for use by a different session when the transaction ends. Using the transaction as the unit of granularity has the following consequences:

- Connection reuse can happen after each individual statement when the RDS for MySQL or Aurora MySQL `autocommit` setting is enabled.
- Conversely, when the `autocommit` setting is disabled, the first statement you issue in a session begins a new transaction. Thus, if you enter a sequence of `SELECT`, `INSERT`, `UPDATE`, and other data manipulation language (DML) statements, connection reuse doesn't happen until you issue a `COMMIT`, `ROLLBACK`, or otherwise end the transaction.
- Entering a data definition language (DDL) statement causes the transaction to end after that statement completes.

RDS Proxy detects when a transaction ends through the network protocol used by the database client application. Transaction detection doesn't rely on keywords such as `COMMIT` or `ROLLBACK` appearing in the text of the SQL statement.

In some cases, RDS Proxy might detect a database request that makes it impractical to move your session to a different connection. In these cases, it turns off multiplexing for that connection the remainder of your session. The same rule applies if RDS Proxy can't be certain that multiplexing is practical for the session. This operation is called *pinning*. For ways to detect and minimize pinning, see [Avoiding pinning \(p. 312\)](#).

Getting started with RDS Proxy

In the following sections, you can find how to set up RDS Proxy. You can also find how to set the related security options that control who can access each proxy and how each proxy connects to DB instances.

Topics

- [Setting up network prerequisites \(p. 295\)](#)
- [Setting up database credentials in AWS Secrets Manager \(p. 296\)](#)
- [Setting up AWS Identity and Access Management \(IAM\) policies \(p. 297\)](#)
- [Creating an RDS Proxy \(p. 299\)](#)
- [Viewing an RDS Proxy \(p. 302\)](#)
- [Connecting to a database through RDS Proxy \(p. 304\)](#)

Setting up network prerequisites

Using RDS Proxy requires you to have a common virtual private cloud (VPC) between your Aurora DB cluster or RDS DB instance and RDS Proxy. This VPC should have a minimum of two subnets that are in different Availability Zones. Your account can either own these subnets or share them with other accounts. For information about VPC sharing, see [Work with shared VPCs](#). Your client application resources such as Amazon EC2, Lambda, or Amazon ECS can be in the same VPC or in a separate VPC from the proxy. Note that if you've successfully connected to any RDS DB instances or Aurora DB clusters, you already have the required network resources.

If you're just getting started with RDS or Aurora, you can learn the basics of connecting to a database by following the procedures in [Setting up your environment for Amazon Aurora \(p. 84\)](#). You can also follow the tutorial in [Getting started with Amazon Aurora \(p. 89\)](#).

The following Linux example shows AWS CLI commands that examine the VPCs and subnets owned by your AWS account. In particular, you pass subnet IDs as parameters when you create a proxy using the CLI.

```
aws ec2 describe-vpcs
aws ec2 describe-internet-gateways
aws ec2 describe-subnets --query '*[].[VpcId,SubnetId]' --output text | sort
```

The following Linux example shows AWS CLI commands to determine the subnet IDs corresponding to a specific Aurora DB cluster or RDS DB instance. For an Aurora cluster, first you find the ID for one of the associated DB instances. You can extract the subnet IDs used by that DB instance by examining the nested fields within the `DBSubnetGroup` and `Subnets` attributes in the describe output for the DB instance. You specify some or all of those subnet IDs when setting up a proxy for that database server.

```
$ # Optional first step, only needed if you're starting from an Aurora cluster. Find the ID
of any DB instance in the cluster.
$ aws rds describe-db-clusters --db-cluster-identifier my_cluster_id --query '*[].[DBClusterMembers][0|[0|[*].DBInstanceIdentifier' --output text
my_instance_id
instance_id_2
instance_id_3
...
$ # From the DB instance, trace through the DBSubnetGroup and Subnets to find the subnet
IDs.
$ aws rds describe-db-instances --db-instance-identifier my_instance_id --query '*[].[DBSubnetGroup][0|[0|[Subnets][0|[*].SubnetIdentifier' --output text
```

```
subnet_id_1
subnet_id_2
subnet_id_3
...
```

As an alternative, you can first find the VPC ID for the DB instance. Then you can examine the VPC to find its subnets. The following Linux example shows how.

```
$ # From the DB instance, find the VPC.
$ aws rds describe-db-instances --db-instance-identifier my_instance_id --query '*[].[DBSubnetGroup|[0]|[]].VpcId' --output text
my_vpc_id

$ aws ec2 describe-subnets --filters Name=vpc-id,Values=my_vpc_id --query '*[].[SubnetId]' --output text
subnet_id_1
subnet_id_2
subnet_id_3
subnet_id_4
subnet_id_5
subnet_id_6
```

Setting up database credentials in AWS Secrets Manager

For each proxy that you create, you first use the Secrets Manager service to store sets of user name and password credentials. You create a separate Secrets Manager secret for each database user account that the proxy connects to on the RDS DB instance or Aurora DB cluster.

In Secrets Manager, you create these secrets with values for the **username** and **password** fields. Doing so allows the proxy to connect to the corresponding database users on whichever RDS DB instances or Aurora DB clusters that you associate with the proxy. To do this, you can use the setting **Credentials for other database**, **Credentials for RDS database**, or **Other type of secrets**. Fill in the appropriate values for the **User name** and **Password** fields, and placeholder values for any other required fields. The proxy ignores other fields such as **Host** and **Port** if they're present in the secret. Those details are automatically supplied by the proxy.

You can also choose **Other type of secrets**. In this case, you create the secret with keys named **username** and **password**.

Because the secrets used by your proxy aren't tied to a specific database server, you can reuse a secret across multiple proxies if you use the same credentials across multiple database servers. For example, you might use the same credentials across a group of development and test servers.

To connect through the proxy as a specific user, make sure that the password associated with a secret matches the database password for that user. If there's a mismatch, you can update the associated secret in Secrets Manager. In this case, you can still connect to other accounts where the secret credentials and the database passwords do match.

When you create a proxy through the AWS CLI or RDS API, you specify the Amazon Resource Names (ARNs) of the corresponding secrets for all the DB user accounts that the proxy can access. In the AWS Management Console, you choose the secrets by their descriptive names.

For instructions about creating secrets in Secrets Manager, see the [Creating a secret](#) page in the Secrets Manager documentation. Use one of the following techniques:

- Use [Secrets Manager](#) in the console.
- To use the CLI to create a Secrets Manager secret for use with RDS Proxy, use a command such as the following.

```
aws secretsmanager create-secret \
--name "secret_name"
--description "secret_description"
--region region_name
--secret-string '{"username":"db_user","password":"db_user_password"}'
```

For example, the following commands create Secrets Manager secrets for two database users, one named admin and the other named app-user.

```
aws secretsmanager create-secret \
--name admin_secret_name --description "db admin user" \
--secret-string '{"username":"admin","password":"choose_your_own_password"}'

aws secretsmanager create-secret \
--name proxy_secret_name --description "application user" \
--secret-string '{"username":"app-user","password":"choose_your_own_password"}'
```

To see the secrets owned by your AWS account, use a command such as the following.

```
aws secretsmanager list-secrets
```

When you create a proxy using the CLI, you pass the Amazon Resource Names (ARNs) of one or more secrets to the --auth parameter. The following Linux example shows how to prepare a report with only the name and ARN of each secret owned by your AWS account. This example uses the --output table parameter that is available in AWS CLI version 2. If you are using AWS CLI version 1, use --output text instead.

```
aws secretsmanager list-secrets --query '*[].[Name,ARN]' --output table
```

To verify that you stored the correct credentials and in the right format in a secret, use a command such as the following. Substitute the short name or the ARN of the secret for *your_secret_name*.

```
aws secretsmanager get-secret-value --secret-id your_secret_name
```

The output should include a line displaying a JSON-encoded value like the following.

```
"SecretString": "{\"username\":\"your_username\",\"password\":\"your_password\"}",
```

Setting up AWS Identity and Access Management (IAM) policies

After you create the secrets in Secrets Manager, you create an IAM policy that can access those secrets. For general information about using IAM with RDS and Aurora, see [Identity and access management in Amazon Aurora \(p. 1724\)](#).

Tip

The following procedure applies if you use the IAM console. If you use the AWS Management Console for RDS, RDS can create the IAM policy for you automatically. In that case, you can skip the following procedure.

To create an IAM policy that accesses your Secrets Manager secrets for use with your proxy

1. Sign in to the IAM console. Follow the [Create role](#) process, as described in [Creating IAM roles](#). Include the [Add Role to Database](#) step.
2. For the new role, perform the [Add inline policy](#) step. Use the same general procedures as in [Editing IAM policies](#). Paste the following JSON into the JSON text box. Substitute your own account ID.

Substitute your AWS Region for `us-east-2`. Substitute the Amazon Resource Names (ARNs) for the secrets that you created. For the `kms:Decrypt` action, substitute the ARN of the default AWS KMS key or your own KMS key depending on which one you used to encrypt the Secrets Manager secrets.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor0",
            "Effect": "Allow",
            "Action": "secretsmanager:GetSecretValue",
            "Resource": [
                "arn:aws:secretsmanager:us-east-2:account_id:secret:secret_name_1",
                "arn:aws:secretsmanager:us-east-2:account_id:secret:secret_name_2"
            ]
        },
        {
            "Sid": "VisualEditor1",
            "Effect": "Allow",
            "Action": "kms:Decrypt",
            "Resource": "arn:aws:kms:us-east-2:account_id:key/key_id",
            "Condition": {
                "StringEquals": {
                    "kms:ViaService": "secretsmanager.us-east-2.amazonaws.com"
                }
            }
        }
    ]
}
```

3. Edit the trust policy for this IAM role. Paste the following JSON into the JSON text box.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "",
            "Effect": "Allow",
            "Principal": {
                "Service": "rds.amazonaws.com"
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

The following commands perform the same operation through the AWS CLI.

```
PREFIX=choose_an_identifier

aws iam create-role --role-name choose_role_name \
--assume-role-policy-document '{"Version":"2012-10-17","Statement": [{"Effect":"Allow","Principal":{"Service": ["rds.amazonaws.com"]},"Action":"sts:AssumeRole"}]}'

aws iam put-role-policy --role-name same_role_name_as_previous \
--policy-name $PREFIX-secret-reader-policy --policy-document """
same_json_as_in_previous_example
"""

aws kms create-key --description "$PREFIX-test-key" --policy """
{
```

```

"Id": "$PREFIX-kms-policy",
"Version": "2012-10-17",
"Statement":
[
    {
        "Sid": "Enable IAM User Permissions",
        "Effect": "Allow",
        "Principal": {"AWS": "arn:aws:iam::account_id:root"},
        "Action": "kms:*", "Resource": "*"
    },
    {
        "Sid": "Allow access for Key Administrators",
        "Effect": "Allow",
        "Principal": {
            "AWS": [
                "$USER_ARN", "arn:aws:iam::account_id:role/Admin"
            ],
            "Action": [
                "kms>Create*",
                "kms>Describe*",
                "kms>Enable*",
                "kms>List*",
                "kms>Put*",
                "kms>Update*",
                "kms>Revoke*",
                "kms>Disable*",
                "kms>Get*",
                "kms>Delete*",
                "kms>TagResource",
                "kms>UntagResource",
                "kms>ScheduleKeyDeletion",
                "kms>CancelKeyDeletion"
            ],
            "Resource": "*"
        },
        {
            "Sid": "Allow use of the key",
            "Effect": "Allow",
            "Principal": {"AWS": "$ROLE_ARN"},
            "Action": ["kms>Decrypt", "kms>DescribeKey"],
            "Resource": "*"
        }
    }
]
"""

```

Creating an RDS Proxy

To manage connections for a specified set of DB instances, you can create a proxy. You can associate a proxy with an RDS for MySQL DB instance, PostgreSQL DB instance, or an Aurora DB cluster.

AWS Management Console

To create a proxy

1. Sign in to the AWS Management Console and open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. In the navigation pane, choose **Proxies**.
3. Choose **Create proxy**.
4. Choose all the settings for your proxy.

For **Proxy configuration**, provide information for the following:

- **Proxy identifier.** Specify a name of your choosing, unique within your AWS account ID and current AWS Region.
- **Engine compatibility.** Choose either **MySQL** or **POSTGRESQL**.
- **Require Transport Layer Security.** Choose this setting if you want the proxy to enforce TLS/SSL for all client connections. When you use an encrypted or unencrypted connection to a proxy, the proxy uses the same encryption setting when it makes a connection to the underlying database.
- **Idle client connection timeout.** Choose a time period that a client connection can be idle before the proxy can close it. The default is 1,800 seconds (30 minutes). A client connection is considered idle when the application doesn't submit a new request within the specified time after the previous request completed. The underlying database connection stays open and is returned to the connection pool. Thus, it's available to be reused for new client connections.

Consider lowering the idle client connection timeout if you want the proxy to proactively remove stale connections. If your workload is spiking, consider raising the idle client connection timeout to save the cost of establishing connections.

For **Target group configuration**, provide information for the following:

- **Database.** Choose one RDS DB instance or Aurora DB cluster to access through this proxy. The list only includes DB instances and clusters with compatible database engines, engine versions, and other settings. If the list is empty, create a new DB instance or cluster that's compatible with RDS Proxy. To do so, follow the procedure in [Creating an Amazon Aurora DB cluster \(p. 125\)](#). Then try creating the proxy again.
- **Connection pool maximum connections.** Specify a value from 1 through 100. This setting represents the percentage of the `max_connections` value that RDS Proxy can use for its connections. If you only intend to use one proxy with this DB instance or cluster, you can set this value to 100. For details about how RDS Proxy uses this setting, see [MaxConnectionsPercent \(p. 311\)](#).
- **Session pinning filters.** (Optional) This is an advanced setting, for troubleshooting performance issues with particular applications. Currently, the only choice is `EXCLUDE_VARIABLE_SETS`. Choose a filter only if both of following are true: Your application isn't reusing connections due to certain kinds of SQL statements, and you can verify that reusing connections with those SQL statements doesn't affect application correctness. For more information, see [Avoiding pinning \(p. 312\)](#).
- **Connection borrow timeout.** In some cases, you might expect the proxy to sometimes use all available database connections. In such cases, you can specify how long the proxy waits for a database connection to become available before returning a timeout error. You can specify a period up to a maximum of five minutes. This setting only applies when the proxy has the maximum number of connections open and all connections are already in use.
- **Initialization query.** (Optional) You can specify one or more SQL statements for the proxy to run when opening each new database connection. The setting is typically used with `SET` statements to make sure that each connection has identical settings such as time zone and character set. For multiple statements, use semicolons as the separator. You can also include multiple variables in a single `SET` statement, such as `SET x=1, y=2`. Initialization query is not currently supported for PostgreSQL.

For **Connectivity**, provide information for the following:

- **Secrets Manager secrets.** Choose at least one Secrets Manager secret that contains DB user credentials for the RDS DB instance or Aurora DB cluster that you intend to access with this proxy.

- **IAM role.** Choose an IAM role that has permission to access the Secrets Manager secrets that you chose earlier. You can also choose for the AWS Management Console to create a new IAM role for you and use that.
- **IAM Authentication.** Choose whether to require or disallow IAM authentication for connections to your proxy. The choice of IAM authentication or native database authentication applies to all DB users that access this proxy.
- **Subnets.** This field is prepopulated with all the subnets associated with your VPC. You can remove any subnets that you don't need for this proxy. You must leave at least two subnets.

Provide additional connectivity configuration:

- **VPC security group.** Choose an existing VPC security group. You can also choose for the AWS Management Console to create a new security group for you and use that.

Note

This security group must allow access to the database the proxy connects to. The same security group is used for ingress from your applications to the proxy, and for egress from the proxy to the database. For example, suppose that you use the same security group for your database and your proxy. In this case, make sure that you specify that resources in that security group can communicate with other resources in the same security group. When using a shared VPC, you can't use the default security group for the VPC, or one that belongs to another account. Choose a security group that belongs to your account. If one doesn't exist, create one. For more information about this limitation, see [Work with shared VPCs](#).

(Optional) Provide advanced configuration:

- **Enable enhanced logging.** You can enable this setting to troubleshoot proxy compatibility or performance issues.

When this setting is enabled, RDS Proxy includes detailed information about SQL statements in its logs. This information helps you to debug issues involving SQL behavior or the performance and scalability of the proxy connections. The debug information includes the text of SQL statements that you submit through the proxy. Thus, only enable this setting when needed for debugging, and only when you have security measures in place to safeguard any sensitive information that appears in the logs.

To minimize overhead associated with your proxy, RDS Proxy automatically turns this setting off 24 hours after you enable it. Enable it temporarily to troubleshoot a specific issue.

5. Choose **Create Proxy**.

AWS CLI

To create a proxy, use the AWS CLI command [create-db-proxy](#). The --engine-family value is case-sensitive.

Example

For Linux, macOS, or Unix:

```
aws rds create-db-proxy \
--db-proxy-name proxy_name \
--engine-family { MYSQL | POSTGRESQL } \
--auth ProxyAuthenticationConfig_JSON_string \
--role-arn iam_role \
--vpc-subnet-ids space_separated_list \
```

```
[--vpc-security-group-ids space-separated_list] \
[--require-tls | --no-require-tls] \
[--idle-client-timeout value] \
[--debug-logging | --no-debug-logging] \
[--tags comma-separated_list]
```

For Windows:

```
aws rds create-db-proxy ^
--db-proxy-name proxy_name ^
--engine-family { MYSQL | POSTGRESOL } ^
--auth ProxyAuthenticationConfig_JSON_string ^
--role-arn iam_role ^
--vpc-subnet-ids space-separated_list ^
[--vpc-security-group-ids space-separated_list] ^
[--require-tls | --no-require-tls] ^
[--idle-client-timeout value] ^
[--debug-logging | --no-debug-logging] ^
[--tags comma-separated_list]
```

Tip

If you don't already know the subnet IDs to use for the `--vpc-subnet-ids` parameter, see [Setting up network prerequisites \(p. 295\)](#) for examples of how to find the subnet IDs that you can use.

Note

The security group must allow access to the database the proxy connects to. The same security group is used for ingress from your applications to the proxy, and for egress from the proxy to the database. For example, suppose that you use the same security group for your database and your proxy. In this case, make sure that you specify that resources in that security group can communicate with other resources in the same security group.

When using a shared VPC, you can't use the default security group for the VPC, or one that belongs to another account. Choose a security group that belongs to your account. If one doesn't exist, create one. For more information about this limitation, see [Work with shared VPCs](#).

To create the required information and associations for the proxy, you also use the `register-db-proxy-targets` command. Specify the target group name `default`. RDS Proxy automatically creates a target group with this name when you create each proxy.

```
aws rds register-db-proxy-targets
--db-proxy-name value
[--target-group-name target_group_name]
[--db-instance-identifiers space-separated_list] # rds db instances, or
[--db-cluster-identifiers cluster_id]           # rds db cluster (all instances), or
[--db-cluster-endpoint endpoint_name]          # rds db cluster endpoint (all
instances)
```

RDS API

To create an RDS proxy, call the Amazon RDS API operation [CreateDBProxy](#). You pass a parameter with the `AuthConfig` data structure.

RDS Proxy automatically creates a target group named `default` when you create each proxy. You associate an RDS DB instance or Aurora DB cluster with the target group by calling the function [RegisterDBProxyTargets](#).

Viewing an RDS Proxy

After you create one or more RDS proxies, you can view them all to examine their configuration details and choose which ones to modify, delete, and so on.

Any database applications that use the proxy require the proxy endpoint to use in the connection string.

AWS Management Console

To view your proxy

1. Sign in to the AWS Management Console and open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. In the upper-right corner of the AWS Management Console, choose the AWS Region in which you created the RDS Proxy.
3. In the navigation pane, choose **Proxies**.
4. Choose the name of an RDS proxy to display its details.
5. On the details page, the **Target groups** section shows how the proxy is associated with a specific RDS DB instance or Aurora DB cluster. You can follow the link to the **default** target group page to see more details about the association between the proxy and the database. This page is where you see settings that you specified when creating the proxy, such as maximum connection percentage, connection borrow timeout, engine compatibility, and session pinning filters.

CLI

To view your proxy using the CLI, use the `describe-db-proxies` command. By default, it displays all proxies owned by your AWS account. To see details for a single proxy, specify its name with the `--db-proxy-name` parameter.

```
aws rds describe-db-proxies [--db-proxy-name proxy_name]
```

To view the other information associated with the proxy, use the following commands.

```
aws rds describe-db-proxy-target-groups --db-proxy-name proxy_name  
aws rds describe-db-proxy-targets --db-proxy-name proxy_name
```

Use the following sequence of commands to see more detail about the things that are associated with the proxy:

1. To get a list of proxies, run `describe-db-proxies`.
2. To show connection parameters such as the maximum percentage of connections that the proxy can use, run `describe-db-proxy-target-groups` `--db-proxy-name` and use the name of the proxy as the parameter value.
3. To see the details of the RDS DB instance or Aurora DB cluster associated with the returned target group, run `describe-db-proxy-targets`.

RDS API

To view your proxies using the RDS API, use the `DescribeDBProxies` operation. It returns values of the `DBProxy` data type.

To see details of the connection settings for the proxy, use the proxy identifiers from this return value with the `DescribeDBProxyTargetGroups` operation. It returns values of the `DBProxyTargetGroup` data type.

To see the RDS instance or Aurora DB cluster associated with the proxy, use the `DescribeDBProxyTargets` operation. It returns values of the `DBProxyTarget` data type.

Connecting to a database through RDS Proxy

You connect to an RDS DB instance or Aurora DB cluster through a proxy in generally the same way as you connect directly to the database. The main difference is that you specify the proxy endpoint instead of the instance or cluster endpoint. For an Aurora DB cluster, by default all proxy connections have read/write capability and use the writer instance. If you normally use the reader endpoint for read-only connections, you can create an additional read-only endpoint for the proxy and use that endpoint the same way. For more information, see [Overview of proxy endpoints \(p. 315\)](#).

Topics

- [Connecting to a proxy using native authentication \(p. 304\)](#)
- [Connecting to a proxy using IAM authentication \(p. 304\)](#)
- [Considerations for connecting to a proxy with PostgreSQL \(p. 305\)](#)

Connecting to a proxy using native authentication

Use the following basic steps to connect to a proxy using native authentication:

1. Find the proxy endpoint. In the AWS Management Console, you can find the endpoint on the details page for the corresponding proxy. With the AWS CLI, you can use the `describe-db-proxies` command. The following example shows how.

```
# Add --output text to get output as a simple tab-separated list.
$ aws rds describe-db-proxies --query '*[*].{DBProxyName:DBProxyName,Endpoint:Endpoint}'
[
    [
        {
            "Endpoint": "the-proxy.proxy-demo.us-east-1.rds.amazonaws.com",
            "DBProxyName": "the-proxy"
        },
        {
            "Endpoint": "the-proxy-other-secret.proxy-demo.us-east-1.rds.amazonaws.com",
            "DBProxyName": "the-proxy-other-secret"
        },
        {
            "Endpoint": "the-proxy-rds-secret.proxy-demo.us-east-1.rds.amazonaws.com",
            "DBProxyName": "the-proxy-rds-secret"
        },
        {
            "Endpoint": "the-proxy-t3.proxy-demo.us-east-1.rds.amazonaws.com",
            "DBProxyName": "the-proxy-t3"
        }
    ]
]
```

2. Specify that endpoint as the host parameter in the connection string for your client application. For example, specify the proxy endpoint as the value for the `mysql -h` option or `psql -h` option.
3. Supply the same database user name and password as you usually do.

Connecting to a proxy using IAM authentication

When you use IAM authentication with RDS Proxy, set up your database users to authenticate with regular user names and passwords. The IAM authentication applies to RDS Proxy retrieving the user name and password credentials from Secrets Manager. The connection from RDS Proxy to the underlying database doesn't go through IAM.

To connect to RDS Proxy using IAM authentication, follow the same general procedure as for connecting to an RDS DB instance or Aurora cluster using IAM authentication. For general information about using IAM with RDS and Aurora, see [Security in Amazon Aurora \(p. 1706\)](#).

The major differences in IAM usage for RDS Proxy include the following:

- You don't configure each individual database user with an authorization plugin. The database users still have regular user names and passwords within the database. You set up Secrets Manager secrets containing these user names and passwords, and authorize RDS Proxy to retrieve the credentials from Secrets Manager.

The IAM authentication applies to the connection between your client program and the proxy. The proxy then authenticates to the database using the user name and password credentials retrieved from Secrets Manager.

- Instead of the instance, cluster, or reader endpoint, you specify the proxy endpoint. For details about the proxy endpoint, see [Connecting to your DB cluster using IAM authentication \(p. 1750\)](#).
- In the direct database IAM authentication case, you selectively choose database users and configure them to be identified with a special authentication plugin. You can then connect to those users using IAM authentication.

In the proxy use case, you provide the proxy with Secrets that contain some user's user name and password (native authentication). You then connect to the proxy using IAM authentication. Here, you do this by generating an authentication token with the proxy endpoint, not the database endpoint. You also use a user name that matches one of the user names for the secrets that you provided.

- Make sure that you use Transport Layer Security (TLS)/Secure Sockets Layer (SSL) when connecting to a proxy using IAM authentication.

You can grant a specific user access to the proxy by modifying the IAM policy. An example follows.

```
"Resource": "arn:aws:rds-db:us-east-2:1234567890:dbuser:prx-ABCDEFGHIJKL01234/db_user"
```

Considerations for connecting to a proxy with PostgreSQL

For PostgreSQL, when a client starts a connection to a PostgreSQL database, it sends a startup message that includes pairs of parameter name and value strings. For details, see the [StartupMessage](#) in [PostgreSQL message formats](#) in the PostgreSQL documentation.

When connecting through an RDS proxy, the startup message can include the following currently recognized parameters:

- `user`
- `database`
- `replication`

The startup message can also include the following additional runtime parameters:

- `application_name`
- `client_encoding`
- `DateStyle`
- `TimeZone`
- `extra_float_digits`

For more information about PostgreSQL messaging, see the [Frontend/Backend protocol](#) in the PostgreSQL documentation.

For PostgreSQL, if you use JDBC we recommend the following to avoid pinning:

- Set the JDBC connection parameter `assumeMinServerVersion` to at least 9.0 to avoid pinning. Doing this prevents the JDBC driver from performing an extra round trip during connection startup when it runs `SET extra_float_digits = 3`.
- Set the JDBC connection parameter `ApplicationName` to `any/your-application-name` to avoid pinning. Doing this prevents the JDBC driver from performing an extra round trip during connection startup when it runs `SET application_name = "PostgreSQL JDBC Driver"`. Note the JDBC parameter is `ApplicationName` but the PostgreSQL `StartupMessage` parameter is `application_name`.
- Set the JDBC connection parameter `preferQueryMode` to `extendedForPrepared` to avoid pinning. The `extendedForPrepared` ensures that the extended mode is used only for prepared statements.

The default for the `preferQueryMode` parameter is `extended`, which uses the extended mode for all queries. The extended mode uses a series of `Prepare`, `Bind`, `Execute`, and `Sync` requests and corresponding responses. This type of series causes connection pinning in an RDS proxy.

For more information, see [Avoiding pinning \(p. 312\)](#). For more information about connecting using JDBC, see [Connecting to the database](#) in the PostgreSQL documentation.

Managing an RDS Proxy

Following, you can find an explanation of how to manage RDS Proxy operation and configuration. These procedures help your application make the most efficient use of database connections and achieve maximum connection reuse. The more that you can take advantage of connection reuse, the more CPU and memory overhead that you can save. This in turn reduces latency for your application and enables the database to devote more of its resources to processing application requests.

Topics

- [Modifying an RDS Proxy \(p. 306\)](#)
- [Adding a new database user \(p. 310\)](#)
- [Changing the password for a database user \(p. 311\)](#)
- [Configuring connection settings \(p. 311\)](#)
- [Avoiding pinning \(p. 312\)](#)
- [Deleting an RDS Proxy \(p. 314\)](#)

Modifying an RDS Proxy

You can change certain settings associated with a proxy after you create the proxy. You do so by modifying the proxy itself, its associated target group, or both. Each proxy has an associated target group.

AWS Management Console

To modify the settings for a proxy

1. Sign in to the AWS Management Console and open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. In the navigation pane, choose **Proxies**.
3. In the list of proxies, choose the proxy whose settings you want to modify or go to its details page.
4. For **Actions**, choose **Modify**.
5. Enter or choose the properties to modify. You can modify the following:

- **Proxy identifier** – Rename the proxy by entering a new identifier.
 - **Require Transport Layer Security** – Turn the requirement for Transport layer Security (TLS) on or off.
 - **Idle client connection timeout** – Enter a time period for the idle client connection timeout.
 - **Secrets Manager secrets** – Add or remove Secrets Manager secrets. These secrets correspond to database user names and passwords.
 - **IAM role** – Change the IAM role used to retrieve the secrets from Secrets Manager.
 - **IAM Authentication** – Require or disallow IAM authentication for connections to the proxy.
 - **VPC security group** – Add or remove VPC security groups for the proxy to use.
 - **Enable enhanced logging** – Enable or disable enhanced logging.
6. Choose **Modify**.

If you didn't find the settings listed that you want to change, use the following procedure to update the target group for the proxy. The *target group* associated with a proxy controls the settings related to the physical database connections. Each proxy has one associated target group named `default`, which is created automatically along with the proxy.

You can only modify the target group from the proxy details page, not from the list on the **Proxies** page.

To modify the settings for a proxy target group

1. On the **Proxies** page, go to the details page for a proxy.
2. For **Target groups**, choose the `default` link. Currently, all proxies have a single target group named `default`.
3. On the details page for the `default` target group, choose **Modify**.
4. Choose new settings for the properties that you can modify:
 - **Database** – Choose a different RDS DB instance or Aurora cluster.
 - **Connection pool maximum connections** – Adjust what percentage of the maximum available connections the proxy can use.
 - **Session pinning filters** – (Optional) Choose a session pinning filter. Doing this can help reduce performance issues due to insufficient transaction-level reuse for connections. Using this setting requires understanding of application behavior and the circumstances under which RDS Proxy pins a session to a database connection.
 - **Connection borrow timeout** – Adjust the connection borrow timeout interval. This setting applies when the maximum number of connections is already being used for the proxy. The setting determines how long the proxy waits for a connection to become available before returning a timeout error.
 - **Initialization query** – (Optional) Add an initialization query, or modify the current one. You can specify one or more SQL statements for the proxy to run when opening each new database connection. The setting is typically used with `SET` statements to make sure that each connection has identical settings such as time zone and character set. For multiple statements, use semicolons as the separator. You can also include multiple variables in a single `SET` statement, such as `SET x=1, y=2`. Initialization query is not currently supported for PostgreSQL.

You can't change certain properties, such as the target group identifier and the database engine.

5. Choose **Modify target group**.

AWS CLI

To modify a proxy using the AWS CLI, use the commands [modify-db-proxy](#), [modify-db-proxy-target-group](#), [deregister-db-proxy-targets](#), and [register-db-proxy-targets](#).

With the `modify-db-proxy` command, you can change properties such as the following:

- The set of Secrets Manager secrets used by the proxy.
- Whether TLS is required.
- The idle client timeout.
- Whether to log additional information from SQL statements for debugging.
- The IAM role used to retrieve Secrets Manager secrets.
- The security groups used by the proxy.

The following example shows how to rename an existing proxy.

```
aws rds modify-db-proxy --db-proxy-name the-proxy --new-db-proxy-name the_new_name
```

To modify connection-related settings or rename the target group, use the `modify-db-proxy-target-group` command. Currently, all proxies have a single target group named `default`. When working with this target group, you specify the name of the proxy and `default` for the name of the target group.

The following example shows how to first check the `MaxIdleConnectionsPercent` setting for a proxy and then change it, using the target group.

```
aws rds describe-db-proxy-target-groups --db-proxy-name the-proxy

{
    "TargetGroups": [
        {
            "Status": "available",
            "UpdatedDate": "2019-11-30T16:49:30.342Z",
            "ConnectionPoolConfig": {
                "MaxIdleConnectionsPercent": 50,
                "ConnectionBorrowTimeout": 120,
                "MaxConnectionsPercent": 100,
                "SessionPinningFilters": []
            },
            "TargetGroupName": "default",
            "CreatedDate": "2019-11-30T16:49:27.940Z",
            "DBProxyName": "the-proxy",
            "IsDefault": true
        }
    ]
}

aws rds modify-db-proxy-target-group --db-proxy-name the-proxy --target-group-name default
--connection-pool-config '
{ "MaxIdleConnectionsPercent": 75 }'

{
    "DBProxyTargetGroup": {
        "Status": "available",
        "UpdatedDate": "2019-12-02T04:09:50.420Z",
        "ConnectionPoolConfig": {
            "MaxIdleConnectionsPercent": 75,
            "ConnectionBorrowTimeout": 120,
            "MaxConnectionsPercent": 100,
            "SessionPinningFilters": []
        }
    }
}
```

```

        "SessionPinningFilters": []
    },
    "TargetGroupName": "default",
    "CreatedDate": "2019-11-30T16:49:27.940Z",
    "DBProxyName": "the-proxy",
    "IsDefault": true
}
}

```

With the `deregister-db-proxy-targets` and `register-db-proxy-targets` commands, you change which RDS DB instance or Aurora DB cluster the proxy is associated with through its target group. Currently, each proxy can connect to one RDS DB instance or Aurora DB cluster. The target group tracks the connection details for all the RDS DB instances in a Multi-AZ configuration, or all the DB instances in an Aurora cluster.

The following example starts with a proxy that is associated with an Aurora MySQL cluster named `cluster-56-2020-02-25-1399`. The example shows how to change the proxy so that it can connect to a different cluster named `provisioned-cluster`.

When you work with an RDS DB instance, you specify the `--db-instance-identifier` option. When you work with an Aurora DB cluster, you specify the `--db-cluster-identifier` option instead.

The following example modifies an Aurora MySQL proxy. An Aurora PostgreSQL proxy has port 5432.

```

aws rds describe-db-proxy-targets --db-proxy-name the-proxy

{
    "Targets": [
        {
            "Endpoint": "instance-9814.demo.us-east-1.rds.amazonaws.com",
            "Type": "RDS_INSTANCE",
            "Port": 3306,
            "RdsResourceId": "instance-9814"
        },
        {
            "Endpoint": "instance-8898.demo.us-east-1.rds.amazonaws.com",
            "Type": "RDS_INSTANCE",
            "Port": 3306,
            "RdsResourceId": "instance-8898"
        },
        {
            "Endpoint": "instance-1018.demo.us-east-1.rds.amazonaws.com",
            "Type": "RDS_INSTANCE",
            "Port": 3306,
            "RdsResourceId": "instance-1018"
        },
        {
            "Type": "TRACKED_CLUSTER",
            "Port": 0,
            "RdsResourceId": "cluster-56-2020-02-25-1399"
        },
        {
            "Endpoint": "instance-4330.demo.us-east-1.rds.amazonaws.com",
            "Type": "RDS_INSTANCE",
            "Port": 3306,
            "RdsResourceId": "instance-4330"
        }
    ]
}

aws rds deregister-db-proxy-targets --db-proxy-name the-proxy --db-cluster-identifier
cluster-56-2020-02-25-1399

```

```
aws rds describe-db-proxy-targets --db-proxy-name the-proxy

{
    "Targets": []
}

aws rds register-db-proxy-targets --db-proxy-name the-proxy --db-cluster-identifier
provisioned-cluster

{
    "DBProxyTargets": [
        {
            "Type": "TRACKED_CLUSTER",
            "Port": 0,
            "RdsResourceId": "provisioned-cluster"
        },
        {
            "Endpoint": "gkldje.demo.us-east-1.rds.amazonaws.com",
            "Type": "RDS_INSTANCE",
            "Port": 3306,
            "RdsResourceId": "gkldje"
        },
        {
            "Endpoint": "provisioned-1.demo.us-east-1.rds.amazonaws.com",
            "Type": "RDS_INSTANCE",
            "Port": 3306,
            "RdsResourceId": "provisioned-1"
        }
    ]
}
```

RDS API

To modify a proxy using the RDS API, you use the operations [ModifyDBProxy](#), [ModifyDBProxyTargetGroup](#), [DeregisterDBProxyTargets](#), and [RegisterDBProxyTargets](#) operations.

With [ModifyDBProxy](#), you can change properties such as the following:

- The set of Secrets Manager secrets used by the proxy.
- Whether TLS is required.
- The idle client timeout.
- Whether to log additional information from SQL statements for debugging.
- The IAM role used to retrieve Secrets Manager secrets.
- The security groups used by the proxy.

With [ModifyDBProxyTargetGroup](#), you can modify connection-related settings or rename the target group. Currently, all proxies have a single target group named `default`. When working with this target group, you specify the name of the proxy and `default` for the name of the target group.

With [DeregisterDBProxyTargets](#) and [RegisterDBProxyTargets](#), you change which RDS DB instance or Aurora DB cluster the proxy is associated with through its target group. Currently, each proxy can connect to one RDS DB instance or Aurora DB cluster. The target group tracks the connection details for all the RDS DB instances in a Multi-AZ configuration, or all the DB instances in an Aurora cluster.

Adding a new database user

In some cases, you might add a new database user to an RDS DB instance or Aurora cluster that's associated with a proxy. If so, add or repurpose a Secrets Manager secret to store the credentials for that user. To do this, choose one of the following options:

- Create a new Secrets Manager secret, using the procedure described in [Setting up database credentials in AWS Secrets Manager \(p. 296\)](#).
- Update the IAM role to give RDS Proxy access to the new Secrets Manager secret. To do so, update the resources section of the IAM role policy.
- If the new user takes the place of an existing one, update the credentials stored in the proxy's Secrets Manager secret for the existing user.

Changing the password for a database user

In some cases, you might change the password for a database user in an RDS DB instance or Aurora cluster that's associated with a proxy. If so, update the corresponding Secrets Manager secret with the new password.

Configuring connection settings

To adjust RDS Proxy's connection pooling, you can modify the following settings:

- [IdleClientTimeout \(p. 311\)](#)
- [MaxConnectionsPercent \(p. 311\)](#)
- [MaxIdleConnectionsPercent \(p. 312\)](#)
- [ConnectionBorrowTimeout \(p. 312\)](#)

IdleClientTimeout

You can specify how long a client connection can be idle before the proxy can close it. The default is 1,800 seconds (30 minutes).

A client connection is considered *idle* when the application doesn't submit a new request within the specified time after the previous request completed. The underlying database connection stays open and is returned to the connection pool. Thus, it's available to be reused for new client connections. If you want the proxy to proactively remove stale connections, consider lowering the idle client connection timeout. If your workload establishes frequent connections with the proxy, consider raising the idle client connection timeout to save the cost of establishing connections.

This setting is represented by the **Idle client connection timeout** field in the RDS console and the `IdleClientTimeout` setting in the AWS CLI and the API. To learn how to change the value of the **Idle client connection timeout** field in the RDS console, see [AWS Management Console \(p. 306\)](#). To learn how to change the value of the `IdleClientTimeout` setting, see the CLI command [modify-db-proxy](#) or the API operation [ModifyDBProxy](#).

MaxConnectionsPercent

You can limit the number of connections that an RDS Proxy can establish with the database. You specify the limit as a percentage of the maximum connections available for your database. The proxy doesn't create all of these connections in advance. This setting reserves the right for the proxy to establish these connections as the workload needs them.

For example, suppose that you configured RDS Proxy to use 75 percent of the maximum connections for your database that supports a maximum of 1,000 concurrent connections. In that case, RDS Proxy can open up to 750 database connections.

This setting is represented by the **Connection pool maximum connections** field in the RDS console and the `MaxConnectionsPercent` setting in the AWS CLI and the API. To learn how to change the value of the **Connection pool maximum connections** field in the RDS console, see [AWS Management Console \(p. 306\)](#). To learn how to change the value of the `MaxConnectionsPercent` setting, see the CLI command [modify-db-proxy-target-group](#) or the API operation [ModifyDBProxyTargetGroup](#).

For information on database connection limits, see [Maximum connections to an Aurora MySQL DB instance](#) and [Maximum connections to an Aurora PostgreSQL DB instance](#).

MaxIdleConnectionsPercent

You can control the number of idle database connections that RDS Proxy can keep in the connection pool. RDS Proxy considers a database connection in its pool to be *idle* when there's been no activity on the connection for five minutes.

You specify the limit as a percentage of the maximum connections available for your database. The default value is 50 percent and the upper limit is the value of `MaxConnectionsPercent`. With a high value, the proxy leaves a high percentage of idle database connections open. With a low value, the proxy closes a high percentage of idle database connections. If your workloads are unpredictable, consider setting a high value for `MaxIdleConnectionsPercent` so that RDS Proxy can accommodate surges in activity without opening a lot of new database connections.

This setting is represented by the `MaxIdleConnectionsPercent` setting of `DBProxyTargetGroup` in the AWS CLI and the API. To learn how to change the value of the `MaxIdleConnectionsPercent` setting, see the CLI command [modify-db-proxy-target-group](#) or the API operation [ModifyDBProxyTargetGroup](#).

Note

RDS Proxy closes database connections some time after 24 hours when they are no longer in use. The proxy performs this action regardless of the value of the maximum idle connections setting.

For information on database connection limits, see [Maximum connections to an Aurora MySQL DB instance](#) and [Maximum connections to an Aurora PostgreSQL DB instance](#).

ConnectionBorrowTimeout

You can choose how long RDS Proxy waits for a database connection in the connection pool to become available for use before returning a timeout error. The default is 120 seconds. This setting applies when the number of connections is at the maximum, and so no connections are available in the connection pool. It also applies if no appropriate database instance is available to handle the request because, for example, a failover operation is in process. Using this setting, you can set the best wait period for your application without having to change the query timeout in your application code.

This setting is represented by the **Connection borrow timeout** field in the RDS console or the `ConnectionBorrowTimeout` setting of `DBProxyTargetGroup` in the AWS CLI or API. To learn how to change the value of the **Connection borrow timeout** field in the RDS console, see [AWS Management Console \(p. 306\)](#). To learn how to change the value of the `ConnectionBorrowTimeout` setting, see the CLI command [modify-db-proxy-target-group](#) or the API operation [ModifyDBProxyTargetGroup](#).

Avoiding pinning

Multiplexing is more efficient when database requests don't rely on state information from previous requests. In that case, RDS Proxy can reuse a connection at the conclusion of each transaction. Examples of such state information include most variables and configuration parameters that you can change through `SET` or `SELECT` statements. SQL transactions on a client connection can multiplex between underlying database connections by default.

Your connections to the proxy can enter a state known as *pinning*. When a connection is pinned, each later transaction uses the same underlying database connection until the session ends. Other client connections also can't reuse that database connection until the session ends. The session ends when the client connection is dropped.

RDS Proxy automatically pins a client connection to a specific DB connection when it detects a session state change that isn't appropriate for other sessions. Pinning reduces the effectiveness of connection

reuse. If all or almost all of your connections experience pinning, consider modifying your application code or workload to reduce the conditions that cause the pinning.

For example, if your application changes a session variable or configuration parameter, later statements can rely on the new variable or parameter to be in effect. Thus, when RDS Proxy processes requests to change session variables or configuration settings, it pins that session to the DB connection. That way, the session state remains in effect for all later transactions in the same session.

This rule doesn't apply to all parameters you can set. RDS Proxy tracks changes to the character set, collation, time zone, autocommit, current database, SQL mode, and `session_track_schema` settings. Thus RDS Proxy doesn't pin the session when you modify these. In this case, RDS Proxy only reuses the connection for other sessions that have the same values for those settings.

Performance tuning for RDS Proxy involves trying to maximize transaction-level connection reuse (multiplexing) by minimizing pinning. You can do so by doing the following:

- Avoid unnecessary database requests that might cause pinning.
- Set variables and configuration settings consistently across all connections. That way, later sessions are more likely to reuse connections that have those particular settings.

However, for PostgreSQL setting a variable leads to session pinning.

- Apply a session pinning filter to the proxy. You can exempt certain kinds of operations from pinning the session if you know that doing so doesn't affect the correct operation of your application.
- See how frequently pinning occurs by monitoring the CloudWatch metric `DatabaseConnectionsCurrentlySessionPinned`. For information about this and other CloudWatch metrics, see [Monitoring RDS Proxy metrics with Amazon CloudWatch \(p. 324\)](#).
- If you use `SET` statements to perform identical initialization for each client connection, you can do so while preserving transaction-level multiplexing. In this case, you move the statements that set up the initial session state into the initialization query used by a proxy. This property is a string containing one or more SQL statements, separated by semicolons.

For example, you can define an initialization query for a proxy that sets certain configuration parameters. Then, RDS Proxy applies those settings whenever it sets up a new connection for that proxy. You can remove the corresponding `SET` statements from your application code, so that they don't interfere with transaction-level multiplexing.

Important

For proxies associated with MySQL databases, don't set the configuration parameter `sql_auto_is_null` to `true` or a nonzero value in the initialization query. Doing so might cause incorrect application behavior.

The proxy pins the session to the current connection in the following situations where multiplexing might cause unexpected behavior:

- Any statement with a text size greater than 16 KB causes the proxy to pin the session.
- Prepared statements cause the proxy to pin the session. This rule applies whether the prepared statement uses SQL text or the binary protocol.
- Explicit MySQL statements `LOCK TABLE`, `LOCK TABLES`, or `FLUSH TABLES WITH READ LOCK` cause the proxy to pin the session.
- Setting a user variable or a system variable (with some exceptions) causes the proxy to pin the session. If this situation reduces your connection reuse too much, you can choose for `SET` operations not to cause pinning. For information about how to do so by setting the `SessionPinningFilters` property, see [Creating an RDS Proxy \(p. 299\)](#).
- Creating a temporary table causes the proxy to pin the session. That way, the contents of the temporary table are preserved throughout the session regardless of transaction boundaries.

- Calling the MySQL functions `ROW_COUNT`, `FOUND_ROWS`, and `LAST_INSERT_ID` sometimes causes pinning.

The exact circumstances where these functions cause pinning might differ between Aurora MySQL versions that are compatible with MySQL 5.6 and MySQL 5.7.

Calling MySQL stored procedures and stored functions doesn't cause pinning. RDS Proxy doesn't detect any session state changes resulting from such calls. Therefore, make sure that your application doesn't change session state inside stored routines and rely on that session state to persist across transactions. For example, if a stored procedure creates a temporary table that is intended to persist across transactions, that application currently isn't compatible with RDS Proxy.

For PostgreSQL, the following interactions also cause pinning:

- Using `SET` commands
- Using the PostgreSQL extended query protocol such as by using JDBC default settings
- Creating temporary sequences, tables, or views
- Declaring cursors
- Discarding the session state
- Listening on a notification channel
- Loading a library module such as `auto_explain`
- Manipulating sequences using functions such as `nextval` and `setval`
- Interacting with locks using functions such as `pg_advisory_lock` and `pg_try_advisory_lock`
- Using prepared statements, setting parameters, or resetting a parameter to its default

If you have expert knowledge about your application behavior, you can skip the pinning behavior for certain application statements. To do so, choose the **Session pinning filters** option when creating the proxy. Currently, you can opt out of session pinning for setting session variables and configuration settings.

For metrics about how often pinning occurs for a proxy, see [Monitoring RDS Proxy metrics with Amazon CloudWatch \(p. 324\)](#).

Deleting an RDS Proxy

You can delete a proxy if you no longer need it. You might delete a proxy because the application that was using it is no longer relevant. Or you might delete a proxy if you take the DB instance or cluster associated with it out of service.

AWS Management Console

To delete a proxy

1. Sign in to the AWS Management Console and open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. In the navigation pane, choose **Proxies**.
3. Choose the proxy to delete from the list.
4. Choose **Delete Proxy**.

AWS CLI

To delete a DB proxy, use the AWS CLI command `delete-db-proxy`. To remove related associations, also use the `deregister-db-proxy-targets` command.

```
aws rds delete-db-proxy --name proxy_name
```

```
aws rds deregister-db-proxy-targets
--db-proxy-name proxy_name
[--target-group-name target_group_name]
[--target-ids comma_separated_list]      # or
[--db-instance-identifiers instance_id]      # or
[--db-cluster-identifiers cluster_id]
```

RDS API

To delete a DB proxy, call the Amazon RDS API function [DeleteDBProxy](#). To delete related items and associations, you also call the functions [DeleteDBProxyTargetGroup](#) and [DeregisterDBProxyTargets](#).

Working with Amazon RDS Proxy endpoints

Following, you can learn about endpoints for RDS Proxy and how to use them. By using endpoints, you can take advantage of the following capabilities:

- You can use multiple endpoints with a proxy to monitor and troubleshoot connections from different applications independently.
- You can use reader endpoints with Aurora DB clusters to improve read scalability and high availability for your query-intensive applications.
- You can use a cross-VPC endpoint to allow access to databases in one VPC from resources such as Amazon EC2 instances in a different VPC.

Topics

- [Overview of proxy endpoints \(p. 315\)](#)
- [Using reader endpoints with Aurora clusters \(p. 316\)](#)
- [Accessing Aurora and RDS databases across VPCs \(p. 319\)](#)
- [Creating a proxy endpoint \(p. 319\)](#)
- [Viewing proxy endpoints \(p. 321\)](#)
- [Modifying a proxy endpoint \(p. 322\)](#)
- [Deleting a proxy endpoint \(p. 323\)](#)
- [Limits for proxy endpoints \(p. 324\)](#)

Overview of proxy endpoints

Working with RDS Proxy endpoints involves the same kinds of procedures as with Aurora DB cluster and reader endpoints and RDS instance endpoints. If you aren't familiar with Aurora endpoints, find more information in [Amazon Aurora connection management \(p. 32\)](#).

By default, the endpoint that you connect to when you use RDS Proxy with an Aurora cluster has read/write capability. As a consequence, this endpoint sends all requests to the writer instance of the cluster, and all of those connections count against the `max_connections` value for the writer instance. If your proxy is associated with an Aurora DB cluster, you can create additional read/write or read-only endpoints for that proxy.

You can use a read-only endpoint with your proxy for read-only queries, the same way that you use the reader endpoint for an Aurora provisioned cluster. Doing so helps you to take advantage of the read scalability of an Aurora cluster with one or more reader DB instances. You can run more simultaneous

queries and make more simultaneous connections by using a read-only endpoint and adding more reader DB instances to your Aurora cluster as needed.

Tip

When you create a proxy for an Aurora cluster using the AWS Management Console, you can choose for RDS Proxy to automatically create a reader endpoint. For information about the benefits of a reader endpoint, see [Using reader endpoints with Aurora clusters \(p. 316\)](#).

For a proxy endpoint that you create, you can also associate the endpoint with a different virtual private cloud (VPC) than the proxy itself uses. By doing so, you can connect to the proxy from a different VPC, for example a VPC used by a different application within your organization.

For information about limits associated with proxy endpoints, see [Limits for proxy endpoints \(p. 324\)](#).

In the RDS Proxy logs, each entry is prefixed with the name of the associated proxy endpoint. This name can be the name you specified for a user-defined endpoint, or the special name `default` for read/write requests using the default endpoint of a proxy.

Each proxy endpoint has its own set of CloudWatch metrics. You can monitor the metrics for all endpoints of a proxy. You can also monitor metrics for a specific endpoint, or for all the read/write or read-only endpoints of a proxy. For more information, see [Monitoring RDS Proxy metrics with Amazon CloudWatch \(p. 324\)](#).

A proxy endpoint uses the same authentication mechanism as its associated proxy. RDS Proxy automatically sets up permissions and authorizations for the user-defined endpoint, consistent with the properties of the associated proxy.

Using reader endpoints with Aurora clusters

You can create and connect to read-only endpoints called *reader endpoints* when you use RDS Proxy with Aurora clusters. These reader endpoints help to improve the read scalability of your query-intensive applications. Reader endpoints also help to improve the availability of your connections if a reader DB instance in your cluster becomes unavailable.

Note

When you specify that a new endpoint is read-only, RDS Proxy requires that the Aurora cluster has one or more reader DB instances. If you change the target of the proxy to an Aurora cluster containing only a single writer or a multi-writer Aurora cluster, any requests to the reader endpoint fail with an error. Requests also fail if the target of the proxy is an RDS instance instead of an Aurora cluster.

If an Aurora cluster has reader instances but those instances aren't available, RDS Proxy waits to send the request instead of returning an error immediately. If no reader instance becomes available within the connection borrow timeout period, the request fails with an error.

How reader endpoints help application availability

In some cases, one or more reader instances in your cluster might become unavailable. If so, connections that use a reader endpoint of a DB proxy can recover more quickly than ones that use the Aurora reader endpoint. RDS Proxy routes connections to only the available reader instances in the cluster. There isn't a delay due to DNS caching when an instance becomes unavailable.

If the connection is multiplexed, RDS Proxy directs subsequent queries to a different reader DB instance without any interruption to your application. During the automatic switchover to a new reader instance, RDS Proxy checks the replication lag of the old and new reader instances. RDS Proxy makes sure that the new reader instance is up to date with the same changes as the previous reader instance. That way, your application never sees stale data when RDS Proxy switches from one reader DB instance to another.

If the connection is pinned, the next query on the connection returns an error. However, your application can immediately reconnect to the same endpoint. RDS Proxy routes the connection to a different reader

DB instance that's in available state. When you manually reconnect, RDS Proxy doesn't check the replication lag between the old and new reader instances.

If your Aurora cluster doesn't have any available reader instances, RDS Proxy checks whether this condition is temporary or permanent. The behavior in each case is as follows:

- Suppose that your cluster has one or more reader DB instances, but none of them are in the Available state. For example, all reader instances might be rebooting or encountering problems. In that case, attempts to connect to a reader endpoint wait for a reader instance to become available. If no reader instance becomes available within the connection borrow timeout period, the connection attempt fails. If a reader instance does become available, the connection attempt succeeds.
- Suppose that your cluster has no reader DB instances. In that case, RDS Proxy returns an error immediately if you try to connect to a reader endpoint. To resolve this problem, add one or more reader instances to your cluster before you connect to the reader endpoint.

How reader endpoints help query scalability

Reader endpoints for a proxy help with Aurora query scalability in the following ways:

- As you add reader instances to your Aurora cluster, RDS Proxy can route new connections to any reader endpoints to the different reader instances. That way, queries performed using one reader endpoint connection don't slow down queries performed using another reader endpoint connection. The queries run on separate DB instances. Each DB instance has its own compute resources, buffer cache, and so on.
- Where practical, RDS Proxy uses the same reader DB instance for all the queries issued using a particular reader endpoint connection. That way, a set of related queries on the same tables can take advantage of caching, plan optimization, and so on, on a particular DB instance.
- If a reader DB instance becomes unavailable, the effect on your application depends on whether the session is multiplexed or pinned. If the session is multiplexed, RDS Proxy routes any subsequent queries to a different reader DB instance without any action on your part. If the session is pinned, your application gets an error and must reconnect. You can reconnect to the reader endpoint immediately and RDS Proxy routes the connection to an available reader DB instance. For more information about multiplexing and pinning for proxy sessions, see [Overview of RDS Proxy concepts \(p. 291\)](#).
- The more reader DB instances you have in the cluster, the more simultaneous connections you can make using reader endpoints. For example, suppose that your cluster has four reader DB instances, each configured to allow 200 simultaneous connections. Suppose that your proxy is configured to use 50% of the maximum connections. Here, the maximum number of connections that you can make through the reader endpoints in the proxy is 100 (50% of 200) for reader 1. It's also 100 for reader 2, and so on, for a total of 400. If you double the number of reader DB instances in the cluster to eight, the maximum number of connections through the reader endpoints also doubles to 800.

Examples of using reader endpoints

The following Linux example shows how you can confirm that you're connected to an Aurora MySQL cluster through a reader endpoint. The `innodb_read_only` configuration setting is enabled. Attempts to perform write operations such as `CREATE DATABASE` statements fail with an error. And you can confirm that you're connected to a reader DB instance by checking the DB instance name using the `aurora_server_id` variable.

Tip

Don't rely only on checking the DB instance name to determine whether the connection is read/write or read-only. Remember that DB instances in an Aurora cluster can change roles between writer and reader when failovers happen.

```
$ mysql -h endpoint-demo-reader.endpoint.proxy-demo.us-east-1.rds.amazonaws.com -u admin -p  
...
```

```

mysql> select @@innodb_read_only;
+-----+
| @@innodb_read_only |
+-----+
| 1 |
+-----+
mysql> create database shouldnt_work;
ERROR 1290 (HY000): The MySQL server is running with the --read-only option so it cannot
execute this statement

mysql> select @@aurora_server_id;
+-----+
| @@aurora_server_id |
+-----+
| proxy-reader-endpoint-demo-instance-3 |
+-----+

```

The following example shows how your connection to a proxy reader endpoint can keep working even when the reader DB instance is deleted. In this example, the Aurora cluster has two reader instances, `instance-5507` and `instance-7448`. The connection to the reader endpoint begins using one of the reader instances. During the example, this reader instance is deleted by a `delete-db-instance` command. RDS Proxy switches to a different reader instance for subsequent queries.

```

$ mysql -h reader-demo.endpoint.proxy-demo.us-east-1.rds.amazonaws.com
-u my_user -p
...
mysql> select @@aurora_server_id;
+-----+
| @@aurora_server_id |
+-----+
| instance-5507 |
+-----+

mysql> select @@innodb_read_only;
+-----+
| @@innodb_read_only |
+-----+
| 1 |
+-----+

mysql> select count(*) from information_schema.tables;
+-----+
| count(*) |
+-----+
| 328 |
+-----+

```

While the `mysql` session is still running, the following command deletes the reader instance that the reader endpoint is connected to.

```
aws rds delete-db-instance --db-instance-identifier instance-5507 --skip-final-snapshot
```

Queries in the `mysql` session continue working without the need to reconnect. RDS Proxy automatically switches to a different reader DB instance.

```

mysql> select @@aurora_server_id;
+-----+
| @@aurora_server_id |
+-----+
| instance-7448 |
+-----+

```

```
mysql> select count(*) from information_schema.TABLES;
+-----+
| count(*) |
+-----+
|      328 |
+-----+
```

Accessing Aurora and RDS databases across VPCs

By default, the components of your RDS and Aurora technology stack are all in the same Amazon VPC. For example, suppose that an application running on an Amazon EC2 instance connects to an Amazon RDS DB instance or an Aurora DB cluster. In this case, the application server and database must both be within the same VPC.

With RDS Proxy, you can set up access to an Aurora cluster or RDS instance in one VPC from resources such as EC2 instances in another VPC. For example, your organization might have multiple applications that access the same database resources. Each application might be in its own VPC.

To enable cross-VPC access, you create a new endpoint for the proxy. If you aren't familiar with creating proxy endpoints, see [Working with Amazon RDS Proxy endpoints \(p. 315\)](#) for details. The proxy itself resides in the same VPC as the Aurora DB cluster or RDS instance. However, the cross-VPC endpoint resides in the other VPC, along with the other resources such as the EC2 instances. The cross-VPC endpoint is associated with subnets and security groups from the same VPC as the EC2 and other resources. These associations let you connect to the endpoint from the applications that otherwise can't access the database due to the VPC restrictions.

The following steps explain how to create and access a cross-VPC endpoint through RDS Proxy:

1. Create two VPCs, or choose two VPCs that you already use for Aurora and RDS work. Each VPC should have its own associated network resources such as an Internet gateway, route tables, subnets, and security groups. If you only have one VPC, you can consult [Getting started with Amazon Aurora \(p. 89\)](#) for the steps to set up another VPC to use Aurora successfully. You can also examine your existing VPC in the Amazon EC2 console to see what kinds of resources to connect together.
2. Create a DB proxy associated with the Aurora DB cluster or RDS instance that you want to connect to. Follow the procedure in [Creating an RDS Proxy \(p. 299\)](#).
3. On the **Details** page for your proxy in the RDS console, under the **Proxy endpoints** section, choose **Create endpoint**. Follow the procedure in [Creating a proxy endpoint \(p. 319\)](#).
4. Choose whether to make the cross-VPC endpoint read/write or read-only.
5. Instead of accepting the default of the same VPC as the Aurora DB cluster or RDS instance, choose a different VPC. This VPC must be in the same AWS Region as the VPC where the proxy resides.
6. Now instead of accepting the defaults for subnets and security groups from the same VPC as the Aurora DB cluster or RDS instance, make new selections. Make these based on the subnets and security groups from the VPC that you chose.
7. You don't need to change any of the settings for the Secrets Manager secrets. The same credentials work for all endpoints for your proxy, regardless of which VPC each endpoint is in.
8. Wait for the new endpoint to reach the **Available** state.
9. Make a note of the full endpoint name. This is the value ending in `Region_name.rds.amazonaws.com` that you supply as part of the connection string for your database application.
10. Access the new endpoint from a resource in the same VPC as the endpoint. A simple way to test this process is to create a new EC2 instance in this VPC. Then you can log into the EC2 instance and run the `mysql` or `psql` commands to connect by using the endpoint value in your connection string.

Creating a proxy endpoint

Console

To create a proxy endpoint

1. Sign in to the AWS Management Console and open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. In the navigation pane, choose **Proxies**.
3. Click the name of the proxy that you want to create a new endpoint for.

The details page for that proxy appears.

4. In the **Proxy endpoints** section, choose **Create proxy endpoint**.
- The **Create proxy endpoint** window appears.
5. For **Proxy endpoint name**, enter a descriptive name of your choice.
 6. For **Target role**, choose whether to make the endpoint read/write or read-only.

Connections that use a read/write endpoint can perform any kind of operation: data definition language (DDL) statements, data manipulation language (DML) statements, and queries. These endpoints always connect to the primary instance of the Aurora cluster. You can use read/write endpoints for general database operations when you only use a single endpoint in your application. You can also use read/write endpoints for administrative operations, online transaction processing (OLTP) applications, and extract-transform-load (ETL) jobs.

Connections that use a read-only endpoint can only perform queries. When there are multiple reader instances in the Aurora cluster, RDS Proxy can use a different reader instance for each connection to the endpoint. That way, a query-intensive application can take advantage of Aurora's clustering capability. You can add more query capacity to the cluster by adding more reader DB instances. These read-only connections don't impose any overhead on the primary instance of the cluster. That way, your reporting and analysis queries don't slow down the write operations of your OLTP applications.

7. For **Virtual Private Cloud (VPC)**, choose the default if you plan to access the endpoint from the same EC2 instances or other resources where you normally access the proxy or its associated database. If you want to set up cross-VPC access for this proxy, choose a VPC other than the default. For more information about cross-VPC access, see [Accessing Aurora and RDS databases across VPCs \(p. 319\)](#).
8. For **Subnets**, RDS Proxy fills in the same subnets as the associated proxy by default. If you want to restrict access to the endpoint so that only a portion of the address range of the VPC can connect to it, remove one or more subnets from the set of choices.
9. For **VPC security group**, you can choose an existing security group or create a new one. RDS Proxy fills in the same security group or groups as the associated proxy by default. If the inbound and outbound rules for the proxy are appropriate for this endpoint, you can leave the default choice.

If you choose to create a new security group, specify a name for the security group on this page. Then edit the security group settings from the EC2 console afterward.

10. Choose **Create proxy endpoint**.

AWS CLI

To create a proxy endpoint, use the AWS CLI `create-db-proxy-endpoint` command.

Include the following required parameters:

- `--db-proxy-name value`
- `--db-proxy-endpoint-name value`

- `--vpc-subnet-ids` *list_of_ids*. Separate the subnet IDs with spaces. You don't specify the ID of the VPC itself.

You can also include the following optional parameters:

- `--target-role` { `READ_WRITE` | `READ_ONLY` }. This parameter defaults to `READ_WRITE`. The `READ_ONLY` value only has an effect on Aurora provisioned clusters that contain one or more reader DB instances. When the proxy is associated with an RDS instance or with an Aurora cluster that only contains a writer DB instance, you can't specify `READ_ONLY`. For more information about the intended use of read-only endpoints with Aurora clusters, see [Using reader endpoints with Aurora clusters \(p. 316\)](#).
- `--vpc-security-group-ids` *value*. Separate the security group IDs with spaces. If you omit this parameter, RDS Proxy uses the default security group for the VPC. RDS Proxy determines the VPC based on the subnet IDs that you specify for the `--vpc-subnet-ids` parameter.

Example

The following example creates a proxy endpoint named `my-endpoint`.

For Linux, macOS, or Unix:

```
aws rds create-db-proxy-endpoint \
--db-proxy-name my-proxy \
--db-proxy-endpoint-name my-endpoint \
--vpc-subnet-ids subnet_id subnet_id subnet_id ... \
--target-role READ_ONLY \
--vpc-security-group-ids security_group_id ]
```

For Windows:

```
aws rds create-db-proxy-endpoint ^
--db-proxy-name my-proxy ^
--db-proxy-endpoint-name my-endpoint ^
--vpc-subnet-ids subnet_id_1 subnet_id_2 subnet_id_3 ... ^
--target-role READ_ONLY ^
--vpc-security-group-ids security_group_id
```

RDS API

To create a proxy endpoint, use the RDS API [CreateProxyEndpoint](#) action.

Viewing proxy endpoints

Console

To view the details for a proxy endpoint

1. Sign in to the AWS Management Console and open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. In the navigation pane, choose **Proxies**.
3. In the list, choose the proxy whose endpoint you want to view. Click the proxy name to view its details page.
4. In the **Proxy endpoints** section, choose the endpoint that you want to view. Click its name to view the details page.
5. Examine the parameters whose values you're interested in. You can check properties such as the following:

- Whether the endpoint is read/write or read-only.
- The endpoint address that you use in a database connection string.
- The VPC, subnets, and security groups associated with the endpoint.

AWS CLI

To view one or more DB proxy endpoints, use the AWS CLI [describe-db-proxy-endpoints](#) command.

You can include the following optional parameters:

- `--db-proxy-endpoint-name`
- `--db-proxy-name`

The following example describes the `my-endpoint` proxy endpoint.

Example

For Linux, macOS, or Unix:

```
aws rds describe-db-proxy-endpoints \
--db-proxy-endpoint-name my-endpoint
```

For Windows:

```
aws rds describe-db-proxy-endpoints ^
--db-proxy-endpoint-name my-endpoint
```

RDS API

To describe one or more proxy endpoints, use the RDS API [DescribeDBProxyEndpoints](#) operation.

Modifying a proxy endpoint

Console

To modify one or more proxy endpoints

1. Sign in to the AWS Management Console and open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. In the navigation pane, choose **Proxies**.
3. In the list, choose the proxy whose endpoint you want to modify. Click the proxy name to view its details page.
4. In the **Proxy endpoints** section, choose the endpoint that you want to modify. You can select it in the list, or click its name to view the details page.
5. On the proxy details page, under the **Proxy endpoints** section, choose **Edit**. Or on the proxy endpoint details page, for **Actions**, choose **Edit**.
6. Change the values of the parameters that you want to modify.
7. Choose **Save changes**.

AWS CLI

To modify a DB proxy endpoint, use the AWS CLI [modify-db-proxy-endpoint](#) command with the following required parameters:

- `--db-proxy-endpoint-name`

Specify changes to the endpoint properties by using one or more of the following parameters:

- `--new-db-proxy-endpoint-name`
- `--vpc-security-group-ids`. Separate the security group IDs with spaces.

The following example renames the `my-endpoint` proxy endpoint to `new-endpoint-name`.

Example

For Linux, macOS, or Unix:

```
aws rds modify-db-proxy-endpoint \
--db-proxy-endpoint-name my-endpoint \
--new-db-proxy-endpoint-name new-endpoint-name
```

For Windows:

```
aws rds modify-db-proxy-endpoint ^
--db-proxy-endpoint-name my-endpoint ^
--new-db-proxy-endpoint-name new-endpoint-name
```

RDS API

To modify a proxy endpoint, use the RDS API [ModifyDBProxyEndpoint](#) operation.

Deleting a proxy endpoint

You can delete an endpoint for your proxy using the console as described following.

Note

You can't delete the default endpoint that RDS Proxy automatically creates for each proxy. When you delete a proxy, RDS Proxy automatically deletes all the associated endpoints.

Console

To delete a proxy endpoint using the AWS Management Console

1. In the navigation pane, choose **Proxies**.
2. In the list, choose the proxy whose endpoint you want to endpoint. Click the proxy name to view its details page.
3. In the **Proxy endpoints** section, choose the endpoint that you want to delete. You can select one or more endpoints in the list, or click the name of a single endpoint to view the details page.
4. On the proxy details page, under the **Proxy endpoints** section, choose **Delete**. Or on the proxy endpoint details page, for **Actions**, choose **Delete**.

AWS CLI

To delete a proxy endpoint, run the `delete-db-proxy-endpoint` command with the following required parameters:

- `--db-proxy-endpoint-name`

The following command deletes the proxy endpoint named `my-endpoint`.

For Linux, macOS, or Unix:

```
aws rds delete-db-proxy-endpoint \
--db-proxy-endpoint-name my-endpoint
```

For Windows:

```
aws rds delete-db-proxy-endpoint ^
--db-proxy-endpoint-name my-endpoint
```

RDS API

To delete a proxy endpoint with the RDS API, run the [DeleteDBProxyEndpoint](#) operation. Specify the name of the proxy endpoint for the `DBProxyEndpointName` parameter.

Limits for proxy endpoints

Each proxy has a default endpoint that you can modify but not create or delete.

The maximum number of user-defined endpoints for a proxy is 20. Thus, a proxy can have up to 21 endpoints: the default endpoint, plus 20 that you create.

When you associate additional endpoints with a proxy, RDS Proxy automatically determines which DB instances in your cluster to use for each endpoint. You can't choose specific instances the way that you can with Aurora custom endpoints.

Reader endpoints aren't available for Aurora multi-writer clusters.

You can connect to proxy endpoints that you create using the SSL modes `REQUIRED` and `VERIFY_CA`. You can't connect to an endpoint that you create using the SSL mode `VERIFY_IDENTITY`.

Monitoring RDS Proxy metrics with Amazon CloudWatch

You can monitor RDS Proxy by using Amazon CloudWatch. CloudWatch collects and processes raw data from the proxies into readable, near-real-time metrics. To find these metrics in the CloudWatch console, choose **Metrics**, then choose **RDS**, and choose **Per-Proxy Metrics**. For more information, see [Using Amazon CloudWatch metrics](#) in the Amazon CloudWatch User Guide.

Note

RDS publishes these metrics for each underlying Amazon EC2 instance associated with a proxy.

A single proxy might be served by more than one EC2 instance. Use CloudWatch statistics to aggregate the values for a proxy across all the associated instances.

Some of these metrics might not be visible until after the first successful connection by a proxy.

In the RDS Proxy logs, each entry is prefixed with the name of the associated proxy endpoint. This name can be the name you specified for a user-defined endpoint, or the special name `default` for read/write requests using the default endpoint of a proxy.

All RDS Proxy metrics are in the group `proxy`.

Each proxy endpoint has its own CloudWatch metrics. You can monitor the usage of each proxy endpoint independently. For more information about proxy endpoints, see [Working with Amazon RDS Proxy endpoints \(p. 315\)](#).

You can aggregate the values for each metric using one of the following dimension sets. For example, by using the `ProxyName` dimension set, you can analyze all the traffic for a particular proxy. By using the other dimension sets, you can split the metrics in different ways. You can split the metrics based on

the different endpoints or target databases of each proxy, or the read/write and read-only traffic to each database.

- Dimension set 1: `ProxyName`
- Dimension set 2: `ProxyName, EndpointName`
- Dimension set 3: `ProxyName, TargetGroup, Target`
- Dimension set 4: `ProxyName, TargetGroup, TargetRole`

Metric	Description	Valid period	CloudWatch dimension set
AvailabilityPercentage	The percentage of time for which the target group was available in the role indicated by the dimension. This metric is reported every minute. The most useful statistic for this metric is <code>Average</code> .	1 minute	Dimension set 4 (p. 325)
ClientConnections	The current number of client connections. This metric is reported every minute. The most useful statistic for this metric is <code>Sum</code> .	1 minute	Dimension set 1 (p. 325) , Dimension set 2 (p. 325)
ClientConnectionsClosed	The number of client connections closed. The most useful statistic for this metric is <code>Sum</code> .	1 minute and above	Dimension set 1 (p. 325) , Dimension set 2 (p. 325)
ClientConnectionsNotEncrypted	The current number of client connections without Transport Layer Security (TLS). This metric is reported every minute. The most useful statistic for this metric is <code>Sum</code> .	1 minute and above	Dimension set 1 (p. 325) , Dimension set 2 (p. 325)
ClientConnectionsReceived	The number of client connection requests received. The most useful statistic for this metric is <code>Sum</code> .	1 minute and above	Dimension set 1 (p. 325) , Dimension set 2 (p. 325)
ClientConnectionsRejected	The number of client connection attempts that failed due to misconfigured authentication or TLS. The most useful statistic for this metric is <code>Sum</code> .	1 minute and above	Dimension set 1 (p. 325) , Dimension set 2 (p. 325)

Metric	Description	Valid period	CloudWatch dimension set
ClientConnectionsSet	The number of client connections successfully established with any authentication mechanism with or without TLS. The most useful statistic for this metric is Sum.	1 minute and above	Dimension set 1 (p. 325), Dimension set 2 (p. 325)
ClientConnectionsTLS	The current number of client connections with TLS. This metric is reported every minute. The most useful statistic for this metric is Sum.	1 minute and above	Dimension set 1 (p. 325), Dimension set 2 (p. 325)
DatabaseConnectionRequests	The number of requests to create a database connection. The most useful statistic for this metric is Sum.	1 minute and above	Dimension set 1 (p. 325), Dimension set 3 (p. 325), Dimension set 4 (p. 325)
DatabaseConnectionRequestsTLS	The number of requests to create a database connection with TLS. The most useful statistic for this metric is Sum.	1 minute and above	Dimension set 1 (p. 325), Dimension set 3 (p. 325), Dimension set 4 (p. 325)
DatabaseConnections	The current number of database connections. This metric is reported every minute. The most useful statistic for this metric is Sum.	1 minute	Dimension set 1 (p. 325), Dimension set 3 (p. 325), Dimension set 4 (p. 325)
DatabaseConnectionsLatency	The latency in microseconds that it takes for the proxy being monitored to get a database connection. The most useful statistic for this metric is Average.	1 minute and above	Dimension set 1 (p. 325), Dimension set 2 (p. 325)
DatabaseConnectionsPending	The current number of database connections in the borrow state. This metric is reported every minute. The most useful statistic for this metric is Sum.	1 minute	Dimension set 1 (p. 325), Dimension set 3 (p. 325), Dimension set 4 (p. 325)

Metric	Description	Valid period	CloudWatch dimension set
DatabaseConnections	The current number of database connections in a transaction. This metric is reported every minute. The most useful statistic for this metric is Sum.	1 minute	Dimension set 1 (p. 325), Dimension set 3 (p. 325), Dimension set 4 (p. 325)
DatabaseConnections	The current session of database connections currently pinned because of operations in client requests that change session state. This metric is reported every minute. The most useful statistic for this metric is Sum.	1 minute	Dimension set 1 (p. 325), Dimension set 3 (p. 325), Dimension set 4 (p. 325)
DatabaseConnections	The number of database connection requests that failed. The most useful statistic for this metric is Sum.	1 minute and above	Dimension set 1 (p. 325), Dimension set 3 (p. 325), Dimension set 4 (p. 325)
DatabaseConnections	The number of database connections successfully established with or without TLS. The most useful statistic for this metric is Sum.	1 minute and above	Dimension set 1 (p. 325), Dimension set 3 (p. 325), Dimension set 4 (p. 325)
DatabaseConnections	The current number of database connections with TLS. This metric is reported every minute. The most useful statistic for this metric is Sum.	1 minute	Dimension set 1 (p. 325), Dimension set 3 (p. 325), Dimension set 4 (p. 325)
MaxDatabaseConnections	The maximum number of database connections allowed. This metric is reported every minute. The most useful statistic for this metric is Sum.	1 minute	Dimension set 1 (p. 325), Dimension set 3 (p. 325), Dimension set 4 (p. 325)

Metric	Description	Valid period	CloudWatch dimension set
QueryDatabaseResponseLatency	The time in microseconds that the database took to respond to the query. The most useful statistic for this metric is Average .	1 minute and above	Dimension set 1 (p. 325) , Dimension set 2 (p. 325) , Dimension set 3 (p. 325) , Dimension set 4 (p. 325)
QueryRequests	The number of queries received. A query including multiple statements is counted as one query. The most useful statistic for this metric is Sum .	1 minute and above	Dimension set 1 (p. 325) , Dimension set 2 (p. 325)
QueryRequestsNoTLS	The number of queries received from non-TLS connections. A query including multiple statements is counted as one query. The most useful statistic for this metric is Sum .	1 minute and above	Dimension set 1 (p. 325) , Dimension set 2 (p. 325)
QueryRequestsTLS	The number of queries received from TLS connections. A query including multiple statements is counted as one query. The most useful statistic for this metric is Sum .	1 minute and above	Dimension set 1 (p. 325) , Dimension set 2 (p. 325)
QueryResponseLatency	The time in microseconds between getting a query request and the proxy responding to it. The most useful statistic for this metric is Average .	1 minute and above	Dimension set 1 (p. 325) , Dimension set 2 (p. 325)

You can find logs of RDS Proxy activity under CloudWatch in the AWS Management Console. Each proxy has an entry in the **Log groups** page.

Important

These logs are intended for human consumption for troubleshooting purposes and not for programmatic access. The format and content of the logs is subject to change. In particular, older logs don't contain any prefixes indicating the endpoint for each request. In newer logs, each entry is prefixed with the name of the associated proxy endpoint. This name can be the name that you specified for a user-defined endpoint, or the special name `default` for requests using the default endpoint of a proxy.

Working with RDS Proxy events

An *event* indicates a change in an environment. This can be an AWS environment or a service or application from a software as a service (SaaS) partner. Or it can be one of your own custom applications or services. For example, Amazon Aurora generates an event when you create or modify an RDS Proxy. Amazon Aurora delivers events to CloudWatch Events and Amazon EventBridge in near-real time. Following, you can find a list of RDS Proxy events that you can subscribe to and an example of an RDS Proxy event.

For more information about working with events, see the following:

- For instructions on how to view events by using the AWS Management Console, AWS CLI, or RDS API, see [Viewing Amazon RDS events \(p. 674\)](#).
- To learn how to configure Amazon Aurora to send events to EventBridge, see [Creating a rule that triggers on an Amazon Aurora event \(p. 692\)](#).

RDS Proxy events

The following table shows the event category and a list of events when an RDS Proxy is the source type.

Category	RDS event ID	Description
Configuration change	RDS-EVENT-0204	RDS modified the DB proxy (RDS Proxy).
Configuration change	RDS-EVENT-0207	RDS modified the endpoint of the DB proxy (RDS Proxy).
Configuration change	RDS-EVENT-0213	RDS detected the addition of the DB instance and automatically added it to the target group of the DB proxy (RDS Proxy).
Configuration change	RDS-EVENT-0214	RDS detected the deletion of the DB instance and automatically removed it from the target group of the DB proxy (RDS Proxy).
Configuration change	RDS-EVENT-XXXX	RDS detected the deletion of the DB cluster and automatically removed it from the target group of the DB proxy (RDS Proxy).
Creation	RDS-EVENT-0203	RDS created the DB proxy (RDS Proxy).
Creation	RDS-EVENT-0206	RDS created the endpoint for the DB proxy (RDS Proxy).
Deletion	RDS-EVENT-0205	RDS deleted the DB proxy (RDS Proxy).
Deletion	RDS-EVENT-0208	RDS deleted the endpoint of DB proxy (RDS Proxy).

The following is an example of an RDS Proxy event in JSON format. The event shows that RDS modified the endpoint named `my-endpoint` of the RDS Proxy named `my-rds-proxy`. The event ID is RDS-EVENT-0207.

```
{  
  "version": "0",  
  "eventVersion": "1.0",  
  "eventID": "12345678901234567890123456789012",  
  "eventSource": "aws.rds.proxy",  
  "eventSourceARN": "arn:aws:rds:us-east-1:123456789012:proxy:my-rds-proxy",  
  "eventName": "RDSProxyEvent",  
  "time": "2023-01-01T12:00:00Z",  
  "region": "us-east-1",  
  "resources": [{"resourceType": "DBProxy", "resourceARN": "arn:aws:rds:us-east-1:123456789012:proxy:my-rds-proxy"}],  
  "details": {"action": "ModifyDBProxy", "target": "my-endpoint", "status": "Success", "targetArn": "arn:aws:rds:us-east-1:123456789012:db-endpoint:my-endpoint"}}
```

```

    "id": "68f6e973-1a0c-d37b-f2f2-94a7f62ffd4e",
    "detail-type": "RDS DB Proxy Event",
    "source": "aws.rds",
    "account": "123456789012",
    "time": "2018-09-27T22:36:43Z",
    "region": "us-east-1",
    "resources": [
        "arn:aws:rds:us-east-1:123456789012:db-proxy:my-rds-proxy"
    ],
    "detail": {
        "EventCategories": [
            "configuration change"
        ],
        "SourceType": "DB_PROXY",
        "SourceArn": "arn:aws:rds:us-east-1:123456789012:db-proxy:my-rds-proxy",
        "Date": "2018-09-27T22:36:43.292Z",
        "Message": "RDS modified endpoint my-endpoint of DB Proxy my-rds-proxy.",
        "SourceIdentifier": "my-endpoint",
        "EventID": "RDS-EVENT-0207"
    }
}

```

RDS Proxy command-line examples

To see how combinations of connection commands and SQL statements interact with RDS Proxy, look at the following examples.

Examples

- [Preserving Connections to a MySQL Database Across a Failover](#)
- [Adjusting the max_connections Setting for an Aurora DB Cluster](#)

Example Preserving connections to a MySQL database across a failover

This MySQL example demonstrates how open connections continue working during a failover, for example when you reboot a database or it becomes unavailable due to a problem. This example uses a proxy named `the-proxy` and an Aurora DB cluster with DB instances `instance-8898` and `instance-9814`. When you run the `failover-db-cluster` command from the Linux command line, the writer instance that the proxy is connected to changes to a different DB instance. You can see that the DB instance associated with the proxy changes while the connection remains open.

```

$ mysql -h the-proxy.proxy-demo.us-east-1.rds.amazonaws.com -u admin_user -p
Enter password:
...
mysql> select @@aurora_server_id;
+-----+
| @@aurora_server_id |
+-----+
| instance-9814      |
+-----+
1 row in set (0.01 sec)

mysql>
[1]+  Stopped                  mysql -h the-proxy.proxy-demo.us-east-1.rds.amazonaws.com -
u admin_user -p
$ # Initially, instance-9814 is the writer.
$ aws rds failover-db-cluster --db-cluster-identifier cluster-56-2019-11-14-1399
JSON output
$ # After a short time, the console shows that the failover operation is complete.

```

```
$ # Now instance-8898 is the writer.
$ fg
mysql -h the-proxy.proxy-demo.us.us-east-1.rds.amazonaws.com -u admin_user -p

mysql> select @@aurora_server_id;
+-----+
| @@aurora_server_id |
+-----+
| instance-8898      |
+-----+
1 row in set (0.01 sec)

mysql>
[1]+  Stopped                  mysql -h the-proxy.proxy-demo.us-east-1.rds.amazonaws.com -
u admin_user -p
$ aws rds failover-db-cluster --db-cluster-identifier cluster-56-2019-11-14-1399
JSON output
$ # After a short time, the console shows that the failover operation is complete.
$ # Now instance-9814 is the writer again.
$ fg
mysql -h the-proxy.proxy-demo.us-east-1.rds.amazonaws.com -u admin_user -p

mysql> select @@aurora_server_id;
+-----+
| @@aurora_server_id |
+-----+
| instance-9814      |
+-----+
1 row in set (0.01 sec)
+-----+-----+
| Variable_name | Value      |
+-----+-----+
| hostname     | ip-10-1-3-178 |
+-----+-----+
1 row in set (0.02 sec)
```

Example Adjusting the max_connections setting for an Aurora DB cluster

This example demonstrates how you can adjust the max_connections setting for an Aurora MySQL DB cluster. To do so, you create your own DB cluster parameter group based on the default parameter settings for clusters that are compatible with MySQL 5.6 or 5.7. You specify a value for the max_connections setting, overriding the formula that sets the default value. You associate the DB cluster parameter group with your DB cluster.

```
export REGION=us-east-1
export CLUSTER_PARAM_GROUP=rds-proxy-mysql-56-max-connections-demo
export CLUSTER_NAME=rds-proxy-mysql-56

aws rds create-db-parameter-group --region $REGION \
--db-parameter-group-family aurora5.6 \
--db-parameter-group-name $CLUSTER_PARAM_GROUP \
--description "Aurora MySQL 5.6 cluster parameter group for RDS Proxy demo."

aws rds modify-db-cluster --region $REGION \
--db-cluster-identifier $CLUSTER_NAME \
--db-cluster-parameter-group-name $CLUSTER_PARAM_GROUP

echo "New cluster param group is assigned to cluster:"
aws rds describe-db-clusters --region $REGION \
--db-cluster-identifier $CLUSTER_NAME \
--query '*[*].{DBClusterParameterGroup:DBClusterParameterGroup}'

echo "Current value for max_connections:"
aws rds describe-db-cluster-parameters --region $REGION \
```

```
--db-cluster-parameter-group-name $CLUSTER_PARAM_GROUP \
--query '*[*].{ParameterName:ParameterName,ParameterValue:ParameterValue}' \
--output text | grep "max_connections"

echo -n "Enter number for max_connections setting: "
read answer

aws rds modify-db-cluster-parameter-group --region $REGION --db-cluster-parameter-group-
name $CLUSTER_PARAM_GROUP \
--parameters "ParameterName=max_connections,ParameterValue=$
$answer,ApplyMethod=immediate"

echo "Updated value for max_connections:"
aws rds describe-db-cluster-parameters --region $REGION \
--db-cluster-parameter-group-name $CLUSTER_PARAM_GROUP \
--query '*[*].{ParameterName:ParameterName,ParameterValue:ParameterValue}' \
--output text | grep "max_connections"
```

Troubleshooting for RDS Proxy

Following, you can find troubleshooting ideas for some common RDS Proxy issues and information on CloudWatch logs for RDS Proxy.

In the RDS Proxy logs, each entry is prefixed with the name of the associated proxy endpoint. This name can be the name you specified for a user-defined endpoint, or the special name `default` for read/write requests using the default endpoint of a proxy. For more information about proxy endpoints, see [Working with Amazon RDS Proxy endpoints \(p. 315\)](#).

Topics

- [Verifying connectivity for a proxy \(p. 332\)](#)
- [Common issues and solutions \(p. 333\)](#)

Verifying connectivity for a proxy

You can use the following commands to verify that all components of the connection mechanism can communicate with the other components.

Examine the proxy itself using the `describe-db-proxies` command. Also examine the associated target group using the `describe-db-proxy-target-groups`. Check that the details of the targets match the RDS DB instance or Aurora DB cluster that you intend to associate with the proxy. Use commands such as the following.

```
aws rds describe-db-proxies --db-proxy-name $DB_PROXY_NAME
aws rds describe-db-proxy-target-groups --db-proxy-name $DB_PROXY_NAME
```

To confirm that the proxy can connect to the underlying database, examine the targets specified in the target groups using the `describe-db-proxy-targets` command. Use a command such as the following.

```
aws rds describe-db-proxy-targets --db-proxy-name $DB_PROXY_NAME
```

The output of the `describe-db-proxy-targets` command includes a `TargetHealth` field. You can examine the fields `State`, `Reason`, and `Description` inside `TargetHealth` to check if the proxy can communicate with the underlying DB instance.

- A `State` value of `AVAILABLE` indicates that the proxy can connect to the DB instance.

- A State value of UNAVAILABLE indicates a temporary or permanent connection problem. In this case, examine the Reason and Description fields. For example, if Reason has a value of PENDING_PROXY_CAPACITY, try connecting again after the proxy finishes its scaling operation. If Reason has a value of UNREACHABLE, CONNECTION_FAILED, or AUTH_FAILURE, use the explanation from the Description field to help you diagnose the issue.
- The State field might have a value of REGISTERING for a brief time before changing to AVAILABLE or UNAVAILABLE.

If the following Netcat command (nc) is successful, you can access the proxy endpoint from the EC2 instance or other system where you're logged in. This command reports failure if you're not in the same VPC as the proxy and the associated database. You might be able to log directly in to the database without being in the same VPC. However, you can't log into the proxy unless you're in the same VPC.

```
nc -zx MySQL_proxy_endpoint 3306
nc -zx PostgreSQL_proxy_endpoint 5432
```

You can use the following commands to make sure that your EC2 instance has the required properties. In particular, the VPC for the EC2 instance must be the same as the VPC for the RDS DB instance or Aurora DB cluster that the proxy connects to.

```
aws ec2 describe-instances --instance-ids your_ec2_instance_id
```

Examine the Secrets Manager secrets used for the proxy.

```
aws secretsmanager list-secrets
aws secretsmanager get-secret-value --secret-id your_secret_id
```

Make sure that the SecretString field displayed by get-secret-value is encoded as a JSON string that includes username and password fields. The following example shows the format of the SecretString field.

```
{
    "ARN": "some_arn",
    "Name": "some_name",
    "VersionId": "some_version_id",
    "SecretString": '{"username":"some_username","password":"some_password"}',
    "VersionStages": [ "some_stage" ],
    "CreatedDate": some_timestamp
}
```

Common issues and solutions

For possible causes and solutions to some common problems that you might encounter using RDS Proxy, see the following.

You might encounter the following issues while creating a new proxy or connecting to a proxy.

Error	Causes or workarounds
403: The security token included in the request is invalid	Select an existing IAM role instead of choosing to create a new one.

You might encounter the following issues while connecting to a MySQL proxy.

Error	Causes or workarounds
ERROR 1040 (HY000): Connections rate limit exceeded (<i>limit_value</i>)	The rate of connection requests from the client to the proxy has exceeded the limit.
ERROR 1040 (HY000): IAM authentication rate limit exceeded	The number of simultaneous requests with IAM authentication from the client to the proxy has exceeded the limit.
ERROR 1040 (HY000): Number simultaneous connections exceeded (<i>limit_value</i>)	The number of simultaneous connection requests from the client to the proxy exceeded the limit.
ERROR 1045 (28000): Access denied for user ' <i>DB_USER</i> '@'%' (using password: YES)	<p>Some possible reasons include the following:</p> <ul style="list-style-type: none"> The Secrets Manager secret used by the proxy doesn't match the user name and password of an existing database user. Either update the credentials in the Secrets Manager secret, or make sure the database user exists and has the same password as in the secret.
ERROR 1105 (HY000): Unknown error	An unknown error occurred.
ERROR 1231 (42000): Variable 'character_set_client' can't be set to the value of <i>value</i>	The value set for the <code>character_set_client</code> parameter is not valid. For example, the value <code>ucs2</code> is not valid because it can crash the MySQL server.
ERROR 3159 (HY000): This RDS Proxy requires TLS connections.	<p>You enabled the setting Require Transport Layer Security in the proxy but your connection included the parameter <code>ssl-mode=DISABLED</code> in the MySQL client. Do either of the following:</p> <ul style="list-style-type: none"> Disable the setting Require Transport Layer Security for the proxy. Connect to the database using the minimum setting of <code>ssl-mode=REQUIRED</code> in the MySQL client.
ERROR 2026 (HY000): SSL connection error: Internal Server Error	<p>The TLS handshake to the proxy failed. Some possible reasons include the following:</p> <ul style="list-style-type: none"> SSL is required but the server doesn't support it. An internal server error occurred. A bad handshake occurred.
ERROR 9501 (HY000): Timed-out waiting to	The proxy timed-out waiting to acquire a database connection. Some possible reasons include the following:

Error	Causes or workarounds
acquire database connection	<ul style="list-style-type: none"> The proxy is unable to establish a database connection because the maximum connections have been reached The proxy is unable to establish a database connection because the database is unavailable.

You might encounter the following issues while connecting to a PostgreSQL proxy.

Error	Cause	Solution
IAM authentication is allowed only with SSL connections.	The user tried to connect to the database using IAM authentication with the setting <code>sslmode=disable</code> in the PostgreSQL client.	The user needs to connect to the database using the minimum setting of <code>sslmode=require</code> in the PostgreSQL client. For more information, see the PostgreSQL SSL support documentation.
This RDS Proxy requires TLS connections.	The user enabled the option Require Transport Layer Security but tried to connect with <code>sslmode=disable</code> in the PostgreSQL client.	To fix this error, do one of the following: <ul style="list-style-type: none"> Disable the proxy's Require Transport Layer Security option. Connect to the database using the minimum setting of <code>sslmode=allow</code> in the PostgreSQL client.
IAM authentication failed for user <code>user_name</code> . Check the IAM token for this user and try again.	This error might be due to the following reasons: <ul style="list-style-type: none"> The client supplied the incorrect IAM user name. The client supplied an incorrect IAM authorization token for the user. The client is using an IAM policy that does not have the necessary permissions. The client supplied an expired IAM authorization token for the user. 	To fix this error, do the following: <ol style="list-style-type: none"> Confirm that the provided IAM user exists. Confirm that the IAM authorization token belongs to the provided IAM user. Confirm that the IAM policy has adequate permissions for RDS. Check the validity of the IAM authorization token used.
This RDS proxy has no credentials for the role <code>role_name</code> . Check the credentials for this role and try again.	There is no Secrets Manager secret for this role.	Add a Secrets Manager secret for this role.
RDS supports only IAM or MD5 authentication.	The database client being used to connect to the proxy is using an authentication mechanism not currently supported by the proxy, such as SCRAM-SHA-256.	If you're not using IAM authentication, use the MD5 password authentication only.

Error	Cause	Solution
A user name is missing from the connection startup packet. Provide a user name for this connection.	The database client being used to connect to the proxy isn't sending a user name when trying to establish a connection.	Make sure to define a user name when setting up a connection to the proxy using the PostgreSQL client of your choice.
Feature not supported: RDS Proxy supports only version 3.0 of the PostgreSQL messaging protocol.	The PostgreSQL client used to connect to the proxy uses a protocol older than 3.0.	Use a newer PostgreSQL client that supports the 3.0 messaging protocol. If you're using the PostgreSQL <code>psql</code> CLI, use a version greater than or equal to 7.4.
Feature not supported: RDS Proxy currently doesn't support streaming replication mode.	The PostgreSQL client used to connect to the proxy is trying to use the streaming replication mode, which isn't currently supported by RDS Proxy.	Turn off the streaming replication mode in the PostgreSQL client being used to connect.
Feature not supported: RDS Proxy currently doesn't support the option <code>option_name</code> .	Through the startup message, the PostgreSQL client used to connect to the proxy is requesting an option that isn't currently supported by RDS Proxy.	Turn off the option being shown as not supported from the message above in the PostgreSQL client being used to connect.
The IAM authentication failed because of too many competing requests.	The number of simultaneous requests with IAM authentication from the client to the proxy has exceeded the limit.	Reduce the rate in which connections using IAM authentication from a PostgreSQL client are established.
The maximum number of client connections to the proxy exceeded <code>number_value</code> .	The number of simultaneous connection requests from the client to the proxy exceeded the limit.	Reduce the number of active connections from PostgreSQL clients to this RDS proxy.
Rate of connection to proxy exceeded <code>number_value</code> .	The rate of connection requests from the client to the proxy has exceeded the limit.	Reduce the rate in which connections from a PostgreSQL client are established.
The password that was provided for the role <code>role_name</code> is wrong.	The password for this role doesn't match the Secrets Manager secret.	Check the secret for this role in Secrets Manager to see if the password is the same as what's being used in your PostgreSQL client.
The IAM authentication failed for the role <code>role_name</code> . Check the IAM token for this role and try again.	There is a problem with the IAM token used for IAM authentication.	Generate a new authentication token and use it in a new connection.
IAM is allowed only with SSL connections.	A client tried to connect using IAM authentication, but SSL wasn't enabled.	Enable SSL in the PostgreSQL client.

Error	Cause	Solution
Unknown error.	An unknown error occurred.	Reach out to AWS Support to investigate the issue.
Timed-out waiting to acquire database connection.	The proxy timed-out waiting to acquire a database connection. Some possible reasons include the following: <ul style="list-style-type: none"> The proxy can't establish a database connection because the maximum connections have been reached. The proxy can't establish a database connection because the database is unavailable. 	Possible solutions are the following: <ul style="list-style-type: none"> Check the target of the RDS DB instance or Aurora DB cluster status to see if it's unavailable. Check if there are long-running transactions and/or queries being executed. They can use database connections from the connection pool for a long time.
Request returned an error: <i>database_error</i> .	The database connection established from the proxy returned an error.	The solution depends on the specific database error. One example is: Request returned an error: database "your-database-name" does not exist. This means the specified database name, or the user name used as a database name (in case a database name hasn't been specified), doesn't exist in the database server.

Using RDS Proxy with AWS CloudFormation

You can use RDS Proxy with AWS CloudFormation. Doing so helps you to create groups of related resources, including a proxy that can connect to a newly created Amazon RDS DB instance or Aurora DB cluster. RDS Proxy support in AWS CloudFormation involves two new registry types: `DBProxy` and `DBProxyTargetGroup`.

The following listing shows a sample AWS CloudFormation template for RDS Proxy.

```

Resources:
  DBProxy:
    Type: AWS::RDS::DBProxy
    Properties:
      DBProxyName: CanaryProxy
      EngineFamily: MYSQL
      RoleArn:
        Fn::ImportValue: SecretReaderRoleArn
      Auth:
        - {AuthScheme: SECRETS, SecretArn: !ImportValue ProxySecret, IAMAuth: DISABLED}
      VpcSubnetIds:
        Fn::Split: [",", "Fn::ImportValue": SubnetIds]

  ProxyTargetGroup:
    Type: AWS::RDS::DBProxyTargetGroup
    Properties:
      DBProxyName: CanaryProxy

```

```
TargetGroupName: default
DBInstanceIdentifiers:
  - Fn::ImportValue: DBInstanceName
DependsOn: DBProxy
```

For more information about the Amazon RDS and Aurora resources that you can create using AWS CloudFormation, see [RDS resource type reference](#).

Working with DB parameter groups and DB cluster parameter groups

Database parameters specify how the database is configured. For example, database parameters can specify the amount of resources, such as memory, to allocate to a database.

You manage your database configuration by associating your DB instances and Aurora DB clusters with parameter groups. Amazon RDS defines parameter groups with default settings.

Important

You can define your own parameter groups with customized settings. Then you can modify your DB instances and Aurora clusters to use your own parameter groups.

For information about modifying a DB cluster or DB instance, see [Modifying an Amazon Aurora DB cluster \(p. 372\)](#).

A *DB parameter group* acts as a container for engine configuration values that are applied to one or more DB instances. DB parameter groups apply to DB instances in both Amazon RDS and Aurora. These configuration settings apply to properties that can vary among the DB instances within an Aurora cluster, such as the sizes for memory buffers.

A *DB cluster parameter group* acts as a container for engine configuration values that are applied to every DB instance in an Aurora DB cluster. For example, the Aurora shared storage model requires that every DB instance in an Aurora cluster use the same setting for parameters such as `innodb_file_per_table`. Thus, parameters that affect the physical storage layout are part of the cluster parameter group. The DB cluster parameter group also includes default values for all the instance-level parameters

If you create a DB instance without specifying a DB parameter group, the DB instance uses a default DB parameter group. Likewise, if you create an Aurora DB cluster without specifying a DB cluster parameter group, the DB cluster uses a default DB cluster parameter group. Each default parameter group contains database engine defaults and Amazon RDS system defaults based on the engine, compute class, and allocated storage of the instance. You can't modify the parameter settings of a default parameter group. Instead, you create your own parameter group where you choose your own parameter settings. Not all DB engine parameters can be changed in a parameter group that you create.

If you want to use your own parameter group, you create a new parameter group and modify the parameters that you want to. You then modify your DB instance or DB cluster to use the new parameter group. If you update parameters within a DB parameter group, the changes apply to all DB instances that are associated with that parameter group. Likewise, if you update parameters within a DB cluster parameter group, the changes apply to all Aurora clusters that are associated with that DB cluster parameter group.

You can copy an existing DB parameter group with the AWS CLI [copy-db-parameter-group](#) command. You can copy an existing DB cluster parameter group with the AWS CLI [copy-db-cluster-parameter-group](#) command. Copying a parameter group can be convenient when you want to include most of an existing parameter group's custom parameters and values in a new parameter group.

Here are some important points about working with parameters in a parameter group:

- Database parameters are either *static* or *dynamic*. When you change a static parameter and save the DB parameter group, the parameter change takes effect after you manually reboot the DB instance. You can reboot a DB instance using the RDS console, by calling the `reboot-db-instance` CLI command, or by calling the `RebootDBInstance` API operation. The requirement to reboot the associated DB instance after a static parameter change helps mitigate the risk of a parameter misconfiguration affecting an API call, such as calling `ModifyDBInstance` to change DB instance class or scale storage.

When you change a dynamic parameter and save the DB parameter group, the change is applied to the parameter group immediately regardless of the **Apply Immediately** setting. If you use the pending-reboot setting in the AWS CLI or RDS API, the change is still applied to the parameter group immediately. However, applying the parameter change to DB instances that use the parameter group requires a reboot.

If a DB instance isn't using the latest changes to its associated DB parameter group, the AWS Management Console shows the DB parameter group with a status of **pending-reboot**. The **pending-reboot** parameter groups status doesn't result in an automatic reboot during the next maintenance window. To apply the latest parameter changes to that DB instance, manually reboot the DB instance.

- When you associate a new DB parameter group with a DB instance, the modified static and dynamic parameters are applied only after the DB instance is rebooted. However, if you modify dynamic parameters in the newly associated DB parameter group, these changes are applied immediately without a reboot. For more information about changing the DB parameter group, see [Modifying an Amazon Aurora DB cluster \(p. 372\)](#).

Note

After you change the DB cluster parameter group associated with a DB cluster, reboot the primary DB instance in the cluster to apply the changes to all of the DB instances in the cluster.

To determine whether the primary DB instance of a DB cluster must be rebooted to apply changes, run the following AWS CLI command:

```
aws rds describe-db-clusters --db-cluster-identifier  
db_cluster_identifier
```

Check the `DBClusterParameterGroupStatus` value for the primary DB instance in the output. If the value is `pending-reboot`, then reboot the primary DB instance of the DB cluster.

- You can specify integer and Boolean parameters using expressions, formulas, and functions. Functions can include a mathematical log expression. For more information, see [Specifying DB parameters \(p. 362\)](#).
- Set any parameters that relate to the character set or collation of your database in your parameter group before creating the DB instance and before you create a database in your DB instance. This ensures that the default database and new databases in your DB instance use the character set and collation values that you specify. If you change character set or collation parameters for your DB instance, the parameter changes are not applied to existing databases.

For some DB engines, you can change character set or collation values for an existing database using the `ALTER DATABASE` command, for example:

```
ALTER DATABASE database_name CHARACTER SET character_set_name COLLATE collation;
```

For more information about changing the character set or collation values for a database, check the documentation for your DB engine.

- Improperly setting parameters in a parameter group can have unintended adverse effects, including degraded performance and system instability. Always exercise caution when modifying database parameters and back up your data before modifying a parameter group. Try out parameter group setting changes on a test DB instance before applying those parameter group changes to a production DB instance.
- For an Aurora global database, you can specify different configuration settings for the individual Aurora clusters. Make sure that the settings are similar enough to produce consistent behavior if you promote a secondary cluster to be the primary cluster. For example, use the same settings for time zones and character sets across all the clusters of an Aurora global database.
- To determine the supported parameters for your DB engine, you can view the parameters in the DB parameter group and DB cluster parameter group used by the DB cluster. For more information, see

[Viewing parameter values for a DB parameter group \(p. 359\)](#) and [Viewing parameter values for a DB cluster parameter group \(p. 360\)](#).

Topics

- [Amazon Aurora DB cluster and DB instance parameters \(p. 341\)](#)
- [Creating a DB parameter group \(p. 342\)](#)
- [Creating a DB cluster parameter group \(p. 343\)](#)
- [Associating a DB parameter group with a DB instance \(p. 345\)](#)
- [Associating a DB cluster parameter group with a DB cluster \(p. 346\)](#)
- [Modifying parameters in a DB parameter group \(p. 347\)](#)
- [Modifying parameters in a DB cluster parameter group \(p. 349\)](#)
- [Resetting parameters in a DB parameter group to their default values \(p. 351\)](#)
- [Resetting parameters in a DB cluster parameter group \(p. 353\)](#)
- [Copying a DB parameter group \(p. 354\)](#)
- [Copying a DB cluster parameter group \(p. 356\)](#)
- [Listing DB parameter groups \(p. 357\)](#)
- [Listing DB cluster parameter groups \(p. 358\)](#)
- [Viewing parameter values for a DB parameter group \(p. 359\)](#)
- [Viewing parameter values for a DB cluster parameter group \(p. 360\)](#)
- [Comparing parameter groups \(p. 362\)](#)
- [Specifying DB parameters \(p. 362\)](#)

Amazon Aurora DB cluster and DB instance parameters

Aurora uses a two-level system of configuration settings, as follows:

- Parameters in a *DB cluster parameter group* apply to every DB instance in a DB cluster. Your data is stored in the Aurora shared storage subsystem. Because of this, all parameters related to physical layout of table data must be the same for all DB instances in an Aurora cluster. Likewise, because Aurora DB instances are connected by replication, all the parameters for replication settings must be identical throughout an Aurora cluster.
- Parameters in a *DB parameter group* apply to a single DB instance in an Aurora DB cluster. These parameters are related to aspects such as memory usage that you can vary across DB instances in the same Aurora cluster. For example, a cluster often contains DB instances with different AWS instance classes.

Every Aurora cluster is associated with a DB cluster parameter group. Each DB instance within the cluster inherits the settings from that DB cluster parameter group, and is associated with a DB parameter group. Aurora assigns default parameter groups when you create a cluster or a new DB instance, based on the specified database engine and version. You can change the parameter groups later to ones that you create, where you can edit the parameter values.

The DB cluster parameter groups also include default values for all the instance-level parameters from the DB parameter group. These defaults are mainly intended for configuring Aurora Serverless clusters, which are only associated with DB cluster parameter groups, not DB parameter groups. You can modify the instance-level parameter settings in the DB cluster parameter group. Then, Aurora applies those settings to each new DB instance that's added to a Serverless cluster. To learn more about configuration

settings for Aurora Serverless clusters and which settings you can modify, see [Parameter groups and Aurora Serverless v1 \(p. 156\)](#).

For non-Serverless clusters, any configuration values that you modify in the DB cluster parameter group override default values in the DB parameter group. If you edit the corresponding values in the DB parameter group, those values override the settings from the DB cluster parameter group.

Any DB parameter settings that you modify take precedence over the DB cluster parameter group values, even if you change the configuration parameters back to their default values. You can see which parameters are overridden by using the `describe-db-parameters` AWS CLI command or the `DescribeDBParameters` RDS API. The `Source` field contains the value `user` if you modified that parameter. To reset one or more parameters so that the value from the DB cluster parameter group takes precedence, use the `reset-db-parameter-group` AWS CLI command or the `ResetDBParameterGroup` RDS API operation.

The DB cluster and DB instance parameters available to you in Aurora vary depending on database engine compatibility.

Database engine	Parameters
Aurora MySQL	See Aurora MySQL configuration parameters (p. 1042) . For Aurora Serverless clusters, see additional details in Parameter groups and Aurora Serverless v1 (p. 156) .
Aurora PostgreSQL	See Amazon Aurora PostgreSQL parameters (p. 1547) .

Creating a DB parameter group

You can create a new DB parameter group using the AWS Management Console, the AWS CLI, or the RDS API.

Console

To create a DB parameter group

1. Sign in to the AWS Management Console and open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. In the navigation pane, choose **Parameter groups**.
3. Choose **Create parameter group**.
The **Create parameter group** window appears.
4. In the **Parameter group family** list, select a DB parameter group family.
5. In the **Type** list, select **DB Parameter Group**.
6. In the **Group name** box, enter the name of the new DB parameter group.
7. In the **Description** box, enter a description for the new DB parameter group.
8. Choose **Create**.

AWS CLI

To create a DB parameter group, use the AWS CLI `create-db-parameter-group` command. The following example creates a DB parameter group named `mydbparametergroup` for MySQL version 5.6 with a description of "My new parameter group."

Include the following required parameters:

- `--db-parameter-group-name`
- `--db-parameter-group-family`
- `--description`

To list all of the available parameter group families, use the following command:

```
aws rds describe-db-engine-versions --query "DBEngineVersions[].DBParameterGroupFamily"
```

Note

The output contains duplicates.

Example

For Linux, macOS, or Unix:

```
aws rds create-db-parameter-group \
--db-parameter-group-name mydbparametergroup \
--db-parameter-group-family aurora5.6 \
--description "My new parameter group"
```

For Windows:

```
aws rds create-db-parameter-group ^
--db-parameter-group-name mydbparametergroup ^
--db-parameter-group-family aurora5.6 ^
--description "My new parameter group"
```

This command produces output similar to the following:

```
DBPARAMETERGROUP  mydbparametergroup  aurora5.6  My new parameter group
```

RDS API

To create a DB parameter group, use the RDS API [CreateDBParameterGroup](#) operation.

Include the following required parameters:

- `DBParameterGroupName`
- `DBParameterGroupFamily`
- `Description`

Creating a DB cluster parameter group

You can create a new DB cluster parameter group using the AWS Management Console, the AWS CLI, or the RDS API.

Console

To create a DB cluster parameter group

1. Sign in to the AWS Management Console and open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.

2. In the navigation pane, choose **Parameter groups**.
3. Choose **Create parameter group**.
The **Create parameter group** window appears.
4. In the **Parameter group family** list, select a DB parameter group family
5. In the **Type** list, select **DB Cluster Parameter Group**.
6. In the **Group name** box, enter the name of the new DB cluster parameter group.
7. In the **Description** box, enter a description for the new DB cluster parameter group.
8. Choose **Create**.

AWS CLI

To create a DB cluster parameter group, use the AWS CLI [create-db-cluster-parameter-group](#) command. The following example creates a DB cluster parameter group named *mydbclusterparametergroup* for MySQL version 5.6 with a description of "My new cluster parameter group."

Include the following required parameters:

- `--db-cluster-parameter-group-name`
- `--db-parameter-group-family`
- `--description`

To list all of the available parameter group families, use the following command:

```
aws rds describe-db-engine-versions --query "DBEngineVersions[].DBParameterGroupFamily"
```

Note

The output contains duplicates.

Example

For Linux, macOS, or Unix:

```
aws rds create-db-cluster-parameter-group \
--db-cluster-parameter-group-name mydbclusterparametergroup \
--db-parameter-group-family aurora5.6 \
--description "My new cluster parameter group"
```

For Windows:

```
aws rds create-db-cluster-parameter-group ^
--db-cluster-parameter-group-name mydbclusterparametergroup ^
--db-parameter-group-family aurora5.6 ^
--description "My new cluster parameter group"
```

This command produces output similar to the following:

```
DBCLUSTERPARAMETERGROUP mydbclusterparametergroup mysql5.6 My cluster new parameter
group
```

RDS API

To create a DB cluster parameter group, use the RDS API [CreateDBClusterParameterGroup](#) action.

Include the following required parameters:

- `DBClusterParameterGroupName`
- `DBParameterGroupFamily`
- `Description`

Associating a DB parameter group with a DB instance

You can create your own DB parameter groups with customized settings. You can associate a DB parameter group with a DB instance using the AWS Management Console, the AWS CLI, or the RDS API. You can do so when you create or modify a DB instance.

For information about creating a DB parameter group, see [Creating a DB parameter group \(p. 342\)](#). For information about modifying a DB instance, see [Modify a DB instance in a DB cluster \(p. 373\)](#).

Note

When you associate a new DB parameter group with a DB instance, the modified static and dynamic parameters are applied only after the DB instance is rebooted. However, if you modify dynamic parameters in the newly associated DB parameter group, these changes are applied immediately without a reboot.

Console

To associate a DB parameter group with a DB instance

1. Sign in to the AWS Management Console and open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. In the navigation pane, choose **Databases**, and then choose the DB instance that you want to modify.
3. Choose **Modify**. The **Modify DB Instance** page appears.
4. Change the **DB parameter group** setting.
5. Choose **Continue** and check the summary of modifications.
6. (Optional) Choose **Apply immediately** to apply the changes immediately. Choosing this option can cause an outage in some cases.
7. On the confirmation page, review your changes. If they are correct, choose **Modify DB instance** to save your changes.

Or choose **Back** to edit your changes or **Cancel** to cancel your changes.

AWS CLI

To associate a DB parameter group with a DB instance, use the AWS CLI `modify-db-instance` command with the following options:

- `--db-instance-identifier`
- `--db-parameter-group-name`

The following example associates the `mydbpg` DB parameter group with the `database-1` DB instance. The changes are applied immediately by using `--apply-immediately`. Use `--no-apply-immediately` to apply the changes during the next maintenance window.

Example

For Linux, macOS, or Unix:

```
aws rds modify-db-instance \
--db-instance-identifier database-1 \
--db-parameter-group-name mydbpg \
--apply-immediately
```

For Windows:

```
aws rds modify-db-instance ^
--db-instance-identifier database-1 ^
--db-parameter-group-name mydbpg ^
--apply-immediately
```

RDS API

To associate a DB parameter group with a DB instance, use the RDS API [ModifyDBInstance](#) operation with the following parameters:

- `DBInstanceName`
- `DBParameterGroupName`

Associating a DB cluster parameter group with a DB cluster

You can create your own DB cluster parameter groups with customized settings. You can associate a DB cluster parameter group with a DB cluster using the AWS Management Console, the AWS CLI, or the RDS API. You can do so when you create or modify a DB cluster.

For information about creating a DB cluster parameter group, see [Creating a DB cluster parameter group \(p. 343\)](#). For information about creating a DB cluster, see [Creating an Amazon Aurora DB cluster \(p. 125\)](#). For information about modifying a DB cluster, see [Modifying an Amazon Aurora DB cluster \(p. 372\)](#).

Note

After you change the DB cluster parameter group associated with a DB cluster, reboot the primary DB instance in the cluster to apply the changes to all of the DB instances in the cluster.

To determine whether the primary DB instance of a DB cluster must be rebooted to apply changes, run the following AWS CLI command:

```
aws rds describe-db-clusters --db-cluster-identifier
db_cluster_identifier
```

Check the `DBClusterParameterGroupStatus` value for the primary DB instance in the output. If the value is `pending-reboot`, then reboot the primary DB instance of the DB cluster.

Console

To associate a DB cluster parameter group with a DB cluster

1. Sign in to the AWS Management Console and open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. In the navigation pane, choose **Databases**, and then select the DB cluster that you want to modify.

3. Choose **Modify**. The **Modify DB cluster** page appears.
4. Change the **DB cluster parameter group** setting.
5. Choose **Continue** and check the summary of modifications.

The change is applied immediately regardless of the **Scheduling of modifications** setting.

6. On the confirmation page, review your changes. If they are correct, choose **Modify cluster** to save your changes.

Alternatively, choose **Back** to edit your changes, or choose **Cancel** to cancel your changes.

AWS CLI

To associate a DB cluster parameter group with a DB cluster, use the AWS CLI [modify-db-cluster](#) command with the following options:

- `--db-cluster-name`
- `--db-cluster-parameter-group-name`

The following example associates the `mydbclpg` DB parameter group with the `mydbccluster` DB cluster.

Example

For Linux, macOS, or Unix:

```
aws rds modify-db-cluster \
    --db-cluster-identifier mydbccluster \
    --db-cluster-parameter-group-name mydbclpg
```

For Windows:

```
aws rds modify-db-cluster ^
    --db-cluster-identifier mydbccluster ^
    --db-cluster-parameter-group-name mydbclpg
```

RDS API

To associate a DB cluster parameter group with a DB cluster, use the RDS API [ModifyDBCluster](#) operation with the following parameters:

- `DBClusterIdentifier`
- `DBClusterParameterGroupName`

Modifying parameters in a DB parameter group

You can modify parameter values in a customer-created DB parameter group; you can't change the parameter values in a default DB parameter group. Changes to parameters in a customer-created DB parameter group are applied to all DB instances that are associated with the DB parameter group.

Changes to some parameters are applied to the DB instance immediately without a reboot. Changes to other parameters are applied only after the DB instance is rebooted. The RDS console shows the status of the DB parameter group associated with a DB instance on the **Configuration** tab. For example, if the DB instance isn't using the latest changes to its associated DB parameter group, the RDS console shows

the DB parameter group with a status of **pending-reboot**. To apply the latest parameter changes to that DB instance, manually reboot the DB instance.

The screenshot shows the AWS RDS console interface for managing a DB instance named 'cluster-2-instance-1'. The 'Configuration' tab is selected. In the 'Parameter group' section, the text 'test-aurora56-instance (pending-reboot)' is highlighted with a red box. Other visible details include the DB instance ID, engine version, and instance class.

DB identifier	Role	Engine	Engine version	Region & AZ
cluster-2	Regional	Aurora MySQL	5.6.10a	eu-central-1
cluster-2-instance-1	Writer	Aurora MySQL	5.6.10a	eu-central-1a

Configuration

- DB instance id: cluster-2-instance-1
- Engine version: 5.6.10a
- DB name: -
- Option groups: default:aurora-5-6
- ARN: arn:aws:rds:eu-central-1:...:db:cluster-2-instance-1
- Resource id: db-...
- Created time: Fri Apr 03 2020 10:48:37 GMT-0400 (Eastern Daylight Time)
- Parameter group: test-aurora56-instance (pending-reboot)

Instance class

- Instance class: db.t2.small
- vCPU: 1
- RAM: 2 GB

Availability

- Failover priority: 1

Console

To modify a DB parameter group

- Sign in to the AWS Management Console and open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
 - In the navigation pane, choose **Parameter groups**.
 - In the list, choose the parameter group that you want to modify.
 - For **Parameter group actions**, choose **Edit**.
 - Change the values of the parameters that you want to modify. You can scroll through the parameters using the arrow keys at the top right of the dialog box.
- You can't change values in a default parameter group.
- Choose **Save changes**.

AWS CLI

To modify a DB parameter group, use the AWS CLI [modify-db-parameter-group](#) command with the following required options:

- `--db-parameter-group-name`
- `--parameters`

The following example modifies the `max_connections` and `max_allowed_packet` values in the DB parameter group named *mydbparametergroup*.

Example

For Linux, macOS, or Unix:

```
aws rds modify-db-parameter-group \
    --db-parameter-group-name mydbparametergroup \
    --parameters "ParameterName=max_connections,ParameterValue=250,ApplyMethod=immediate" \
    "ParameterName=max_allowed_packet,ParameterValue=1024,ApplyMethod=immediate"
```

For Windows:

```
aws rds modify-db-parameter-group ^
    --db-parameter-group-name mydbparametergroup ^
    --parameters "ParameterName=max_connections,ParameterValue=250,ApplyMethod=immediate" ^
    "ParameterName=max_allowed_packet,ParameterValue=1024,ApplyMethod=immediate"
```

The command produces output like the following:

```
DBPARAMETERGROUP mydbparametergroup
```

RDS API

To modify a DB parameter group, use the RDS API [ModifyDBParameterGroup](#) operation with the following required parameters:

- `DBParameterGroupName`
- `Parameters`

Modifying parameters in a DB cluster parameter group

You can modify parameter values in a customer-created DB cluster parameter group. You can't change the parameter values in a default DB cluster parameter group. Changes to parameters in a customer-created DB cluster parameter group are applied to all DB clusters that are associated with the DB cluster parameter group.

Console

To modify a DB cluster parameter group

1. Sign in to the AWS Management Console and open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.

2. In the navigation pane, choose **Parameter groups**.
3. In the list, choose the parameter group that you want to modify.
4. For **Parameter group actions**, choose **Edit**.
5. Change the values of the parameters you want to modify. You can scroll through the parameters using the arrow keys at the top right of the dialog box.

You can't change values in a default parameter group.
6. Choose **Save changes**.
7. Reboot the primary DB instance in the cluster to apply the changes to all of the DB instances in the cluster.

AWS CLI

To modify a DB cluster parameter group, use the AWS CLI [modify-db-cluster-parameter-group](#) command with the following required parameters:

- `--db-cluster-parameter-group-name`
- `--parameters`

The following example modifies the `server_audit_logging` and `server_audit_logs_upload` values in the DB cluster parameter group named *mydbclusterparametergroup*.

Example

For Linux, macOS, or Unix:

```
aws rds modify-db-cluster-parameter-group \
  --db-cluster-parameter-group-name mydbclusterparametergroup \
  --parameters
  "ParameterName=server_audit_logging,ParameterValue=1,ApplyMethod=immediate" \
  "ParameterName=server_audit_logs_upload,ParameterValue=1,ApplyMethod=immediate"
```

For Windows:

```
aws rds modify-db-cluster-parameter-group ^
  --db-cluster-parameter-group-name mydbclusterparametergroup ^
  --parameters
  "ParameterName=server_audit_logging,ParameterValue=1,ApplyMethod=immediate" ^
  "ParameterName=server_audit_logs_upload,ParameterValue=1,ApplyMethod=immediate"
```

The command produces output like the following:

```
DBCLUSTERPARAMETERGROUP  mydbclusterparametergroup
```

RDS API

To modify a DB cluster parameter group, use the RDS API [ModifyDBClusterParameterGroup](#) command with the following required parameters:

- `DBClusterParameterGroupName`
- `Parameters`

Resetting parameters in a DB parameter group to their default values

You can reset parameter values in a customer-created DB parameter group to their default values. Changes to parameters in a customer-created DB parameter group are applied to all DB instances that are associated with the DB parameter group.

When you use the console, you can reset specific parameters to their default values, but you can't easily reset all of the parameters in the DB parameter group at once. When you use the AWS CLI or RDS API, you can reset specific parameters to their default values, and you can reset all of the parameters in the DB parameter group at once.

Changes to some parameters are applied to the DB instance immediately without a reboot. Changes to other parameters are applied only after the DB instance is rebooted. The RDS console shows the status of the DB parameter group associated with a DB instance on the **Configuration** tab. For example, if the DB instance isn't using the latest changes to its associated DB parameter group, the RDS console shows the DB parameter group with a status of **pending-reboot**. To apply the latest parameter changes to that DB instance, manually reboot the DB instance.

The screenshot shows the AWS RDS console interface. At the top, the navigation path is RDS > Databases > cluster-2 > cluster-2-instance-1. The main title is "cluster-2-instance-1". Below the title, there's a "Related" section with a search bar labeled "Filter databases". A table lists database identifiers: "cluster-2" (Regional, Aurora MySQL 5.6.10a, eu-central-1) and "cluster-2-instance-1" (Writer, Aurora MySQL 5.6.10a, eu-central-1). The "Configuration" tab is highlighted with a red box. Below this, the "Instance" section displays various instance details. The "Parameter group" field at the bottom of the instance details is also highlighted with a red box and contains the value "test-aurora56-instance (pending-reboot)".

Note

In a default DB parameter group, parameters are always set to their default values.

Console

To reset parameters in a DB parameter group to their default values

1. Sign in to the AWS Management Console and open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. In the navigation pane, choose **Parameter groups**.
3. In the list, choose the parameter group.
4. For **Parameter group actions**, choose **Edit**.
5. Choose the parameters that you want to reset to their default values. You can scroll through the parameters using the arrow keys at the top right of the dialog box.

You can't reset values in a default parameter group.
6. Choose **Reset** and then confirm by choosing **Reset parameters**.

AWS CLI

To reset some or all of the parameters in a DB parameter group, use the AWS CLI `reset-db-parameter-group` command with the following required option: `--db-parameter-group-name`.

To reset all of the parameters in the DB parameter group, specify the `--reset-all-parameters` option. To reset specific parameters, specify the `--parameters` option.

The following example resets all of the parameters in the DB parameter group named `mydbparametergroup` to their default values.

Example

For Linux, macOS, or Unix:

```
aws rds reset-db-parameter-group \
--db-parameter-group-name mydbparametergroup \
--reset-all-parameters
```

For Windows:

```
aws rds reset-db-parameter-group ^
--db-parameter-group-name mydbparametergroup ^
--reset-all-parameters
```

The following example resets the `max_connections` and `max_allowed_packet` options to their default values in the DB parameter group named `mydbparametergroup`.

Example

For Linux, macOS, or Unix:

```
aws rds reset-db-parameter-group \
--db-parameter-group-name mydbparametergroup \
--parameters "ParameterName=max_connections,ApplyMethod=immediate" \
"ParameterName=max_allowed_packet,ApplyMethod=immediate"
```

For Windows:

```
aws rds reset-db-parameter-group ^
--db-parameter-group-name mydbparametergroup ^
--parameters "ParameterName=max_connections,ApplyMethod=immediate" ^
"ParameterName=max_allowed_packet,ApplyMethod=immediate"
```

The command produces output like the following:

```
DBParameterGroupName mydbparametergroup
```

RDS API

To reset parameters in a DB parameter group to their default values, use the RDS API [ResetDBParameterGroup](#) command with the following required parameter: `DBParameterGroupName`.

To reset all of the parameters in the DB parameter group, set the `ResetAllParameters` parameter to `true`. To reset specific parameters, specify the `Parameters` parameter.

Resetting parameters in a DB cluster parameter group

You can reset parameters to their default values in a customer-created DB cluster parameter group. Changes to parameters in a customer-created DB cluster parameter group are applied to all DB clusters that are associated with the DB cluster parameter group.

Note

In a default DB cluster parameter group, parameters are always set to their default values.

Console

To reset parameters in a DB cluster parameter group to their default values

1. Sign in to the AWS Management Console and open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. In the navigation pane, choose **Parameter groups**.
3. In the list, choose the parameter group.
4. For **Parameter group actions**, choose **Edit**.
5. Choose the parameters that you want to reset to their default values. You can scroll through the parameters using the arrow keys at the top right of the dialog box.

You can't reset values in a default parameter group.
6. Choose **Reset** and then confirm by choosing **Reset parameters**.
7. Reboot the primary DB instance in the DB cluster to apply the changes to all of the DB instances in the DB cluster.

AWS CLI

To reset parameters in a DB cluster parameter group to their default values, use the AWS CLI [reset-db-cluster-parameter-group](#) command with the following required option: `--db-cluster-parameter-group-name`.

To reset all of the parameters in the DB cluster parameter group, specify the `--reset-all-parameters` option. To reset specific parameters, specify the `--parameters` option.

The following example resets all of the parameters in the DB parameter group named *mydbparametergroup* to their default values.

Example

For Linux, macOS, or Unix:

```
aws rds reset-db-cluster-parameter-group \
--db-cluster-parameter-group-name mydbparametergroup \
--reset-all-parameters
```

For Windows:

```
aws rds reset-db-cluster-parameter-group ^
--db-cluster-parameter-group-name mydbparametergroup ^
--reset-all-parameters
```

The following example resets the `server_audit_logging` and `server_audit_logs_upload` to their default values in the DB cluster parameter group named *mydbclusterparametergroup*.

Example

For Linux, macOS, or Unix:

```
aws rds reset-db-cluster-parameter-group \
--db-cluster-parameter-group-name mydbclusterparametergroup \
--parameters "ParameterName=server_audit_logging,ApplyMethod=immediate" \
"ParameterName=server_audit_logs_upload,ApplyMethod=immediate"
```

For Windows:

```
aws rds reset-db-cluster-parameter-group ^
--db-cluster-parameter-group-name mydbclusterparametergroup ^
--parameters
"ParameterName=server_audit_logging,ParameterValue=1,ApplyMethod=immediate" ^
"ParameterName=server_audit_logs_upload,ParameterValue=1,ApplyMethod=immediate"
```

The command produces output like the following:

```
DBClusterParameterGroupName mydbclusterparametergroup
```

RDS API

To reset parameters in a DB cluster parameter group to their default values, use the RDS API [ResetDBClusterParameterGroup](#) command with the following required parameter: `DBClusterParameterGroupName`.

To reset all of the parameters in the DB cluster parameter group, set the `ResetAllParameters` parameter to `true`. To reset specific parameters, specify the `Parameters` parameter.

Copying a DB parameter group

You can copy custom DB parameter groups that you create. Copying a parameter group is a convenient solution when you have already created a DB parameter group and you want to include most of the

custom parameters and values from that group in a new DB parameter group. You can copy a DB parameter group by using the AWS Management Console, the AWS CLI [copy-db-parameter-group](#) command, or the RDS API [CopyDBParameterGroup](#) operation.

After you copy a DB parameter group, wait at least 5 minutes before creating your first DB instance that uses that DB parameter group as the default parameter group. Doing this allows Amazon RDS to fully complete the copy action before the parameter group is used. This is especially important for parameters that are critical when creating the default database for a DB instance. An example is the character set for the default database defined by the `character_set_database` parameter. Use the **Parameter Groups** option of the [Amazon RDS console](#) or the [describe-db-parameters](#) command to verify that your DB parameter group is created.

Note

You can't copy a default parameter group. However, you can create a new parameter group that is based on a default parameter group.

Currently, you can't copy a parameter group to a different AWS Region.

Console

To copy a DB parameter group

1. Sign in to the AWS Management Console and open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. In the navigation pane, choose **Parameter groups**.
3. In the list, choose the custom parameter group that you want to copy.
4. For **Parameter group actions**, choose **Copy**.
5. In **New DB parameter group identifier**, enter a name for the new parameter group.
6. In **Description**, enter a description for the new parameter group.
7. Choose **Copy**.

AWS CLI

To copy a DB parameter group, use the AWS CLI [copy-db-parameter-group](#) command with the following required options:

- `--source-db-parameter-group-identifier`
- `--target-db-parameter-group-identifier`
- `--target-db-parameter-group-description`

The following example creates a new DB parameter group named `mygroup2` that is a copy of the DB parameter group `mygroup1`.

Example

For Linux, macOS, or Unix:

```
aws rds copy-db-parameter-group \
--source-db-parameter-group-identifier mygroup1 \
--target-db-parameter-group-identifier mygroup2 \
--target-db-parameter-group-description "DB parameter group 2"
```

For Windows:

```
aws rds copy-db-parameter-group ^
```

```
--source-db-parameter-group-identifier mygroup1 ^
--target-db-parameter-group-identifier mygroup2 ^
--target-db-parameter-group-description "DB parameter group 2"
```

RDS API

To copy a DB parameter group, use the RDS API [CopyDBParameterGroup](#) operation with the following required parameters:

- `SourceDBParameterGroupIdentifier`
- `TargetDBParameterGroupIdentifier`
- `TargetDBParameterGroupDescription`

Copying a DB cluster parameter group

You can copy custom DB cluster parameter groups that you create. Copying a parameter group is a convenient solution when you have already created a DB cluster parameter group and you want to include most of the custom parameters and values from that group in a new DB cluster parameter group. You can copy a DB cluster parameter group by using the AWS CLI [copy-db-cluster-parameter-group](#) command or the RDS API [CopyDBClusterParameterGroup](#) operation.

After you copy a DB cluster parameter group, wait at least 5 minutes before creating your first DB cluster that uses that DB cluster parameter group as the default parameter group. Doing this allows Amazon RDS to fully complete the copy action before the parameter group is used as the default for a new DB cluster. You can use the **Parameter Groups** option of the [Amazon RDS console](#) or the `describe-db-cluster-parameters` command to verify that your DB cluster parameter group is created.

Note

You can't copy a default parameter group. However, you can create a new parameter group that is based on a default parameter group.

Console

To copy a DB cluster parameter group

1. Sign in to the AWS Management Console and open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. In the navigation pane, choose **Parameter groups**.
3. In the list, choose the custom parameter group that you want to copy.
4. For **Parameter group actions**, choose **Copy**.
5. In **New DB parameter group identifier**, enter a name for the new parameter group.
6. In **Description**, enter a description for the new parameter group.
7. Choose **Copy**.

AWS CLI

To copy a DB cluster parameter group, use the AWS CLI [copy-db-cluster-parameter-group](#) command with the following required parameters:

- `--source-db-cluster-parameter-group-identifier`
- `--target-db-cluster-parameter-group-identifier`
- `--target-db-cluster-parameter-group-description`

The following example creates a new DB cluster parameter group named `mygroup2` that is a copy of the DB cluster parameter group `mygroup1`.

Example

For Linux, macOS, or Unix:

```
aws rds copy-db-cluster-parameter-group \
--source-db-cluster-parameter-group-identifier mygroup1 \
--target-db-cluster-parameter-group-identifier mygroup2 \
--target-db-cluster-parameter-group-description "DB parameter group 2"
```

For Windows:

```
aws rds copy-db-cluster-parameter-group ^
--source-db-cluster-parameter-group-identifier mygroup1 ^
--target-db-cluster-parameter-group-identifier mygroup2 ^
--target-db-cluster-parameter-group-description "DB parameter group 2"
```

RDS API

To copy a DB cluster parameter group, use the RDS API [CopyDBClusterParameterGroup](#) operation with the following required parameters:

- `SourceDBClusterParameterGroupIdentifier`
- `TargetDBClusterParameterGroupIdentifier`
- `TargetDBClusterParameterGroupDescription`

Listing DB parameter groups

You can list the DB parameter groups you've created for your AWS account.

Note

Default parameter groups are automatically created from a default parameter template when you create a DB instance for a particular DB engine and version. These default parameter groups contain preferred parameter settings and can't be modified. When you create a custom parameter group, you can modify parameter settings.

Console

To list all DB parameter groups for an AWS account

1. Sign in to the AWS Management Console and open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. In the navigation pane, choose **Parameter groups**.

The DB parameter groups appear in a list.

AWS CLI

To list all DB parameter groups for an AWS account, use the AWS CLI [describe-db-parameter-groups](#) command.

Example

The following example lists all available DB parameter groups for an AWS account.

```
aws rds describe-db-parameter-groups
```

The command returns a response like the following:

```
DBPARAMETERGROUP default.mysql5.6      mysql5.6  Default parameter group for MySQL5.6
DBPARAMETERGROUP mydbparametergroup     mysql5.6  My new parameter group
```

The following example describes the *mydbparamgroup1* parameter group.

For Linux, macOS, or Unix:

```
aws rds describe-db-parameter-groups \
--db-parameter-group-name mydbparamgroup1
```

For Windows:

```
aws rds describe-db-parameter-groups ^
--db-parameter-group-name mydbparamgroup1
```

The command returns a response like the following:

```
DBPARAMETERGROUP mydbparametergroup1  mysql5.6  My new parameter group
```

RDS API

To list all DB parameter groups for an AWS account, use the RDS API [DescribeDBParameterGroups](#) operation.

Listing DB cluster parameter groups

You can list the DB cluster parameter groups you've created for your AWS account.

Note

Default parameter groups are automatically created from a default parameter template when you create a DB cluster for a particular DB engine and version. These default parameter groups contain preferred parameter settings and can't be modified. When you create a custom parameter group, you can modify parameter settings.

Console

To list all DB cluster parameter groups for an AWS account

1. Sign in to the AWS Management Console and open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. In the navigation pane, choose **Parameter groups**.

The DB cluster parameter groups appear in the list with **DB cluster parameter group** for Type.

AWS CLI

To list all DB cluster parameter groups for an AWS account, use the AWS CLI [describe-db-cluster-parameter-groups](#) command.

Example

The following example lists all available DB cluster parameter groups for an AWS account.

```
aws rds describe-db-cluster-parameter-groups
```

The command returns a response like the following:

```
DBCLUSTERPARAMETERGROUPS      arn:aws:rds:us-west-2:1234567890:cluster-
pg:default.aurora5.6 default.aurora5.6      aurora5.6      Default cluster parameter
group for aurora5.6
DBCLUSTERPARAMETERGROUPS      arn:aws:rds:us-west-2:1234567890:cluster-
pg:mydbclusterparametergroup mydbclusterparametergroup      aurora5.6      My new cluster
parameter group
```

The following example describes the *mydbclusterparametergroup* parameter group.

For Linux, macOS, or Unix:

```
aws rds describe-db-cluster-parameter-groups \
--db-cluster-parameter-group-name mydbclusterparametergroup
```

For Windows:

```
aws rds describe-db-cluster-parameter-groups ^
--db-cluster-parameter-group-name mydbclusterparametergroup
```

The command returns a response like the following:

```
DBCLUSTERPARAMETERGROUPS      arn:aws:rds:us-west-2:1234567890:cluster-
pg:mydbclusterparametergroup mydbclusterparametergroup      aurora5.6      My new cluster
parameter group
```

RDS API

To list all DB cluster parameter groups for an AWS account, use the RDS API [DescribeDBClusterParameterGroups](#) action.

Viewing parameter values for a DB parameter group

You can get a list of all parameters in a DB parameter group and their values.

Console

To view the parameter values for a DB parameter group

1. Sign in to the AWS Management Console and open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. In the navigation pane, choose **Parameter groups**.

The DB parameter groups appear in a list.

3. Choose the name of the parameter group to see its list of parameters.

AWS CLI

To view the parameter values for a DB parameter group, use the AWS CLI [describe-db-parameters](#) command with the following required parameter.

- `--db-parameter-group-name`

Example

The following example lists the parameters and parameter values for a DB parameter group named *mydbparametergroup*.

```
aws rds describe-db-parameters --db-parameter-group-name mydbparametergroup
```

The command returns a response like the following:

DBPARAMETER	Parameter Name	Parameter Value	Source	Data Type	Apply
DBPARAMETER	Type	Is Modifiable			
DBPARAMETER	allow-suspicious-udfs	false	engine-default	boolean	static
DBPARAMETER	auto_increment_increment	true	engine-default	integer	dynamic
DBPARAMETER	auto_increment_offset	true	engine-default	integer	dynamic
DBPARAMETER	binlog_cache_size	32768	system	integer	dynamic
DBPARAMETER	socket	/tmp/mysql.sock	system	string	static

RDS API

To view the parameter values for a DB parameter group, use the RDS API [DescribeDBParameters](#) command with the following required parameter.

- `DBParameterGroupName`

Viewing parameter values for a DB cluster parameter group

You can get a list of all parameters in a DB cluster parameter group and their values.

Console

To view the parameter values for a DB cluster parameter group

1. Sign in to the AWS Management Console and open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. In the navigation pane, choose **Parameter groups**.

The DB cluster parameter groups appear in the list with **DB cluster parameter group** for **Type**.

3. Choose the name of the DB cluster parameter group to see its list of parameters.

AWS CLI

To view the parameter values for a DB cluster parameter group, use the AWS CLI [describe-db-cluster-parameters](#) command with the following required parameter.

- `--db-cluster-parameter-group-name`

Example

The following example lists the parameters and parameter values for a DB cluster parameter group named *mydbcclusterparametergroup*, in JSON format.

The command returns a response like the following:

```
aws rds describe-db-cluster-parameters --db-cluster-parameter-group-name mydbcclusterparametergroup
```

```
{  
    "Parameters": [  
        {  
            "ApplyMethod": "pending-reboot",  
            "Description": "Controls whether user-defined functions that have only an xxx symbol for the main function can be loaded",  
            "DataType": "boolean",  
            "AllowedValues": "0,1",  
            "SupportedEngineModes": [  
                "provisioned"  
            ],  
            "Source": "engine-default",  
            "IsModifiable": false,  
            "ParameterName": "allow-suspicious-udfs",  
            "ApplyType": "static"  
        },  
        {  
            "ApplyMethod": "pending-reboot",  
            "Description": "Enables new features in the Aurora engine.",  
            "DataType": "boolean",  
            "IsModifiable": true,  
            "AllowedValues": "0,1",  
            "SupportedEngineModes": [  
                "provisioned"  
            ],  
            "Source": "engine-default",  
            "ParameterValue": "0",  
            "ParameterName": "aurora_lab_mode",  
            "ApplyType": "static"  
        },  
        ...  
    ]  
}
```

The following example lists the parameters and parameter values for a DB cluster parameter group named *mydbcclusterparametergroup*, in plain text format.

```
aws rds describe-db-cluster-parameters --db-cluster-parameter-group-name mydbcclusterparametergroup --output text
```

The command returns a response like the following:

```
PARAMETERS 0,1 pending-reboot static boolean Controls whether user-defined functions
  that have only an xxx symbol for the main function can be loaded False allow-suspicious-
  udfs engine-default SUPPORTEDENGINEMODES provisioned
PARAMETERS 0,1 pending-reboot static boolean Enables new features in the Aurora
  engine. True aurora_lab_mode 0 engine-default SUPPORTEDENGINEMODES provisioned
...
```

RDS API

To view the parameter values for a DB cluster parameter group, use the RDS API [DescribeDBClusterParameters](#) command with the following required parameter.

- `DBClusterParameterGroupName`

Comparing parameter groups

You can use the AWS Management Console to view the differences between two parameter groups for the same DB engine and version.

To compare two parameter groups

1. Sign in to the AWS Management Console and open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. In the navigation pane, choose **Parameter groups**.
3. In the list, choose the two parameter groups that you want to compare.
4. For **Parameter group actions**, choose **Compare**.

Note

If the items you selected aren't equivalent, you can't choose **Compare**. For example, you can't compare a MySQL 5.6 and a MySQL 5.7 parameter group. You can't compare a DB parameter group and an Aurora DB cluster parameter group.

Specifying DB parameters

DB parameter types include the following:

- Integer
- Boolean
- String
- Long
- Double
- Timestamp
- Object of other defined data types
- Array of values of type integer, Boolean, string, long, double, timestamp, or object

You can also specify integer and Boolean parameters using expressions, formulas, and functions.

Contents

- [DB parameter formulas \(p. 363\)](#)
 - [DB parameter formula variables \(p. 363\)](#)
 - [DB parameter formula operators \(p. 363\)](#)
- [DB parameter functions \(p. 364\)](#)

- [DB parameter log expressions \(p. 364\)](#)
- [DB parameter value examples \(p. 365\)](#)

DB parameter formulas

A DB parameter formula is an expression that resolves to an integer value or a Boolean value. You enclose the expression in braces: {}. You can use a formula for either a DB parameter value or as an argument to a DB parameter function.

Syntax

```
{FormulaVariable}  
{FormulaVariable*Integer}  
{FormulaVariable*Integer/Integer}  
{FormulaVariable/Integer}
```

DB parameter formula variables

Each formula variable returns an integer or a Boolean value. The names of the variables are case-sensitive.

AllocatedStorage

Returns an integer representing the size, in bytes, of the data volume.

DBInstanceClassMemory

Returns an integer for the number of bytes of memory available to the database process. This number is internally calculated by taking the total amount of memory for the DB instance class and subtracting memory reserved for the operating system and the RDS processes that manage the instance. Therefore, the number is always somewhat lower than the memory figures shown in the instance class tables in [Aurora DB instance classes \(p. 54\)](#). The exact value depends on a combination of instance class, DB engine, and whether it applies to an RDS instance or an instance that's part of an Aurora cluster.

EndPointPort

Returns an integer representing the port used when connecting to the DB instance.

DB parameter formula operators

DB parameter formulas support two operators: division and multiplication.

Division operator: /

Divides the dividend by the divisor, returning an integer quotient. Decimals in the quotient are truncated, not rounded.

Syntax

```
dividend / divisor
```

The dividend and divisor arguments must be integer expressions.

*Multiplication operator: **

Multiplies the expressions, returning the product of the expressions. Decimals in the expressions are truncated, not rounded.

Syntax

```
expression * expression
```

Both expressions must be integers.

DB parameter functions

You specify the arguments of DB parameter functions as either integers or formulas. Each function must have at least one argument. Specify multiple arguments as a comma-separated list. The list can't have any empty members, such as *argument1,,argument3*. Function names are case-insensitive.

IF

Returns an argument.

Syntax

```
IF(argument1, argument2, argument3)
```

Returns the second argument if the first argument evaluates to true. Returns the third argument otherwise.

GREATEST

Returns the largest value from a list of integers or parameter formulas.

Syntax

```
GREATEST(argument1, argument2,...argumentn)
```

Returns an integer.

LEAST

Returns the smallest value from a list of integers or parameter formulas.

Syntax

```
LEAST(argument1, argument2,...argumentn)
```

Returns an integer.

SUM

Adds the values of the specified integers or parameter formulas.

Syntax

```
SUM(argument1, argument2,...argumentn)
```

Returns an integer.

DB parameter log expressions

You can set an integer DB parameter value to a log expression. You enclose the expression in braces: {}.

For example:

```
{log(DBInstanceClassMemory/8187281418)*1000}
```

The `log` function represents log base 2. This example also uses the `DBInstanceClassMemory` formula variable. See [DB parameter formula variables \(p. 363\)](#).

DB parameter value examples

These examples show using formulas, functions, and expressions for the values of DB parameters.

Note

DB Parameter functions are currently supported only in the console and aren't supported in the AWS CLI.

Warning

Improperly setting parameters in a DB parameter group can have unintended adverse effects. These might include degraded performance and system instability. Use caution when modifying database parameters and back up your data before modifying your DB parameter group. Try out parameter group changes on a test DB instance, created using point-in-time-restores, before applying those parameter group changes to your production DB instances.

Example using the DB parameter function LEAST

You can specify the `LEAST` function in an Aurora MySQL `table_definition_cache` parameter value. Use it to set the number of table definitions that can be stored in the definition cache to the lesser of `DBInstanceClassMemory/393040` or 20,000.

```
LEAST({DBInstanceClassMemory/393040}, 20000)
```

Migrating data to an Amazon Aurora DB cluster

You have several options for migrating data from your existing database to an Amazon Aurora DB cluster, depending on database engine compatibility. Your migration options also depend on the database that you are migrating from and the size of the data that you are migrating.

Migrating data to an Amazon Aurora MySQL DB cluster

You can migrate data from one of the following sources to an Amazon Aurora MySQL DB cluster.

- An RDS for MySQL DB instance
- A MySQL database external to Amazon RDS
- A database that is not MySQL-compatible

For more information, see [Migrating data to an Amazon Aurora MySQL DB cluster \(p. 781\)](#).

Migrating data to an Amazon Aurora PostgreSQL DB cluster

You can migrate data from one of the following sources to an Amazon Aurora PostgreSQL DB cluster.

- An Amazon RDS PostgreSQL DB instance
- A database that is not PostgreSQL-compatible

For more information, see [Migrating data to Amazon Aurora with PostgreSQL compatibility \(p. 1283\)](#).

Managing an Amazon Aurora DB cluster

This section shows how to manage and maintain your Aurora DB cluster. Aurora involves clusters of database servers that are connected in a replication topology. Thus, managing Aurora often involves deploying changes to multiple servers and making sure that all Aurora Replicas are keeping up with the master server. Because Aurora transparently scales the underlying storage as your data grows, managing Aurora requires relatively little management of disk storage. Likewise, because Aurora automatically performs continuous backups, an Aurora cluster does not require extensive planning or downtime for performing backups.

Topics

- [Stopping and starting an Amazon Aurora DB cluster \(p. 368\)](#)
- [Modifying an Amazon Aurora DB cluster \(p. 372\)](#)
- [Adding Aurora Replicas to a DB cluster \(p. 392\)](#)
- [Managing performance and scaling for Aurora DB clusters \(p. 396\)](#)
- [Cloning a volume for an Aurora DB cluster \(p. 402\)](#)
- [Integrating Aurora with other AWS services \(p. 426\)](#)
- [Maintaining an Amazon Aurora DB cluster \(p. 443\)](#)
- [Rebooting an Amazon Aurora DB cluster or Amazon Aurora DB instance \(p. 451\)](#)
- [Deleting Aurora DB clusters and DB instances \(p. 467\)](#)
- [Tagging Amazon RDS resources \(p. 474\)](#)
- [Working with Amazon Resource Names \(ARNs\) in Amazon RDS \(p. 482\)](#)
- [Amazon Aurora updates \(p. 489\)](#)

Stopping and starting an Amazon Aurora DB cluster

Stopping and starting Amazon Aurora clusters helps you manage costs for development and test environments. You can temporarily stop all the DB instances in your cluster, instead of setting up and tearing down all the DB instances each time that you use the cluster.

Topics

- [Overview of stopping and starting an Aurora DB cluster \(p. 368\)](#)
- [Limitations for stopping and starting Aurora DB clusters \(p. 368\)](#)
- [Stopping an Aurora DB cluster \(p. 369\)](#)
- [Possible operations while an Aurora DB cluster is stopped \(p. 370\)](#)
- [Starting an Aurora DB cluster \(p. 370\)](#)

Overview of stopping and starting an Aurora DB cluster

During periods where you don't need an Aurora cluster, you can stop all instances in that cluster at once. You can start the cluster again anytime you need to use it. Starting and stopping simplifies the setup and teardown processes for clusters used for development, testing, or similar activities that don't require continuous availability. You can perform all the AWS Management Console procedures involved with only a single action, regardless of how many instances are in the cluster.

While your DB cluster is stopped, you are charged only for cluster storage, manual snapshots, and automated backup storage within your specified retention window. You aren't charged for any DB instance hours. Aurora automatically starts your DB cluster after seven days so that it doesn't fall behind any required maintenance updates.

To minimize charges for a lightly loaded Aurora cluster, you can stop the cluster instead of deleting all of its Aurora Replicas. For clusters with more than one or two instances, frequently deleting and recreating the DB instances is only practical using the AWS CLI or Amazon RDS API. Such a sequence of operations can also be difficult to perform in the right order, for example to delete all Aurora Replicas before deleting the primary instance to avoid activating the failover mechanism.

Don't use starting and stopping if you need to keep your DB cluster running but it has more capacity than you need. If your cluster is too costly or not very busy, delete one or more DB instances or change all your DB instances to a small instance class. You can't stop an individual Aurora DB instance.

Limitations for stopping and starting Aurora DB clusters

Some Aurora clusters can't be stopped and started:

- You can't stop and start a cluster that's part of an [Aurora global database \(p. 225\)](#).
- For a cluster that uses the [Aurora parallel query \(p. 881\)](#) feature, the minimum Aurora MySQL versions are 1.23.0 and 2.09.0.
- You can't stop and start an [Aurora Serverless cluster \(p. 147\)](#).
- You can't stop and start an [Aurora multi-master cluster \(p. 958\)](#).

If an existing cluster can't be stopped and started, the **Stop** action isn't available from the **Actions** menu on the **Databases** page or the details page.

Stopping an Aurora DB cluster

To use an Aurora DB cluster or perform administration, you always begin with a running Aurora DB cluster, then stop the cluster, and then start the cluster again. While your cluster is stopped, you are charged for cluster storage, manual snapshots, and automated backup storage within your specified retention window, but not for DB instance hours.

The stop operation stops the Aurora Replica instances first, then the primary instance, to avoid activating the failover mechanism.

You can't stop a DB cluster that acts as the replication target for data from another DB cluster, or acts as the replication master and transmits data to another cluster.

You can't stop certain special kinds of clusters. Currently, you can't stop a cluster that's part of an Aurora global database, or a multi-master cluster.

Console

To stop an Aurora cluster

1. Sign in to the AWS Management Console and open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. In the navigation pane, choose **Databases**, and then choose a cluster. You can perform the stop operation from this page, or navigate to the details page for the DB cluster that you want to stop.

If an existing cluster can't be stopped and started, the **Stop** action isn't available from the **Actions** menu on the **Databases** page or the details page. For the kinds of clusters that you can't start and stop, see [Limitations for stopping and starting Aurora DB clusters \(p. 368\)](#).
3. For **Actions**, choose **Stop**.

AWS CLI

To stop a DB instance by using the AWS CLI, call the `stop-db-cluster` command with the following parameters:

- `--db-cluster-identifier` – the name of the Aurora cluster.

Example

```
aws rds stop-db-cluster --db-cluster-identifier mydbcluster
```

RDS API

To stop a DB instance by using the Amazon RDS API, call the `StopDBCluster` operation with the following parameter:

- `DBClusterIdentifier` – the name of the Aurora cluster.

Possible operations while an Aurora DB cluster is stopped

While an Aurora cluster is stopped, you can do a point-in-time restore to any point within your specified automated backup retention window. For details about doing a point-in-time restore, see [Restoring data \(p. 493\)](#).

You can't modify the configuration of an Aurora DB cluster, or any of its DB instances, while the cluster is stopped. You also can't add or remove DB instances from the cluster, or delete the cluster if it still has any associated DB instances. You must start the cluster before performing any such administrative actions.

Stopping a DB cluster removes pending actions, except for the DB cluster parameter group or for the DB parameter groups of the DB cluster instances.

Aurora applies any scheduled maintenance to your stopped cluster after it's started again. Remember that after seven days, Aurora automatically starts any stopped clusters so that they don't fall too far behind in their maintenance status.

Aurora also doesn't perform any automated backups, because the underlying data can't change while the cluster is stopped. Aurora doesn't extend the backup retention period while the cluster is stopped.

Starting an Aurora DB cluster

You always start an Aurora DB cluster beginning with an Aurora cluster that is already in the stopped state. When you start the cluster, all its DB instances become available again. The cluster keeps its configuration settings such as endpoints, parameter groups, and VPC security groups.

Console

To start an Aurora cluster

1. Sign in to the AWS Management Console and open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. In the navigation pane, choose **Databases**, and then choose a cluster. You can perform the start operation from this page, or navigate to the details page for the DB cluster that you want to start.
3. For **Actions**, choose **Start**.

AWS CLI

To start a DB cluster by using the AWS CLI, call the `start-db-cluster` command with the following parameters:

- `--db-cluster-identifier` – the name of the Aurora cluster. This name is either a specific cluster identifier you chose when creating the cluster, or the DB instance identifier you chose with `-cluster` appended to the end.

Example

```
aws rds start-db-cluster --db-cluster-identifier mydbcluster
```

RDS API

To start an Aurora DB cluster by using the Amazon RDS API, call the `StartDBCluster` operation with the following parameter:

- **DBCluster** – the name of the Aurora cluster. This name is either a specific cluster identifier you chose when creating the cluster, or the DB instance identifier you chose with `-cluster` appended to the end.

Modifying an Amazon Aurora DB cluster

You can change the settings of a DB cluster to accomplish tasks such as changing its backup retention period or its database port. You can also modify DB instances in a DB cluster to accomplish tasks such as changing its DB instance class or enabling Performance Insights for it. This topic guides you through modifying an Aurora DB cluster and its DB instances, and describes the settings for each.

We recommend that you test any changes on a test DB cluster or DB instance before modifying a production DB cluster or DB instance, so that you fully understand the impact of each change. This is especially important when upgrading database versions.

Modifying the DB cluster by using the console, CLI, and API

You can modify a DB cluster using the AWS Management Console, the AWS CLI, or the RDS API.

Note

For Aurora, when you modify a DB cluster, only changes to the **DB cluster identifier**, **IAM DB authentication**, and **New master password** settings are affected by the **Apply immediately** setting. All other modifications are applied immediately, regardless of the value of the **Apply immediately** setting.

Console

To modify a DB cluster

1. Sign in to the AWS Management Console and open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. In the navigation pane, choose **Databases**, and then select the DB cluster that you want to modify.
3. Choose **Modify**. The **Modify DB cluster** page appears.
4. Change any of the settings that you want. For information about each setting, see [Settings for Amazon Aurora \(p. 375\)](#).

Note

In the AWS Management Console, some instance level changes only apply to the current DB instance, while others apply to the entire DB cluster. For information about whether a setting applies to the DB instance or the DB cluster, see the scope for the setting in [Settings for Amazon Aurora \(p. 375\)](#). To change a setting that modifies the entire DB cluster at the instance level in the AWS Management Console, follow the instructions in [Modify a DB instance in a DB cluster \(p. 373\)](#).

5. When all the changes are as you want them, choose **Continue** and check the summary of modifications.
6. To apply the changes immediately, select **Apply immediately**.
7. On the confirmation page, review your changes. If they are correct, choose **Modify cluster** to save your changes.

Alternatively, choose **Back** to edit your changes, or choose **Cancel** to cancel your changes.

AWS CLI

To modify a DB cluster using the AWS CLI, call the `modify-db-cluster` command. Specify the DB cluster identifier, and the values for the settings that you want to modify. For information about each setting, see [Settings for Amazon Aurora \(p. 375\)](#).

Note

Some settings only apply to DB instances. To change those settings, follow the instructions in [Modify a DB instance in a DB cluster \(p. 373\)](#).

Example

The following command modifies `mydbcluster` by setting the backup retention period to 1 week (7 days).

For Linux, macOS, or Unix:

```
aws rds modify-db-cluster \
--db-cluster-identifier mydbcluster \
--backup-retention-period 7
```

For Windows:

```
aws rds modify-db-cluster ^
--db-cluster-identifier mydbcluster ^
--backup-retention-period 7
```

RDS API

To modify a DB cluster using the Amazon RDS API, call the [ModifyDBCluster](#) operation. Specify the DB cluster identifier, and the values for the settings that you want to modify. For information about each parameter, see [Settings for Amazon Aurora \(p. 375\)](#).

Note

Some settings only apply to DB instances. To change those settings, follow the instructions in [Modify a DB instance in a DB cluster \(p. 373\)](#).

Modify a DB instance in a DB cluster

You can modify a DB instance in a DB cluster using the AWS Management Console, the AWS CLI, or the RDS API.

When you modify a DB instance, you can apply the changes immediately. To apply changes immediately, you select the **Apply Immediately** option in the AWS Management Console, you use the `--apply-immediately` parameter when calling the AWS CLI, or you set the `ApplyImmediately` parameter to `true` when using the Amazon RDS API.

If you don't choose to apply changes immediately, the changes are deferred until the next maintenance window. During the next maintenance window, any of these deferred changes are applied. If you choose to apply changes immediately, your new changes and any previously deferred changes are applied.

Important

If any of the deferred modifications require downtime, choosing **Apply immediately** can cause unexpected downtime for the DB instance. There is no downtime for the other DB instances in the DB cluster.

Modifications that you defer aren't listed in the output of the `describe-pending-maintenance-actions` CLI command. Maintenance actions only include system upgrades that you schedule for the next maintenance window.

Console

To modify a DB instance in a DB cluster

1. Sign in to the AWS Management Console and open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.

2. In the navigation pane, choose **Databases**, and then select the DB instance that you want to modify.
3. For **Actions**, choose **Modify**. The **Modify DB Instance** page appears.
4. Change any of the settings that you want. For information about each setting, see [Settings for Amazon Aurora \(p. 375\)](#).

Note

Some settings apply to the entire DB cluster and must be changed at the cluster level. To change those settings, follow the instructions in [Modifying the DB cluster by using the console, CLI, and API \(p. 372\)](#).

In the AWS Management Console, some instance level changes only apply to the current DB instance, while others apply to the entire DB cluster. For information about whether a setting applies to the DB instance or the DB cluster, see the scope for the setting in [Settings for Amazon Aurora \(p. 375\)](#).

5. When all the changes are as you want them, choose **Continue** and check the summary of modifications.
6. To apply the changes immediately, select **Apply immediately**.
7. On the confirmation page, review your changes. If they are correct, choose **Modify DB Instance** to save your changes.

Alternatively, choose **Back** to edit your changes, or choose **Cancel** to cancel your changes.

AWS CLI

To modify a DB instance in a DB cluster by using the AWS CLI, call the `modify-db-instance` command. Specify the DB instance identifier, and the values for the settings that you want to modify. For information about each parameter, see [Settings for Amazon Aurora \(p. 375\)](#).

Note

Some settings apply to the entire DB cluster. To change those settings, follow the instructions in [Modifying the DB cluster by using the console, CLI, and API \(p. 372\)](#).

Example

The following code modifies `mydbinstance` by setting the DB instance class to `db.r4.xlarge`. The changes are applied during the next maintenance window by using `--no-apply-immediately`. Use `--apply-immediately` to apply the changes immediately.

For Linux, macOS, or Unix:

```
aws rds modify-db-instance \
--db-instance-identifier mydbinstance \
--db-instance-class db.r4.xlarge \
--no-apply-immediately
```

For Windows:

```
aws rds modify-db-instance ^
--db-instance-identifier mydbinstance ^
--db-instance-class db.r4.xlarge ^
--no-apply-immediately
```

RDS API

To modify a MySQL instance by using the Amazon RDS API, call the `ModifyDBInstance` operation. Specify the DB instance identifier, and the values for the settings that you want to modify. For information about each parameter, see [Settings for Amazon Aurora \(p. 375\)](#).

Note

Some settings apply to the entire DB cluster. To change those settings, follow the instructions in [Modifying the DB cluster by using the console, CLI, and API \(p. 372\)](#).

Settings for Amazon Aurora

The following table contains details about which settings you can modify, the methods for modifying the setting, and the scope of the setting. The scope determines whether the setting applies to the entire DB cluster or if it can be set only for specific DB instances.

Note

Additional settings are available if you are modifying an Aurora Serverless DB cluster. For information about these settings, see [Modifying an Aurora Serverless v1 DB cluster \(p. 170\)](#).

Also, some settings aren't available for Aurora Serverless because of Aurora Serverless limitations. For more information, see [Limitations of Aurora Serverless v1 \(p. 148\)](#).

Setting and description	Method	Scope	Downtime notes
<p>Auto minor version upgrade</p> <p>Whether you want the DB instance to receive preferred minor engine version upgrades automatically when they become available. Upgrades are installed only during your scheduled maintenance window.</p> <p>For more information about engine updates, see Amazon Aurora PostgreSQL updates (p. 1597) and Database engine updates for Amazon Aurora MySQL (p. 1082). For more information about the Auto minor version upgrade setting for Aurora MySQL, see Enabling automatic upgrades between minor Aurora MySQL versions (p. 1089).</p>	<p>Note</p> <p>This setting is enabled by default. For each new cluster, choose the appropriate value for this setting based on its importance, expected lifetime, and the amount of verification testing that you do after each upgrade.</p> <p>When you change this setting, perform this modification for every DB instance in your Aurora cluster. If any DB instance in your cluster has this setting turned off, the cluster isn't automatically upgraded.</p> <p>Using the AWS Management Console, Modify a DB instance in a DB cluster (p. 373).</p> <p>Using the AWS CLI, run <code>modify-db-instance</code></p>	The entire DB cluster	An outage doesn't occur during this change. Outages do occur during future maintenance windows when Aurora applies automatic upgrades.

Setting and description	Method	Scope	Downtime notes
	<p>and set the <code>--auto-minor-version-upgrade</code> <code>--no-auto-minor-version-upgrade</code> option.</p> <p>Using the RDS API, call ModifyDBInstance and set the <code>AutoMinorVersionUpgrade</code> parameter.</p>		
Backup retention period The number of days that automatic backups are retained. The minimum value is 1. For more information, see Backups (p. 491) .	<p>Using the AWS Management Console, Modifying the DB cluster by using the console, CLI, and API (p. 372).</p> <p>Using the AWS CLI, run <code>modify-db-cluster</code> and set the <code>--backup-retention-period</code> option.</p> <p>Using the RDS API, call ModifyDBCluster and set the <code>BackupRetentionPeriod</code> parameter.</p>	The entire DB cluster	An outage doesn't occur during this change.

Setting and description	Method	Scope	Downtime notes
<p>Backup window (Start time)</p> <p>The time range during which automated backups of your database occurs. The backup window is a start time in Universal Coordinated Time (UTC), and a duration in hours.</p> <p>Aurora backups are continuous and incremental, but the backup window is used to create a daily system backup that is preserved within the backup retention period. You can copy it to preserve it outside of the retention period.</p> <p>The maintenance window and the backup window for the DB cluster can't overlap.</p> <p>For more information, see Backup window (p. 491).</p>	<p>Using the AWS Management Console, Modifying the DB cluster by using the console, CLI, and API (p. 372).</p> <p>Using the AWS CLI, run <code>modify-db-cluster</code> and set the <code>--preferred-backup-window</code> option.</p> <p>Using the RDS API, call <code>ModifyDBCluster</code> and set the <code>PreferredBackupWindow</code> parameter.</p>	The entire DB cluster.	An outage doesn't occur during this change.
<p>Certificate Authority</p> <p>The certificate that you want to use.</p>	<p>Using the AWS Management Console, Modify a DB instance in a DB cluster (p. 373).</p> <p>Using the AWS CLI, run <code>modify-db-instance</code> and set the <code>--ca-certificate-identifier</code> option.</p> <p>Using the RDS API, call <code>ModifyDBInstance</code> and set the <code>CACertificateIdentifier</code> parameter.</p>	Only the specified DB instance	An outage occurs during this change. The DB instance is rebooted.

Setting and description	Method	Scope	Downtime notes
Copy tags to snapshots Select to specify that tags defined for this DB cluster are copied to DB snapshots created from this DB cluster. For more information, see Tagging Amazon RDS resources (p. 474) .	Using the AWS Management Console, Modifying the DB cluster by using the console, CLI, and API (p. 372) . Using the AWS CLI, run <code>modify-db-cluster</code> and set the <code>--copy-tags-to-snapshot</code> or <code>--no-copy-tags-to-snapshot</code> option. Using the RDS API, call <code>ModifyDBCluster</code> and set the <code>CopyTagsToSnapshot</code> parameter.	The entire DB cluster	An outage doesn't occur during this change.
Data API You can access Aurora Serverless with web services-based applications, including AWS Lambda and AWS AppSync. This setting only applies to an Aurora Serverless DB cluster. For more information, see Using the Data API for Aurora Serverless (p. 178) .	Using the AWS Management Console, Modifying the DB cluster by using the console, CLI, and API (p. 372) . Using the AWS CLI, run <code>modify-db-cluster</code> and set the <code>--enable-http-endpoint</code> option. Using the RDS API, call <code>ModifyDBCluster</code> and set the <code>EnableHttpEndpoint</code> parameter.	The entire DB cluster	An outage doesn't occur during this change.

Setting and description	Method	Scope	Downtime notes
<p>Database authentication The database authentication you want to use.</p> <p>For MySQL:</p> <ul style="list-style-type: none"> Choose Password authentication to authenticate database users with database passwords only. Choose Password and IAM database authentication to authenticate database users with database passwords and user credentials through IAM users and roles. For more information, see IAM database authentication (p. 1743). <p>For PostgreSQL:</p> <ul style="list-style-type: none"> Choose IAM database authentication to authenticate database users with database passwords and user credentials through IAM users and roles. For more information, see IAM database authentication (p. 1743). Choose Kerberos authentication to authenticate database passwords and user credentials using Kerberos authentication. For more information, see Using Kerberos authentication with Aurora PostgreSQL (p. 1534). 	<p>Using the AWS Management Console, Modifying the DB cluster by using the console, CLI, and API (p. 372).</p> <p>Using the AWS CLI, run <code>modify-db-cluster</code> and set the following options:</p> <ul style="list-style-type: none"> For IAM authentication, set the <code>--enable-iam-database-authentication --no-enable-iam-database-authentication</code> option. For Kerberos authentication, set the <code>--domain</code> and <code>--domain-iam-role-name</code> options. <p>Using the RDS API, call <code>ModifyDBCluster</code> and set the following parameters:</p> <ul style="list-style-type: none"> For IAM authentication, set the <code>EnableIAMDatabaseAuthentication</code> parameter. For Kerberos authentication, set the <code>Domain</code> and <code>DomainIAMRoleName</code> parameters. 	The entire DB cluster	An outage doesn't occur during this change.

Setting and description	Method	Scope	Downtime notes
Database port The port that you want to use to access the DB cluster.	Using the AWS Management Console, Modifying the DB cluster by using the console, CLI, and API (p. 372) . Using the AWS CLI, run <code>modify-db-cluster</code> and set the <code>--port</code> option. Using the RDS API, call <code>ModifyDBCluster</code> and set the <code>Port</code> parameter.	The entire DB cluster	An outage occurs during this change. All of the DB instances in the DB cluster are rebooted immediately.
DB cluster identifier The DB cluster identifier. This value is stored as a lowercase string. When you change the DB cluster identifier, the DB cluster endpoint changes, and the endpoints of the DB instances in the DB cluster change.	Using the AWS Management Console, Modifying the DB cluster by using the console, CLI, and API (p. 372) . Using the AWS CLI, run <code>modify-db-cluster</code> and set the <code>--new-db-cluster-identifier</code> option. Using the RDS API, call <code>ModifyDBCluster</code> and set the <code>NewDBClusterIdentifier</code> parameter.	The entire DB cluster	An outage doesn't occur during this change.
DB cluster parameter group The DB cluster parameter group that you want associated with the DB cluster. For more information, see Working with DB parameter groups and DB cluster parameter groups (p. 339) .	Using the AWS Management Console, Modifying the DB cluster by using the console, CLI, and API (p. 372) . Using the AWS CLI, run <code>modify-db-cluster</code> and set the <code>--db-cluster-parameter-group-name</code> option. Using the RDS API, call <code>ModifyDBCluster</code> and set the <code>DBClusterParameterGroupName</code> parameter.	The entire DB cluster	An outage doesn't occur during this change. When you change the parameter group, changes to some parameters are applied to the DB instances in the DB cluster immediately without a reboot. Changes to other parameters are applied only after the DB instances in the DB cluster are rebooted.

Setting and description	Method	Scope	Downtime notes
DB instance class The DB instance class that you want to use. For more information, see Aurora DB instance classes (p. 54) .	Using the AWS Management Console, Modify a DB instance in a DB cluster (p. 373) . Using the AWS CLI, run <code>modify-db-instance</code> and set the <code>--db-instance-class</code> option. Using the RDS API, call ModifyDBInstance and set the <code>DBInstanceClass</code> parameter.	Only the specified DB instance	An outage occurs during this change.
DB instance identifier The DB instance identifier. This value is stored as a lowercase string.	Using the AWS Management Console, Modify a DB instance in a DB cluster (p. 373) . Using the AWS CLI, run <code>modify-db-instance</code> and set the <code>--new-db-instance-identifier</code> option. Using the RDS API, call ModifyDBInstance and set the <code>NewDBInstanceIdentifier</code> parameter.	Only the specified DB instance	An outage occurs during this change. The DB instance is rebooted.

Setting and description	Method	Scope	Downtime notes
DB parameter group The DB parameter group that you want associated with the DB instance. For more information, see Working with DB parameter groups and DB cluster parameter groups (p. 339) .	Using the AWS Management Console, Modify a DB instance in a DB cluster (p. 373) . Using the AWS CLI, run <code>modify-db-instance</code> and set the <code>--db-parameter-group-name</code> option. Using the RDS API, call ModifyDBInstance and set the <code>DBParameterGroupName</code> parameter.	Only the specified DB instance	An outage doesn't occur during this change. When you associate a new DB parameter group with a DB instance, the modified static and dynamic parameters are applied only after the DB instance is rebooted. However, if you modify dynamic parameters in the newly associated DB parameter group, these changes are applied immediately without a reboot. For more information, see Working with DB parameter groups and DB cluster parameter groups (p. 339) and Rebooting an Amazon Aurora DB cluster or Amazon Aurora DB instance (p. 451) .
Deletion protection Enable deletion protection to prevent your DB cluster from being deleted. For more information, see Deletion protection for Aurora clusters (p. 471) .	Using the AWS Management Console, Modifying the DB cluster by using the console, CLI, and API (p. 372) . Using the AWS CLI, run <code>modify-db-cluster</code> and set the <code>--deletion-protection --no-deletion-protection</code> option. Using the RDS API, call ModifyDBCluster and set the <code>DeletionProtection</code> parameter.	The entire DB cluster	An outage doesn't occur during this change.

Setting and description	Method	Scope	Downtime notes
Engine version The version of the DB engine that you want to use. Before you upgrade your production DB cluster, we recommend that you test the upgrade process on a test DB cluster to verify its duration and to validate your applications.	Using the AWS Management Console, Modifying the DB cluster by using the console, CLI, and API (p. 372) . Using the AWS CLI, run <code>modify-db-cluster</code> and set the <code>--engine-version</code> option. Using the RDS API, call <code>ModifyDBCluster</code> and set the <code>EngineVersion</code> parameter.	The entire DB cluster	An outage occurs during this change.
Enhanced monitoring Enable enhanced monitoring to enable gathering metrics in real time for the operating system that your DB instance runs on. For more information, see Monitoring OS metrics with Enhanced Monitoring (p. 626) .	Using the AWS Management Console, Modify a DB instance in a DB cluster (p. 373) . Using the AWS CLI, run <code>modify-db-instance</code> and set the <code>--monitoring-role-arn</code> and <code>--monitoring-interval</code> options. Using the RDS API, call <code>ModifyDBInstance</code> and set the <code>MonitoringRoleArn</code> and <code>MonitoringInterval</code> parameters.	Only the specified DB instance	An outage doesn't occur during this change.

Setting and description	Method	Scope	Downtime notes
<p>Log exports</p> <p>Select the log types to publish to Amazon CloudWatch Logs.</p> <p>For more information, see Aurora MySQL database log files (p. 700).</p>	<p>Using the AWS Management Console, Modifying the DB cluster by using the console, CLI, and API (p. 372).</p> <p>Using the AWS CLI, run <code>modify-db-cluster</code> and set the <code>--cloudwatch-logs-export-configuration</code> option.</p> <p>Using the RDS API, call <code>ModifyDBCluster</code> and set the <code>CloudwatchLogsExportConfiguration</code> parameter.</p>	The entire DB cluster	An outage doesn't occur during this change.

Setting and description	Method	Scope	Downtime notes
<p>Maintenance window</p> <p>The time range during which system maintenance occurs. System maintenance includes upgrades, if applicable. The maintenance window is a start time in Universal Coordinated Time (UTC), and a duration in hours.</p> <p>If you set the window to the current time, there must be at least 30 minutes between the current time and end of the window to ensure any pending changes are applied.</p> <p>You can set the maintenance window independently for the DB cluster and for each DB instance in the DB cluster. When the scope of a modification is the entire DB cluster, the modification is performed during the DB cluster maintenance window. When the scope of a modification is the a DB instance, the modification is performed during maintenance window of that DB instance.</p> <p>The maintenance window and the backup window for the DB cluster can't overlap.</p> <p>For more information, see The Amazon RDS maintenance window (p. 446).</p>	<p>To change the maintenance window for the DB cluster using the AWS Management Console, Modifying the DB cluster by using the console, CLI, and API (p. 372).</p> <p>To change the maintenance window for a DB instance using the AWS Management Console, Modify a DB instance in a DB cluster (p. 373).</p> <p>To change the maintenance window for the DB cluster using the AWS CLI, run <code>modify-db-cluster</code> and set the <code>--preferred-maintenance-window</code> option.</p> <p>To change the maintenance window for a DB instance using the AWS CLI, run <code>modify-db-instance</code> and set the <code>--preferred-maintenance-window</code> option.</p> <p>To change the maintenance window for the DB cluster using the RDS API, call ModifyDBCluster and set the <code>PreferredMaintenanceWindow</code> parameter.</p> <p>To change the maintenance window for a DB instance using the RDS API, call ModifyDBInstance and set the <code>PreferredMaintenanceWindow</code> parameter.</p>	<p>The entire DB cluster or a single DB instance</p>	<p>If there are one or more pending actions that cause an outage, and the maintenance window is changed to include the current time, then those pending actions are applied immediately, and an outage occurs.</p>

Setting and description	Method	Scope	Downtime notes
New master password The password for your master user. The password must contain from 8 to 41 alphanumeric characters.	Using the AWS Management Console, Modifying the DB cluster by using the console, CLI, and API (p. 372) . Using the AWS CLI, run <code>modify-db-cluster</code> and set the <code>--master-user-password</code> option. Using the RDS API, call <code>ModifyDBCluster</code> and set the <code>MasterUserPassword</code> parameter.	The entire DB cluster	An outage doesn't occur during this change.
Performance Insights Whether to enable Performance Insights, a tool that monitors your DB instance load so that you can analyze and troubleshoot your database performance. For more information, see Monitoring DB load with Performance Insights on Amazon Aurora (p. 573) .	Using the AWS Management Console, Modify a DB instance in a DB cluster (p. 373) . Using the AWS CLI, run <code>modify-db-instance</code> and set the <code>--enable-performance-insights --no-enable-performance-insights</code> option. Using the RDS API, call <code>ModifyDBInstance</code> and set the <code>EnablePerformanceInsights</code> parameter.	Only the specified DB instance	An outage doesn't occur during this change.

Setting and description	Method	Scope	Downtime notes
Performance Insights AWS KMS key The AWS KMS key identifier for encryption of Performance Insights data. The KMS key identifier is the Amazon Resource Name (ARN), key identifier, or key alias for the KMS key. For more information, see Enabling and disabling Performance Insights (p. 577) .	Using the AWS Management Console, Modify a DB instance in a DB cluster (p. 373) . Using the AWS CLI, run <code>modify-db-instance</code> and set the <code>--performance-insights-kms-key-id</code> option. Using the RDS API, call <code>ModifyDBInstance</code> and set the <code>PerformanceInsightsKMSKeyId</code> parameter.	Only the specified DB instance	An outage doesn't occur during this change.
Performance Insights retention period The amount of time, in days, to retain Performance Insights data. Valid values are 7 or 731 (2 years). For more information, see Enabling and disabling Performance Insights (p. 577) .	Using the AWS Management Console, Modify a DB instance in a DB cluster (p. 373) . Using the AWS CLI, run <code>modify-db-instance</code> and set the <code>--performance-insights-retention-period</code> option. Using the RDS API, call <code>ModifyDBInstance</code> and set the <code>PerformanceInsightsRetentionPeriod</code> parameter.	Only the specified DB instance	An outage doesn't occur during this change.
Promotion tier A value that specifies the order in which an Aurora Replica is promoted to the primary instance in a cluster that uses single-master replication, after a failure of the existing primary instance. For more information, see Fault tolerance for an Aurora DB cluster (p. 69) .	Using the AWS Management Console, Modify a DB instance in a DB cluster (p. 373) . Using the AWS CLI, run <code>modify-db-instance</code> and set the <code>--promotion-tier</code> option. Using the RDS API, call <code>ModifyDBInstance</code> and set the <code>PromotionTier</code> parameter.	Only the specified DB instance	An outage doesn't occur during this change.

Setting and description	Method	Scope	Downtime notes
<p>Public access</p> <p>Publicly accessible to give the DB instance a public IP address, meaning that it's accessible outside the VPC. To be publicly accessible, the DB instance also has to be in a public subnet in the VPC.</p> <p>Not publicly accessible to make the DB instance accessible only from inside the VPC.</p> <p>For more information, see Hiding a DB instance in a VPC from the internet (p. 1789).</p> <p>To connect to a DB instance from outside of its Amazon VPC, the DB instance must be publicly accessible, access must be granted using the inbound rules of the DB instance's security group, and other requirements must be met. For more information, see Can't connect to Amazon RDS DB instance (p. 1814).</p> <p>If your DB instance is isn't publicly accessible, you can also use an AWS Site-to-Site VPN connection or an AWS Direct Connect connection to access it from a private network. For more information, see Internetwork traffic privacy (p. 1723).</p>	<p>Using the AWS Management Console, Modify a DB instance in a DB cluster (p. 373).</p> <p>Using the AWS CLI, run <code>modify-db-instance</code> and set the <code>--publicly-accessible --no-publicly-accessible</code> option.</p> <p>Using the RDS API, call ModifyDBInstance and set the <code>PubliclyAccessible</code> parameter.</p>	Only the specified DB instance	An outage doesn't occur during this change.

Setting and description	Method	Scope	Downtime notes
Scaling configuration The scaling properties of the DB cluster. You can only modify scaling properties for DB clusters in serverless DB engine mode. This setting is available only for Aurora MySQL. For information about Aurora Serverless, see Using Amazon Aurora Serverless v1 (p. 147) .	Using the AWS Management Console, Modifying the DB cluster by using the console, CLI, and API (p. 372) . Using the AWS CLI, run <code>modify-db-cluster</code> and set the <code>--scaling-configuration</code> option. Using the RDS API, call <code>ModifyDBCluster</code> and set the <code>ScalingConfiguration</code> parameter.	The entire DB cluster	An outage doesn't occur during this change.
Security group The security group you want associated with the DB cluster. For more information, see Controlling access with security groups (p. 1780) .	Using the AWS Management Console, Modifying the DB cluster by using the console, CLI, and API (p. 372) . Using the AWS CLI, run <code>modify-db-cluster</code> and set the <code>--vpc-security-group-ids</code> option. Using the RDS API, call <code>ModifyDBCluster</code> and set the <code>VpcSecurityGroupIds</code> parameter.	The entire DB cluster	An outage doesn't occur during this change.

Setting and description	Method	Scope	Downtime notes
Target Backtrack window The amount of time you want to be able to backtrack your DB cluster, in seconds. This setting is available only for Aurora MySQL and only if the DB cluster was created with Backtrack enabled.	Using the AWS Management Console, Modifying the DB cluster by using the console, CLI, and API (p. 372) . Using the AWS CLI, run <code>modify-db-cluster</code> and set the <code>--backtrack-window</code> option. Using the RDS API, call <code>ModifyDBCluster</code> and set the <code>BacktrackWindow</code> parameter.	The entire DB cluster	An outage doesn't occur during this change.

Settings that do not apply to Amazon Aurora

The following settings in the AWS CLI command `modify-db-instance` and the RDS API operation [`ModifyDBInstance`](#) do not apply to Amazon Aurora.

Note

The AWS Management Console doesn't allow you to modify these settings for Aurora.

AWS CLI setting	RDS API setting
<code>--allocated-storage</code>	<code>AllocatedStorage</code>
<code>--allow-major-version-upgrade --no-allow-major-version-upgrade</code>	<code>AllowMajorVersionUpgrade</code>
<code>--copy-tags-to-snapshot --no-copy-tags-to-snapshot</code>	<code>CopyTagsToSnapshot</code>
<code>--domain</code>	<code>Domain</code>
<code>--db-security-groups</code>	<code>DBSecurityGroups</code>
<code>--db-subnet-group-name</code>	<code>DBSubnetGroupName</code>
<code>--domain-iam-role-name</code>	<code>DomainIAMRoleName</code>
<code>--multi-az --no-multi-az</code>	<code>MultiAZ</code>
<code>--iops</code>	<code>Iops</code>
<code>--license-model</code>	<code>LicenseModel</code>
<code>--option-group-name</code>	<code>OptionGroupName</code>
<code>--processor-features</code>	<code>ProcessorFeatures</code>

AWS CLI setting	RDS API setting
--storage-type	StorageType
--tde-credential-arn	TdeCredentialArn
--tde-credential-password	TdeCredentialPassword
--use-default-processor-features --no-use-default-processor-features	UseDefaultProcessorFeatures

Adding Aurora Replicas to a DB cluster

An Aurora DB cluster with single-master replication has one primary DB instance and up to 15 Aurora Replicas. The primary DB instance supports read and write operations, and performs all data modifications to the cluster volume. Aurora Replicas connect to the same storage volume as the primary DB instance, but support read operations only. You use Aurora Replicas to offload read workloads from the primary DB instance. For more information, see [Aurora Replicas \(p. 70\)](#).

Amazon Aurora Replicas have the following limitations:

- You can't create an Aurora Replica for an Aurora Serverless v1 DB cluster. Aurora Serverless v1 has a single DB instance that scales up and down automatically to support all database read and write operations.
- You can't create Aurora Replicas for an Aurora multi-master cluster. By design, an Aurora multi-master cluster has read-write DB instances only.

We recommend that you distribute the primary instance and Aurora Replicas of your Aurora DB cluster over multiple Availability Zones to improve the availability of your DB cluster. For more information, see [Region availability \(p. 12\)](#).

To remove an Aurora Replica from an Aurora DB cluster, delete the Aurora Replica by following the instructions in [Deleting a DB instance from an Aurora DB cluster \(p. 472\)](#).

Note

Amazon Aurora also supports replication with an external database, such as an RDS DB instance. The RDS DB instance must be in the same AWS Region as Amazon Aurora. For more information, see [Replication with Amazon Aurora \(p. 70\)](#).

You can add Aurora Replicas to a DB cluster using the AWS Management Console, the AWS CLI, or the RDS API.

Console

To add an Aurora replica to a DB cluster

1. Sign in to the AWS Management Console and open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. In the navigation pane, choose **Databases**, and then select the DB cluster where you want to add the new DB instance.
3. Make sure that both the cluster and the primary instance are in the **Available** state. If the DB cluster or the primary instance are in a transitional state such as **Creating**, you can't add a replica.

If the cluster doesn't have a primary instance, create one using the `create-db-instance` AWS CLI command. This situation can arise if you used the CLI to restore a DB cluster snapshot and then view the cluster in the AWS Management Console.

4. For **Actions**, choose **Add reader**.

The **Add reader** page appears.

5. On the **Add reader** page, specify options for your Aurora Replica. The following table shows settings for an Aurora Replica.

For this option	Do this
Availability zone	Determine if you want to specify a particular Availability Zone. The list includes only those Availability Zones

For this option	Do this
	that are mapped by the DB subnet group you specified earlier. For more information about Availability Zones, see Regions and Availability Zones (p. 11) .
Publicly accessible	Select Yes to give the Aurora Replica a public IP address; otherwise, select No. For more information about hiding Aurora Replicas from public access, see Hiding a DB instance in a VPC from the internet (p. 1789) .
Encryption	Select Enable encryption to enable encryption at rest for this Aurora Replica. For more information, see Encrypting Amazon Aurora resources (p. 1709) .
DB instance class	Select a DB instance class that defines the processing and memory requirements for the Aurora Replica. For more information about DB instance class options, see Aurora DB instance classes (p. 54) .
Aurora replica source	Select the identifier of the primary instance to create an Aurora Replica for.
DB instance identifier	Enter a name for the instance that is unique for your account in the AWS Region you selected. You might choose to add some intelligence to the name such as including the AWS Region and DB engine you selected, for example aurora-read-instance1 .
Priority	Choose a failover priority for the instance. If you don't select a value, the default is tier-1 . This priority determines the order in which Aurora Replicas are promoted when recovering from a primary instance failure. For more information, see Fault tolerance for an Aurora DB cluster (p. 69) .
Database port	The port for an Aurora Replica is the same as the port for the DB cluster.
DB parameter group	Select a parameter group. Aurora has a default parameter group you can use, or you can create your own parameter group. For more information about parameter groups, see Working with DB parameter groups and DB cluster parameter groups (p. 339) .
Enhanced monitoring	Choose Enable enhanced monitoring to enable gathering metrics in real time for the operating system that your DB cluster runs on. For more information, see Monitoring OS metrics with Enhanced Monitoring (p. 626) .
Monitoring Role	Only available if Enhanced Monitoring is set to Enable enhanced monitoring . Choose the IAM role that you created to permit Amazon RDS to communicate with Amazon CloudWatch Logs for you, or choose Default to have RDS create a role for you named <code>rds-monitoring-role</code> . For more information, see Monitoring OS metrics with Enhanced Monitoring (p. 626) .

For this option	Do this
Granularity	Only available if Enhanced Monitoring is set to Enable enhanced monitoring . Set the interval, in seconds, between when metrics are collected for your DB cluster.
Auto minor version upgrade	<p>Select Enable auto minor version upgrade if you want to enable your Aurora DB cluster to receive minor DB Engine version upgrades automatically when they become available.</p> <p>The Auto minor version upgrade setting applies to both Aurora PostgreSQL and Aurora MySQL DB clusters. For Aurora MySQL 1.x and 2.x clusters, this setting upgrades the clusters to a maximum version of 1.22.2 and 2.07.2.</p> <p>For more information about engine updates for Aurora PostgreSQL, see Amazon Aurora PostgreSQL updates (p. 1597).</p> <p>For more information about engine updates for Aurora MySQL, see Database engine updates for Amazon Aurora MySQL (p. 1082).</p>

6. Choose **Add reader** to create the Aurora Replica.

AWS CLI

To create an Aurora Replica in your DB cluster, run the [create-db-instance](#) AWS CLI command. Include the name of the DB cluster as the `--db-cluster-identifier` option. You can optionally specify an Availability Zone for the Aurora Replica using the `--availability-zone` parameter, as shown in the following examples.

For example, the following command creates a new MySQL 5.7-compatible Aurora Replica named `sample-instance-us-west-2a`.

For Linux, macOS, or Unix:

```
aws rds create-db-instance --db-instance-identifier sample-instance-us-west-2a \
    --db-cluster-identifier sample-cluster --engine aurora-mysql --db-instance-class
    db.r5.large \
    --availability-zone us-west-2a
```

For Windows:

```
aws rds create-db-instance --db-instance-identifier sample-instance-us-west-2a ^
    --db-cluster-identifier sample-cluster --engine aurora-mysql --db-instance-class
    db.r5.large ^
    --availability-zone us-west-2a
```

The following command creates a new MySQL 5.6-compatible Aurora Replica named `sample-instance-us-west-2a`.

For Linux, macOS, or Unix:

```
aws rds create-db-instance --db-instance-identifier sample-instance-us-west-2a \
    --db-cluster-identifier sample-cluster --engine aurora --db-instance-class db.r5.large \
    \
```

```
--availability-zone us-west-2a
```

For Windows:

```
aws rds create-db-instance --db-instance-identifier sample-instance-us-west-2a ^
--db-cluster-identifier sample-cluster --engine aurora --db-instance-class db.r5.large
^
--availability-zone us-west-2a
```

The following command creates a new PostgreSQL-compatible Aurora Replica named `sample-instance-us-west-2a`.

For Linux, macOS, or Unix:

```
aws rds create-db-instance --db-instance-identifier sample-instance-us-west-2a \
--db-cluster-identifier sample-cluster --engine aurora-postgresql --db-instance-class
db.r5.large \
--availability-zone us-west-2a
```

For Windows:

```
aws rds create-db-instance --db-instance-identifier sample-instance-us-west-2a ^
--db-cluster-identifier sample-cluster --engine aurora-postgresql --db-instance-class
db.r5.large ^
--availability-zone us-west-2a
```

RDS API

To create an Aurora Replica in your DB cluster, call the [CreateDBInstance](#) operation. Include the name of the DB cluster as the `DBClusterIdentifier` parameter. You can optionally specify an Availability Zone for the Aurora Replica using the `AvailabilityZone` parameter.

Managing performance and scaling for Aurora DB clusters

You can use the following options to manage performance and scaling for Aurora DB clusters and DB instances:

Topics

- [Storage scaling \(p. 396\)](#)
- [Instance scaling \(p. 400\)](#)
- [Read scaling \(p. 400\)](#)
- [Managing connections \(p. 400\)](#)
- [Managing query execution plans \(p. 401\)](#)

Storage scaling

Aurora storage automatically scales with the data in your cluster volume. As your data grows, your cluster volume storage expands up to a maximum of 128 tebibytes (TiB) or 64 TiB. The maximum size depends on the DB engine version. To learn what kinds of data are included in the cluster volume, see [Amazon Aurora storage and reliability \(p. 64\)](#). For details about the maximum size for a specific version, see [Amazon Aurora size limits \(p. 1813\)](#).

The size of your cluster volume is evaluated on an hourly basis to determine your storage costs. For pricing information, see the [Aurora pricing page](#).

Even though an Aurora cluster volume can scale up in size to many tebibytes, you are only charged for the space that you use in the volume. The mechanism for determining billed storage space depends on the version of your Aurora cluster.

- When Aurora data is removed from the cluster volume, the overall billed space decreases by a comparable amount. This dynamic resizing behavior happens when underlying database files are deleted or reorganized to require less space. Thus, you can reduce storage charges by deleting tables, indexes, databases, and so on that you no longer need. Dynamic resizing applies to certain Aurora versions. The following are the Aurora versions where the cluster volume dynamically resizes as you delete data:
 - Aurora MySQL 3 (compatible with MySQL 8.0), all minor versions
 - Aurora MySQL 2.09 (compatible with MySQL 5.7) and higher
 - Aurora MySQL version 1.23 (compatible with MySQL 5.6) and higher
 - All Aurora PostgreSQL 13 versions
 - Aurora PostgreSQL version 12.4 and higher
 - Aurora PostgreSQL version 11.8 and higher
 - Aurora PostgreSQL version 10.13 and higher
- In Aurora versions lower than those in the preceding list, the cluster volume can reuse space that was freed up when you deleted data, but the volume itself never decreases in size.
- This feature is being deployed in phases to the AWS Regions where Aurora is available. Depending on the Region where your cluster is, this feature might not be available yet.

Dynamic resizing applies to operations that physically remove or resize data files within the cluster volume. Thus, it applies to SQL statements such as `DROP TABLE`, `DROP DATABASE`, `TRUNCATE TABLE`, and `ALTER TABLE ... DROP PARTITION`. It doesn't apply to deleting rows using the `DELETE` statement. If you delete a large number of rows from a table, you can run the Aurora MySQL `OPTIMIZE`

TABLE statement or use the Aurora PostgreSQL pg_repack extension afterward to reorganize the table and dynamically resize the cluster volume.

Note

For Aurora MySQL, the innodb_file_per_table affects how table storage is organized.

When tables are part of the system tablespace, dropping the table doesn't reduce the size of the system tablespace. Thus, make sure to use the setting innodb_file_per_table=1 for Aurora MySQL clusters to take full advantage of dynamic resizing.

These Aurora versions also have a higher storage limit for the cluster volume than lower versions do. Thus, you can consider upgrading to one of these versions if you are close to exceeding the original 64 TiB volume size.

You can check how much storage space a cluster is using by monitoring the VolumeBytesUsed metric in CloudWatch.

- In the AWS Management Console, you can see this figure in a chart by viewing the Monitoring tab on the details page for the cluster.
- With the AWS CLI, you can run a command similar to the following Linux example. Substitute your own values for the start and end times and the name of the cluster.

```
aws cloudwatch get-metric-statistics --metric-name "VolumeBytesUsed" \
--start-time "$(date -d '6 hours ago')" --end-time "$(date -d 'now')" --period 60 \
--namespace "AWS/RDS" \
--statistics Average Maximum Minimum \
--dimensions Name=DBClusterIdentifier,Value=my_cluster_identifier
```

That command produces output similar to the following.

```
{  
    "Label": "VolumeBytesUsed",  
    "Datapoints": [  
        {  
            "Timestamp": "2020-08-04T21:25:00+00:00",  
            "Average": 182871982080.0,  
            "Minimum": 182871982080.0,  
            "Maximum": 182871982080.0,  
            "Unit": "Bytes"  
        }  
    ]  
}
```

The following examples show how you might track storage usage for an Aurora cluster over time using AWS CLI commands on a Linux system. The --start-time and --end-time parameters define the overall time interval as one day. The --period parameter requests the measurements at one hour intervals. It doesn't make sense to choose a --period value that's small, because the metrics are collected at intervals, not continuously. Also, Aurora storage operations sometimes continue for some time in the background after the relevant SQL statement finishes.

The first example returns output in the default JSON format. The data points are returned in arbitrary order, not sorted by timestamp. You might import this JSON data into a charting tool to do sorting and visualization.

```
$ aws cloudwatch get-metric-statistics --metric-name "VolumeBytesUsed" \
--start-time "$(date -d '1 day ago')" --end-time "$(date -d 'now')" --period 3600
--namespace "AWS/RDS" --statistics Maximum --dimensions
Name=DBClusterIdentifier,Value=my_cluster_id
{  
    "Label": "VolumeBytesUsed",
```

```

    "Datapoints": [
      {
        "Timestamp": "2020-08-04T19:40:00+00:00",
        "Maximum": 182872522752.0,
        "Unit": "Bytes"
      },
      {
        "Timestamp": "2020-08-05T00:40:00+00:00",
        "Maximum": 198573719552.0,
        "Unit": "Bytes"
      },
      {
        "Timestamp": "2020-08-05T05:40:00+00:00",
        "Maximum": 206827454464.0,
        "Unit": "Bytes"
      },
      {
        "Timestamp": "2020-08-04T17:40:00+00:00",
        "Maximum": 182872522752.0,
        "Unit": "Bytes"
      },
      ...
    ... output omitted ...
  
```

This example returns the same data as the previous one. The `--output` parameter represents the data in compact plain text format. The `aws cloudwatch` command pipes its output to the `sort` command. The `-k` parameter of the `sort` command sorts the output by the third field, which is the timestamp in Universal Coordinated Time (UTC) format.

```

$ aws cloudwatch get-metric-statistics --metric-name "VolumeBytesUsed" \
--start-time "$(date -d '1 day ago')" --end-time "$(date -d 'now')" --period 3600 \
--namespace "AWS/RDS" --statistics Maximum --dimensions \
Name=DBClusterIdentifier,Value=my_cluster_id \
--output text | sort -k 3
VolumeBytesUsed
DATAPOINTS 182872522752.0 2020-08-04T17:41:00+00:00 Bytes
DATAPOINTS 182872522752.0 2020-08-04T18:41:00+00:00 Bytes
DATAPOINTS 182872522752.0 2020-08-04T19:41:00+00:00 Bytes
DATAPOINTS 182872522752.0 2020-08-04T20:41:00+00:00 Bytes
DATAPOINTS 187667791872.0 2020-08-04T21:41:00+00:00 Bytes
DATAPOINTS 190981029888.0 2020-08-04T22:41:00+00:00 Bytes
DATAPOINTS 195587244032.0 2020-08-04T23:41:00+00:00 Bytes
DATAPOINTS 201048915968.0 2020-08-05T00:41:00+00:00 Bytes
DATAPOINTS 205368492032.0 2020-08-05T01:41:00+00:00 Bytes
DATAPOINTS 206827454464.0 2020-08-05T02:41:00+00:00 Bytes
DATAPOINTS 206827454464.0 2020-08-05T03:41:00+00:00 Bytes
DATAPOINTS 206827454464.0 2020-08-05T04:41:00+00:00 Bytes
DATAPOINTS 206827454464.0 2020-08-05T05:41:00+00:00 Bytes
DATAPOINTS 206827454464.0 2020-08-05T06:41:00+00:00 Bytes
DATAPOINTS 206827454464.0 2020-08-05T07:41:00+00:00 Bytes
DATAPOINTS 206827454464.0 2020-08-05T08:41:00+00:00 Bytes
DATAPOINTS 206827454464.0 2020-08-05T09:41:00+00:00 Bytes
DATAPOINTS 206827454464.0 2020-08-05T10:41:00+00:00 Bytes
DATAPOINTS 206827454464.0 2020-08-05T11:41:00+00:00 Bytes
DATAPOINTS 206827454464.0 2020-08-05T12:41:00+00:00 Bytes
DATAPOINTS 206827454464.0 2020-08-05T13:41:00+00:00 Bytes
DATAPOINTS 206827454464.0 2020-08-05T14:41:00+00:00 Bytes
DATAPOINTS 206833664000.0 2020-08-05T15:41:00+00:00 Bytes
DATAPOINTS 206833664000.0 2020-08-05T16:41:00+00:00 Bytes

```

The sorted output shows how much storage was used at the start and end of the monitoring period. You can also find the points during that period when Aurora allocated more storage for the cluster. The following example uses Linux commands to reformat the starting and ending `VolumeBytesUsed` values as gigabytes (GB) and as gibibytes (GiB). Gigabytes represent units measured in powers of 10 and are

commonly used in discussions of storage for rotational hard drives. Gibibytes represent units measured in powers of 2. Aurora storage measurements and limits are typically stated in the power-of-2 units, such as gibibytes and tebibytes.

```
$ GiB=$((1024*1024*1024))
$ GB=$((1000*1000*1000))
$ echo "Start: $((182872522752/$GiB)) GiB, End: $((206833664000/$GiB)) GiB"
Start: 170 GiB, End: 192 GiB
$ echo "Start: $((182872522752/$GB)) GB, End: $((206833664000/$GB)) GB"
Start: 182 GB, End: 206 GB
```

The `VolumeBytesUsed` metric tells you how much storage in the cluster is incurring charges. Thus, it's best to minimize this number when practical. However, this metric doesn't include some storage that Aurora uses internally in the cluster and doesn't charge for. If your cluster is approaching the storage limit and might run out of space, it's more helpful to monitor the `AuroraVolumeBytesLeftTotal` metric and try to maximize that number. The following example runs a similar calculation as the previous one, but for `AuroraVolumeBytesLeftTotal` instead of `VolumeBytesUsed`. You can see that the free size for this cluster reflects the original 64 TiB limit, because the cluster is running Aurora MySQL version 1.22.

```
$ aws cloudwatch get-metric-statistics --metric-name "AuroraVolumeBytesLeftTotal" \
--start-time "$($date -d '1 hour ago')" --end-time "$($date -d 'now')" --period 3600 \
--namespace "AWS/RDS" --statistics Maximum --dimensions
Name=DBClusterIdentifier,Value=my_old_cluster_id \
--output text | sort -k 3
AuroraVolumeBytesLeftTotal
DATAPOINTS      69797193744384.0      2020-08-05T17:52:00+00:00      Count
DATAPOINTS      69797193744384.0      2020-08-05T18:52:00+00:00      Count
$ TiB=$((1024*1024*1024*1024))
$ TB=$((1000*1000*1000*1000))
$ echo "$((69797067915264 / $TB)) TB remaining for this cluster"
69 TB remaining for this cluster
$ echo "$((69797067915264 / $TiB)) TiB remaining for this cluster"
63 TiB remaining for this cluster
```

For a cluster running Aurora MySQL version 1.23 or 2.09 and higher, or Aurora PostgreSQL 3.3.0 or 2.6.0 and higher, the free size reported by `VolumeBytesUsed` increases when data is added and decreases when data is removed. The following example shows how. This report shows the maximum and minimum storage size for a cluster at 15-minute intervals as tables with temporary data are created and dropped. The report lists the maximum value before the minimum value. Thus, to understand how storage usage changed within the 15-minute interval, interpret the numbers from right to left.

```
$ aws cloudwatch get-metric-statistics --metric-name "VolumeBytesUsed" \
--start-time "$($date -d '4 hours ago')" --end-time "$($date -d 'now')" --period 1800 \
--namespace "AWS/RDS" --statistics Maximum Minimum --dimensions
Name=DBClusterIdentifier,Value=my_new_cluster_id
--output text | sort -k 4
VolumeBytesUsed
DATAPOINTS 14545305600.0 14545305600.0 2020-08-05T20:49:00+00:00 Bytes
DATAPOINTS 14545305600.0 14545305600.0 2020-08-05T21:19:00+00:00 Bytes
DATAPOINTS 22022176768.0 14545305600.0 2020-08-05T21:49:00+00:00 Bytes
DATAPOINTS 22022176768.0 22022176768.0 2020-08-05T22:19:00+00:00 Bytes
DATAPOINTS 22022176768.0 22022176768.0 2020-08-05T22:49:00+00:00 Bytes
DATAPOINTS 22022176768.0 15614263296.0 2020-08-05T23:19:00+00:00 Bytes
DATAPOINTS 15614263296.0 15614263296.0 2020-08-05T23:49:00+00:00 Bytes
DATAPOINTS 15614263296.0 15614263296.0 2020-08-06T00:19:00+00:00 Bytes
```

The following example shows how with a cluster running Aurora MySQL version 1.23 or 2.09 and higher, or Aurora PostgreSQL 3.3.0 or 2.6.0 and higher, the free size reported by `AuroraVolumeBytesLeftTotal` reflects the higher 128 TiB size limit.

```
$ aws cloudwatch get-metric-statistics --region us-east-1 --metric-name
"AuroraVolumeBytesLeftTotal" \
--start-time "$(date -d '4 hours ago')" --end-time "$(date -d 'now')" --period 1800 \
--namespace "AWS/RDS" --statistics Minimum --dimensions
Name=DBClusterIdentifier,Value=pg-57 \
--output text | sort -k 3
AuroraVolumeBytesLeftTotal
DATAPOINTS 140515818864640.0 2020-08-05T20:56:00+00:00 Count
DATAPOINTS 140515818864640.0 2020-08-05T21:26:00+00:00 Count
DATAPOINTS 140515818864640.0 2020-08-05T21:56:00+00:00 Count
DATAPOINTS 140514866757632.0 2020-08-05T22:26:00+00:00 Count
DATAPOINTS 140511020580864.0 2020-08-05T22:56:00+00:00 Count
DATAPOINTS 140503168843776.0 2020-08-05T23:26:00+00:00 Count
DATAPOINTS 140503168843776.0 2020-08-05T23:56:00+00:00 Count
DATAPOINTS 140515818864640.0 2020-08-06T00:26:00+00:00 Count
$ TiB=$((1024*1024*1024))
$ TB=$((1000*1000*1000))
$ echo "$((140515818864640 / $TB)) TB remaining for this cluster"
140 TB remaining for this cluster
$ echo "$((140515818864640 / $TiB)) TiB remaining for this cluster"
127 TiB remaining for this cluster
```

Instance scaling

You can scale your Aurora DB cluster as needed by modifying the DB instance class for each DB instance in the DB cluster. Aurora supports several DB instance classes optimized for Aurora, depending on database engine compatibility.

Database engine	Instance scaling
Amazon Aurora MySQL	See Scaling Aurora MySQL DB instances (p. 812)
Amazon Aurora PostgreSQL	See Scaling Aurora PostgreSQL DB instances (p. 1360)

Read scaling

You can achieve read scaling for your Aurora DB cluster by creating up to 15 Aurora Replicas in a DB cluster that uses single-master replication. Each Aurora Replica returns the same data from the cluster volume with minimal replica lag—usually considerably less than 100 milliseconds after the primary instance has written an update. As your read traffic increases, you can create additional Aurora Replicas and connect to them directly to distribute the read load for your DB cluster. Aurora Replicas don't have to be of the same DB instance class as the primary instance.

For information about adding Aurora Replicas to a DB cluster, see [Adding Aurora Replicas to a DB cluster \(p. 392\)](#).

Managing connections

The maximum number of connections allowed to an Aurora DB instance is determined by the `max_connections` parameter in the instance-level parameter group for the DB instance. The default value of that parameter varies depends on the DB instance class used for the DB instance and database engine compatibility.

Database engine	max_connections default value
Amazon Aurora MySQL	See Maximum connections to an Aurora MySQL DB instance (p. 813)
Amazon Aurora PostgreSQL	See Maximum connections to an Aurora PostgreSQL DB instance (p. 1360)

If your applications frequently open and close connections, or have long-lived connections that approach or exceed the specified limits, we recommend using Amazon RDS Proxy. RDS Proxy is a fully managed, highly available database proxy that uses connection pooling to share database connections securely and efficiently. To learn more about RDS Proxy, see [Using Amazon RDS Proxy \(p. 288\)](#).

Managing query execution plans

If you use query plan management for Aurora PostgreSQL, you gain control over which plans the optimizer runs. For more information, see [Managing query execution plans for Aurora PostgreSQL \(p. 1460\)](#).

Cloning a volume for an Aurora DB cluster

By using Aurora cloning, you can quickly and cost-effectively create a new cluster that uses the same Aurora cluster volume and has the same data as the original. The new cluster with its associated data volume is known as a *clone*. Creating a clone is faster and more space-efficient than physically copying the data using other techniques, such as restoring a snapshot.

Aurora supports many different types of cloning. You can create an Aurora provisioned clone from a provisioned Aurora DB cluster. You can create an Aurora Serverless v1 clone from an Aurora Serverless v1 DB cluster. But you can also create Aurora Serverless v1 clones from Aurora provisioned DB clusters, and you can create provisioned clones from Aurora Serverless v1 DB clusters. When you create a clone using a different deployment configuration than the source, the clone is created using the latest minor version of the source's Aurora DB engine.

A cloned Aurora Serverless DB cluster has the same behavior and limitations as any Aurora Serverless v1 DB cluster. For more information, see [Using Amazon Aurora Serverless v1 \(p. 147\)](#).

When you create clones from your Aurora DB clusters, the clones are created in your AWS account—the same account that owns the source Aurora DB cluster. However, you can also share provisioned Aurora DB clusters and clones with other AWS accounts. For more information, see [Cross-account cloning with AWS RAM and Amazon Aurora \(p. 415\)](#).

Cross-account cloning currently doesn't support cloning Aurora Serverless v1 DB clusters. For more information, see [Limitations of cross-account cloning \(p. 415\)](#).

Topics

- [Overview of Aurora cloning \(p. 402\)](#)
- [Limitations of Aurora cloning \(p. 403\)](#)
- [How Aurora cloning works \(p. 403\)](#)
- [Creating an Amazon Aurora clone \(p. 406\)](#)
- [Cross-account cloning with AWS RAM and Amazon Aurora \(p. 415\)](#)

Overview of Aurora cloning

Aurora uses a *copy-on-write protocol* to create a clone. This mechanism uses minimal additional space to create an initial clone. When the clone is first created, Aurora keeps a single copy of the data that is used by the source Aurora DB cluster and the new (cloned) Aurora DB cluster. Additional storage is allocated only when changes are made to data (on the Aurora storage volume) by the source Aurora DB cluster or the Aurora DB cluster clone. To learn more about the copy-on-write protocol, see [How Aurora cloning works \(p. 403\)](#).

Aurora cloning is especially useful for quickly setting up test environments using your production data, without risking data corruption. You can use clones for many types of applications, such as the following:

- Experiment with potential changes (schema changes and parameter group changes, for example) to assess all impacts.
- Run workload-intensive operations, such as exporting data or running analytical queries on the clone.
- Create a copy of your production DB cluster for development, testing, or other purposes.

You can create more than one clone from the same Aurora DB cluster. You can also create multiple clones from another clone.

After creating an Aurora clone, you can configure the Aurora DB instances differently from the source Aurora DB cluster. For example, you might not need a clone for development purposes to meet the same

high availability requirements as the source production Aurora DB cluster. In this case, you can configure the clone with a single Aurora DB instance rather than the multiple DB instances used by the Aurora DB cluster.

When you finish using the clone for your testing, development, or other purposes, you can delete it.

Limitations of Aurora cloning

Aurora cloning currently has the following limitations:

- You can't create a clone in a different AWS Region than the source Aurora DB cluster.
- You can't create an Aurora Serverless v1 clone from a nonencrypted provisioned Aurora DB cluster.
- You can't create a Aurora Serverless v1 clone from a MySQL 5.6-compatible provisioned cluster, or a provisioned clone of a MySQL 5.6-compatible Aurora Serverless v1 cluster.
- You can't create more than 15 clones based on a copy or based on another clone. After creating 15 clones, you can create copies only. However, you can create up to 15 clones of each copy.
- You can't create a clone from an Aurora DB cluster without the parallel query feature to a cluster that uses parallel query. To bring data into a cluster that uses parallel query, create a snapshot of the original cluster and restore it to the cluster that's using the parallel query feature.
- You can't create a clone from an Aurora DB cluster that has no DB instances. You can only clone Aurora DB clusters that have at least one DB instance.
- You can create a clone in a different virtual private cloud (VPC) than that of the Aurora DB cluster. If you do, the subnets of the VPCs must map to the same Availability Zones.

How Aurora cloning works

Aurora cloning works at the storage layer of an Aurora DB cluster. It uses a *copy-on-write* protocol that's both fast and space-efficient in terms of the underlying durable media supporting the Aurora storage volume. You can learn more about Aurora cluster volumes in the [Overview of Aurora storage \(p. 64\)](#).

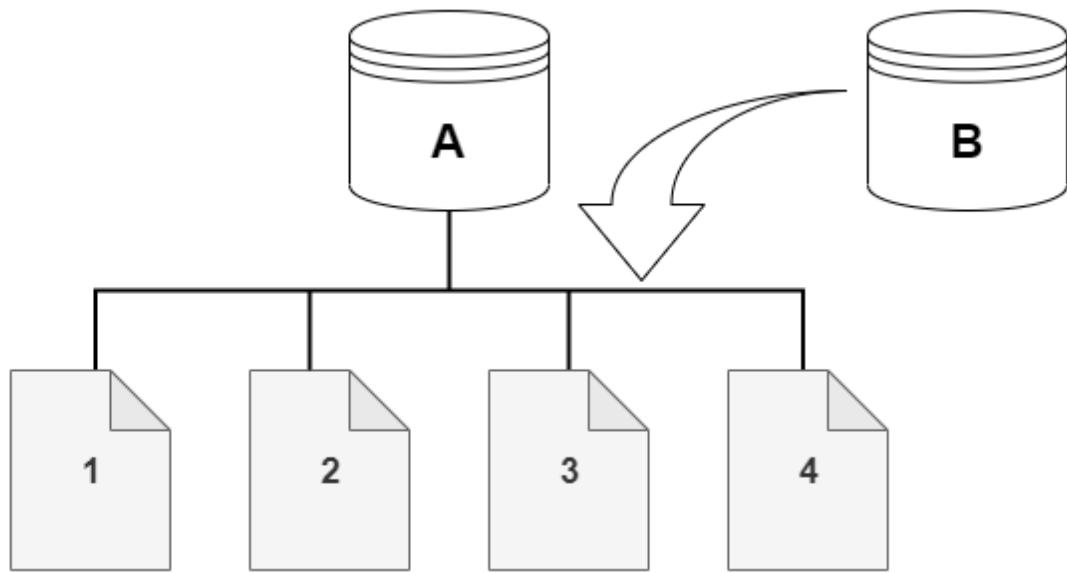
Topics

- [Understanding the copy-on-write protocol \(p. 403\)](#)
- [Deleting a source cluster volume \(p. 406\)](#)

Understanding the copy-on-write protocol

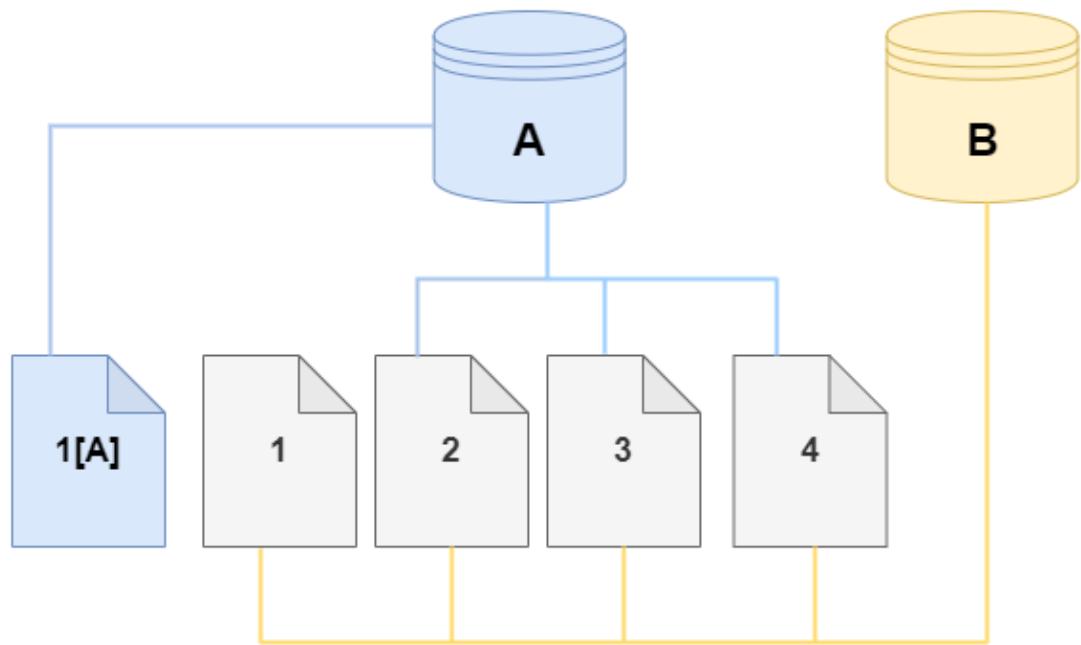
An Aurora DB cluster stores data in pages in the underlying Aurora storage volume.

For example, in the following diagram you can find an Aurora DB cluster (A) that has four data pages, 1, 2, 3, and 4. Imagine that a clone, B, is created from the Aurora DB cluster. When the clone is created, no data is copied. Rather, the clone points to the same set of pages as the source Aurora DB cluster.

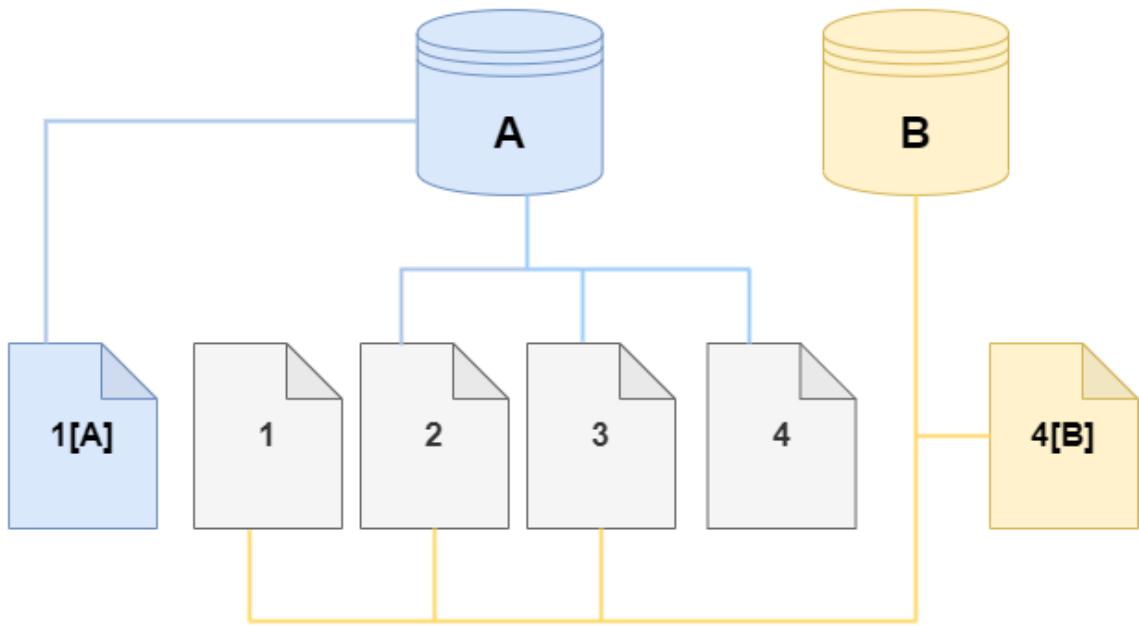


When the clone is created, no additional storage is usually needed. The copy-on-write protocol uses the same segment on the physical storage media as the source segment. Additional storage is required only if the capacity of the source segment isn't sufficient for the entire clone segment. If that's the case, the source segment is copied to another physical device.

In the following diagrams, you can find an example of the copy-on-write protocol in action using the same cluster A and its clone, B, as shown preceding. Let's say that you make a change to your Aurora DB cluster (A) that results in a change to data held on page 1. Instead of writing to the original page 1, Aurora creates a new page 1[A]. The Aurora DB cluster volume for cluster (A) now points to page 1[A], 2, 3, and 4, while the clone (B) still references the original pages.



On the clone, a change is made to page 4 on the storage volume. Instead of writing to the original page 4, Aurora creates a new page, 4[B]. The clone now points to pages 1, 2, 3, and to page 4[B], while the cluster (A) continues pointing to 1[A], 2, 3, and 4.



As more changes occur over time in both the source Aurora DB cluster volume and the clone, more storage is needed to capture and store the changes.

Deleting a source cluster volume

When you delete a source cluster volume that has one or more clones associated with it, the clones aren't affected. The clones continue to point to the pages that were previously owned by the source cluster volume.

Creating an Amazon Aurora clone

You can create a clone in the same AWS account as the source Aurora DB cluster. To do so, you can use the AWS Management Console or the AWS CLI and the procedures following.

To allow another AWS account to create a clone or to share a clone with another AWS account, use the procedures in [Cross-account cloning with AWS RAM and Amazon Aurora \(p. 415\)](#).

By using Aurora cloning, you can do the following types of cloning operations:

- Create a provisioned Aurora DB cluster clone from a provisioned Aurora DB cluster.
- Create an Aurora Serverless v1 cluster clone from an Aurora Serverless v1 DB cluster.
- Create an Aurora Serverless v1 DB cluster clone from a provisioned Aurora DB cluster.
- Create an Aurora provisioned DB cluster clone from an Aurora Serverless v1 DB cluster.

Aurora Serverless v1 DB clusters are always encrypted. When you clone an Aurora Serverless v1 DB cluster into a provisioned Aurora DB cluster, the provisioned Aurora DB cluster is encrypted. You can

choose the encryption key, but you can't disable the encryption. To clone from a provisioned Aurora DB cluster to an Aurora Serverless v1 cluster, you need an encrypted provisioned Aurora DB cluster.

Console

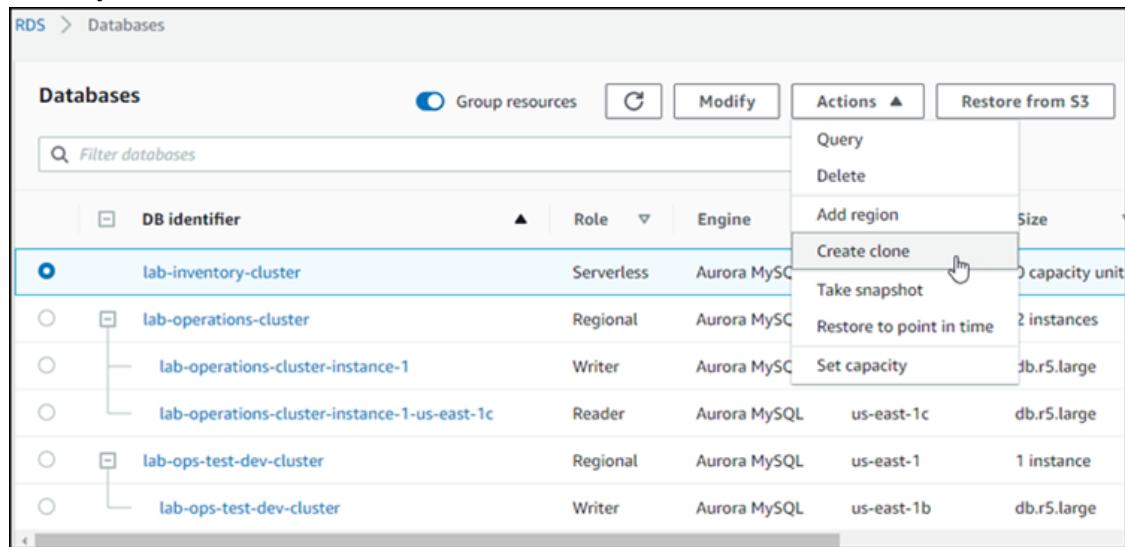
The following procedure describes how to clone an Aurora DB cluster using the AWS Management Console.

Creating a clone using the AWS Management Console results in an Aurora DB cluster with one Aurora DB instance.

These instructions apply for DB clusters owned by the same AWS account that is creating the clone. If the DB cluster is owned by a different AWS account, see [Cross-account cloning with AWS RAM and Amazon Aurora \(p. 415\)](#) instead.

To create a clone of a DB cluster owned by your AWS account using the AWS Management Console

1. Sign in to the AWS Management Console and open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. In the navigation pane, choose **Databases**.
3. Choose your Aurora DB cluster from the list, and for **Actions**, choose **Create clone**.



The Create clone page opens, where you can configure **Instance specifications**, **Connectivity**, and other options for the Aurora DB cluster clone.

4. In the **Instance specifications** section, do the following:
 - a. For **DB cluster identifier**, enter the name that you want to give to your cloned Aurora DB cluster.

Instance specifications

DB engine

Aurora MySQL

Source DB instance [Info](#)

lab-inventory-cluster

DB instance identifier [Info](#)

Type a name for your DB instance. The name must be unique across all DB instances owned by your AWS account in the current AWS Region.

lab-inventory-audit-cluster

The DB instance identifier is case-insensitive, but is stored as all lowercase (as in "mydbinstance"). Constraints: 1 to 60 alphanumeric characters or hyphens (1 to 15 for SQL Server). First character must be a letter. Can't contain two consecutive hyphens. Can't end with a hyphen.

Capacity type [Info](#)

Provisioned

You provision and manage the server instance sizes.

Serverless

You specify the minimum and maximum amount of resources needed, and Aurora scales the capacity based on database load. This is a good option for intermittent or unpredictable workloads.

- b. For **Capacity type**, choose **Provisioned** or **Serverless** as needed for your use case.

You can choose **Serverless** only if the source Aurora DB cluster is an Aurora Serverless v1 DB cluster or is a provisioned Aurora DB cluster that is encrypted.

- If you choose **Provisioned**, you see a **DB instance size** configuration card.

DB instance size

DB instance class [Info](#)

Choose a DB instance class that meets your processing power and memory requirements. The DB instance class options below are limited to those supported by the engine you selected above.

Memory Optimized classes (includes r classes)

Burstable classes (includes t classes)

db.r5.large

2 vCPUs 16 GiB RAM Network: 4,750 Mbps

Include previous generation classes

You can accept the provided setting, or you can use a different DB instance class for your clone.

- If you choose **Serverless**, you see a **Capacity settings** configuration card.

Capacity settings

This billing estimate is based on published prices. [Learn more](#)

Minimum Aurora capacity unit Info	Maximum Aurora capacity unit Info
1 2GB RAM	64 122GB RAM

[► Additional scaling configuration](#)

You can accept the provided settings, or you can change them for your use case.

- c. For **Additional configuration**, choose settings as you usually do for your Aurora DB clusters.

Additional settings include your choice for the database name and whether you want to use many optional features. These features include backup, Enhanced Monitoring, exporting logs to Amazon CloudWatch, deletion protection, and so on.

Some of the choices displayed depend on the type of clone that you are creating. For example, Aurora Serverless doesn't support Amazon RDS Performance Insights, so that option isn't shown in this case.

Encryption is a standard option available in **Additional configuration**. Aurora Serverless DB clusters are always encrypted. You can create an Aurora Serverless clone only from an Aurora Serverless DB cluster or an encrypted provisioned Aurora DB cluster. However, you can choose a different key for the Aurora Serverless clone than that used for the encrypted provisioned cluster.

Encryption

Enable encryption
Choose to encrypt the given instance. Master key IDs and aliases appear in the list after they have been created using the AWS Key Management Service console. [Info](#)

Master key [Info](#)

lab-tester-key

Account

KMS key ID

When you create an Aurora Serverless clone from an Aurora Serverless DB cluster, you can choose a different key for the clone.



- d. Finish entering all settings for your Aurora DB cluster clone. To learn more about Aurora DB cluster and instance settings, see [Creating an Amazon Aurora DB cluster \(p. 125\)](#).
5. Choose **Create clone** to launch the Aurora clone of your chosen Aurora DB cluster.

When the clone is created, it's listed with your other Aurora DB clusters in the console **Databases** section and displays its current state. Your clone is ready to use when its state is **Available**.

AWS CLI

Using the AWS CLI for cloning your Aurora DB cluster involves a couple of steps.

The `restore-db-cluster-to-point-in-time` AWS CLI command that you use results in an empty Aurora DB cluster with 0 Aurora DB instances. That is, the command restores only the Aurora DB cluster, not the DB instances for that cluster. You do that separately after the clone is available. The two steps in the process are as follows:

1. Create the clone by using the `restore-db-cluster-to-point-in-time` CLI command. The parameters that you use with this command control the capacity type and other details of the empty Aurora DB cluster (clone) being created.
2. Create the Aurora DB instance for the clone by using the `create-db-instance` CLI command to recreate the Aurora DB instance in the restored Aurora DB cluster.

The commands following assume that the AWS CLI is set up with your AWS Region as the default. This approach saves you from passing the `--region` name in each of the commands. For more information, see [Configuring the AWS CLI](#). You can also specify the `--region` in each of the CLI commands that follow.

Topics

- [Creating the clone \(p. 410\)](#)
- [Checking the status and getting clone details \(p. 413\)](#)
- [Creating the Aurora DB instance for your clone \(p. 413\)](#)
- [Parameters to use for cloning \(p. 414\)](#)

Creating the clone

The specific parameters that you pass to the `restore-db-cluster-to-point-in-time` CLI command vary. What you pass depends on the engine-mode type of the source DB cluster—Serverless or Provisioned—and the type of clone that you want to create.

Use the following procedure to create an Aurora Serverless clone from an Aurora Serverless DB cluster, or to create a provisioned Aurora clone from a provisioned Aurora DB cluster.

To create a clone of the same engine mode as the source Aurora DB cluster

- Use the `restore-db-cluster-to-point-in-time` CLI command and specify values for the following parameters:
 - `--db-cluster-identifier` – Choose a meaningful name for your clone. You name the clone when you use the `restore-db-cluster-to-point-in-time` CLI command. You then pass the name of the clone in the `create-db-instance` CLI command.
 - `--restore-type` – Use `copy-on-write` to create a clone of the source DB cluster. Without this parameter, the `restore-db-cluster-to-point-in-time` restores the Aurora DB cluster rather than creating a clone.
 - `--source-db-cluster-identifier` – Use the name of the source Aurora DB cluster that you want to clone.
 - `--use-latest-restorable-time` – This value points to the latest restorable volume data for the clone.

The following example creates a clone named `my-clone` from a cluster named `my-source-cluster`.

For Linux, macOS, or Unix:

```
aws rds restore-db-cluster-to-point-in-time \
--source-db-cluster-identifier my-source-cluster \
--db-cluster-identifier my-clone \
--restore-type copy-on-write \
--use-latest-restorable-time
```

For Windows:

```
aws rds restore-db-cluster-to-point-in-time ^
--source-db-cluster-identifier my-source-cluster ^
--db-cluster-identifier my-clone ^
--restore-type copy-on-write ^
--use-latest-restorable-time
```

The command returns the JSON object containing details of the clone. Check to make sure that your cloned DB cluster is available before trying to create the DB instance for your clone. For more information, see [Checking the status and getting clone details \(p. 413\)](#).

To create a clone with a different engine mode than the source Aurora DB cluster

- Use the `restore-db-cluster-to-point-in-time` CLI command and specify values for the following parameters:
 - `--db-cluster-identifier` – Choose a meaningful name for your clone. You name the clone when you use the `restore-db-cluster-to-point-in-time` CLI command. You then pass the name of the clone in the `create-db-instance` CLI command.
 - `--engine-mode` – Use this parameter only to create clones that are of a different type than the source Aurora DB cluster. Choose the value to pass with `--engine-mode` as follows:
 - Use `provisioned` to create a provisioned Aurora DB cluster clone from an Aurora Serverless DB cluster.
 - Use `serverless` to create an Aurora Serverless DB cluster clone from a provisioned Aurora DB cluster. When you specify `serverless` engine mode, you can also choose `--scaling-configuration`.
 - `--restore-type` – Use `copy-on-write` to create a clone of the source DB cluster. Without this parameter, the `restore-db-cluster-to-point-in-time` restores the Aurora DB cluster rather than creating a clone.

- **--scaling-configuration** – (Optional) Use only with **--engine-mode serverless** to configure the minimum and maximum capacity for the clone. If you don't use this parameter, Aurora creates the clone using a minimum capacity of 1. It uses a maximum capacity that matches the capacity of the source provisioned Aurora DB cluster.
- **--source-db-cluster-identifier** – Use the name of the source Aurora DB cluster that you want to clone.
- **--use-latest-restorable-time** – This value points to the latest restorable volume data for the clone.

The following example creates an Aurora Serverless clone (`my-clone`) from a provisioned Aurora DB cluster named `my-source-cluster`. The provisioned Aurora DB cluster is encrypted.

For Linux, macOS, or Unix:

```
aws rds restore-db-cluster-to-point-in-time \
  --source-db-cluster-identifier my-source-cluster \
  --db-cluster-identifier my-clone \
  --engine-mode serverless \
  --scaling-configuration MinCapacity=8, MaxCapacity=64 \
  --restore-type copy-on-write \
  --use-latest-restorable-time
```

For Windows:

```
aws rds restore-db-cluster-to-point-in-time ^
  --source-db-cluster-identifier my-source-cluster ^
  --db-cluster-identifier my-clone ^
  --engine-mode serverless ^
  --scaling-configuration MinCapacity=8, MaxCapacity=64 ^
  --restore-type copy-on-write ^
  --use-latest-restorable-time
```

These commands return the JSON object containing details of the clone that you need to create the DB instance. You can't do that until the status of the clone (the empty Aurora DB cluster) has the status **Available**.

Note

The `restore-db-cluster-to-point-in-time` AWS CLI command only restores the DB cluster, not the DB instances for that DB cluster. You must invoke the `create-db-instance` command to create DB instances for the restored DB cluster, specifying the identifier of the restored DB cluster in `--db-cluster-identifier`. You can create DB instances only after the `restore-db-cluster-to-point-in-time` command has completed and the DB cluster is available.

For example, suppose you have a cluster named `tpch100g` that you want to clone. The following Linux example creates a cloned cluster named `tpch100g-clone` and a primary instance named `tpch100g-clone-instance` for the new cluster. You don't need to supply some parameters, such as `--master-username` `reinvent` and `--master-user-password`. Aurora automatically determines those from the original cluster. You do need to specify the DB engine to use. Thus, the example tests the new cluster to determine the right value to use for the `--engine` parameter.

```
$ aws rds restore-db-cluster-to-point-in-time \
  --source-db-cluster-identifier tpch100g \
  --db-cluster-identifier tpch100g-clone \
  --restore-type copy-on-write \
  --use-latest-restorable-time

$ aws rds describe-db-clusters \
  --db-cluster-identifier tpch100g-clone \
```

```
--query '*[].[Engine]' \
--output text
aurora

$ aws rds create-db-instance \
--db-instance-identifier tpch100g-clone-instance \
--db-cluster-identifier tpch100g-clone \
--db-instance-class db.r5.4xlarge \
--engine aurora
```

Checking the status and getting clone details

You can use the following command to check the status of your newly created empty DB cluster.

```
$ aws rds describe-db-clusters --db-cluster-identifier my-clone --query '*[].[Status]' --
output text
```

Or you can obtain the status and the other values that you need to [create the DB instance for your clone \(p. 413\)](#) by using the following AWS CLI query.

For Linux, macOS, or Unix:

```
aws rds describe-db-clusters --db-cluster-identifier my-clone \
--query '*[[].
{Status:Status,Engine:Engine,EngineVersion:EngineVersion,EngineMode:EngineMode}'
```

For Windows:

```
aws rds describe-db-clusters --db-cluster-identifier my-clone ^
--query "*[[].
{Status:Status,Engine:Engine,EngineVersion:EngineVersion,EngineMode:EngineMode}"
```

This query returns output similar to the following.

```
[

{
    "Status": "available",
    "Engine": "aurora-mysql",
    "EngineVersion": "5.7.mysql_aurora.2.09.1",
    "EngineMode": "provisioned"
}
```

Creating the Aurora DB instance for your clone

Use the [create-db-instance](#) CLI command to create the DB instance for your clone.

The `--db-instance-class` parameter is used for provisioned Aurora DB clusters only.

For Linux, macOS, or Unix:

```
aws rds create-db-instance \
--db-instance-identifier my-new-db \
--db-cluster-identifier my-clone \
--db-instance-class db.r5.4xlarge \
--engine aurora-mysql
```

For Windows:

```
aws rds create-db-instance ^
--db-instance-identifier my-new-db ^
--db-cluster-identifier my-clone ^
--db-instance-class db.r5.4xlarge ^
--engine aurora-mysql
```

For an Aurora Serverless clone created from an Aurora Serverless DB cluster, you specify only a few parameters. The DB instance inherits the --engine-mode, --master-username, and --master-user-password properties from the source DB cluster. You can change the --scaling-configuration.

For Linux, macOS, or Unix:

```
aws rds create-db-instance \
--db-instance-identifier my-new-db \
--db-cluster-identifier my-clone \
--engine aurora-postgresql
```

For Windows:

```
aws rds create-db-instance ^
--db-instance-identifier my-new-db ^
--db-cluster-identifier my-clone ^
--engine aurora-postgresql
```

Parameters to use for cloning

The following table summarizes the various parameters used with `restore-db-cluster-to-point-in-time` to clone Aurora DB clusters.

Parameter	Description
--source-db-cluster-identifier	Use the name of the source Aurora DB cluster that you want to clone.
--db-cluster-identifier	Choose a meaningful name for your clone. You name your clone with the <code>restore-db-cluster-to-point-in-time</code> command. Then you pass this name to the <code>create-db-instance</code> command.
--engine-mode	Use this parameter to create clones that are of a different type than the source Aurora DB cluster. Choose the value to pass with --engine-mode as follows: <ul style="list-style-type: none"> Use <code>provisioned</code> to create a provisioned Aurora DB cluster clone from an Aurora Serverless DB cluster. Use <code>serverless</code> to create an Aurora Serverless DB cluster clone from a provisioned Aurora DB cluster. When you specify <code>serverless</code> engine mode, you can also choose --scaling-configuration
--restore-type	Specify <code>copy-on-write</code> as the --restore-type to create a clone of the source DB cluster rather than restoring the source Aurora DB cluster.
--scaling-configuration	Use this parameter with --engine-mode <code>serverless</code> to configure the minimum and maximum capacity for the clone. If you don't use this parameter, Aurora creates the Aurora Serverless clone using a minimum capacity of 1 and a maximum capacity of 16.

Parameter	Description
--use-latest-restorable-time	This value points to the latest restorable volume data for the clone.

Cross-account cloning with AWS RAM and Amazon Aurora

By using AWS Resource Access Manager (AWS RAM) with Amazon Aurora, you can share Aurora DB clusters and clones that belong to your AWS account with another AWS account or organization. Such *cross-account cloning* is much faster than creating and restoring a database snapshot. You can create a clone of one of your Aurora DB clusters and share the clone. Or you can share your Aurora DB cluster with another AWS account and let the account holder create the clone. The approach that you choose depends on your use case.

For example, you might need to regularly share a clone of your financial database with your organization's internal auditing team. In this case, your auditing team has its own AWS account for the applications that it uses. You can give the auditing team's AWS account the permission to access your Aurora DB cluster and clone it as needed.

On the other hand, if an outside vendor audits your financial data you might prefer to create the clone yourself. You then give the outside vendor access to the clone only.

You can also use cross-account cloning to support many of the same use cases for cloning within the same AWS account, such as development and testing. For example, your organization might use different AWS accounts for production, development, testing, and so on. For more information, see [Overview of Aurora cloning \(p. 402\)](#).

Thus, you might want to share a clone with another AWS account or allow another AWS account to create clones of your Aurora DB clusters. In either case, start by using AWS RAM to create a share object. For complete information about sharing AWS resources between AWS accounts, see the [AWS RAM User Guide](#).

Creating a cross-account clone requires actions from the AWS account that owns the original cluster, and the AWS account that creates the clone. First, the original cluster owner modifies the cluster to allow one or more other accounts to clone it. If any of the accounts is in a different AWS organization, AWS generates a sharing invitation. The other account must accept the invitation before proceeding. Then each authorized account can clone the cluster. Throughout this process, the cluster is identified by its unique Amazon Resource Name (ARN).

As with cloning within the same AWS account, additional storage space is used only if changes are made to the data by the source or the clone. Charges for storage are then applied at that time. If the source cluster is deleted, storage costs are distributed equally among remaining cloned clusters.

Topics

- [Limitations of cross-account cloning \(p. 415\)](#)
- [Allowing other AWS accounts to clone your cluster \(p. 416\)](#)
- [Cloning a cluster that is owned by another AWS account \(p. 418\)](#)

Limitations of cross-account cloning

Aurora cross-account cloning has the following limitations:

- You can't clone an Aurora Serverless cluster across AWS accounts.

- You can't view or accept invitations to shared resources with the AWS Management Console. Use the AWS CLI, the Amazon RDS API, or the AWS RAM console to view and accept invitations to shared resources.
- You can't create new clones from a clone that's been shared with your AWS account.
- You can't share resources (clones or Aurora DB clusters) that have been shared with your AWS account.
- You can't create more than 15 cross-account clones from any single Aurora DB cluster. Each of these 15 clones must be owned by a different AWS account. That is, you can only create one cross-account clone of a cluster within any AWS account.
- You can't share an Aurora DB cluster with other AWS accounts unless the cluster is in an ACTIVE state.
- You can't rename an Aurora DB cluster that's been shared with other AWS accounts.
- You can't create a cross-account clone of a cluster that is encrypted with the default RDS key.
- You can't create nonencrypted clones in one AWS account from encrypted Aurora DB clusters that have been shared by another AWS account. The cluster owner must grant permission to access the source cluster's AWS KMS key. However, you can use a different key when you create the clone.

Allowing other AWS accounts to clone your cluster

To allow other AWS accounts to clone a cluster that you own, use AWS RAM to set the sharing permission. Doing so also sends an invitation to each of the other accounts that's in a different AWS organization.

For the procedures to share resources owned by you in the AWS RAM console, see [Sharing resources owned by you](#) in the *AWS RAM User Guide*.

Topics

- [Granting permission to other AWS accounts to clone your cluster \(p. 416\)](#)
- [Checking if a cluster that you own is shared with other AWS accounts \(p. 418\)](#)

Granting permission to other AWS accounts to clone your cluster

If the cluster that you're sharing is encrypted, you also share the AWS KMS key for the cluster. You can allow AWS Identity and Access Management (IAM) users or roles in one AWS account to use a KMS key in a different account.

To do this, you first add the external account (root user) to the KMS key's key policy through AWS KMS. You don't add the individual IAM users or roles to the key policy, only the external account that owns them. You can only share a KMS key that you create, not the default RDS service key. For information about access control for KMS keys, see [Authentication and access control for AWS KMS](#).

Console

To grant permission to clone your cluster

1. Sign in to the AWS Management Console and open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. In the navigation pane, choose **Databases**.
3. Choose the DB cluster that you want to share to see its **Details** page, and choose the **Connectivity & security** tab.
4. In the **Share DB cluster with other AWS accounts** section, enter the numeric account ID for the AWS account that you want to allow to clone this cluster. For account IDs in the same organization, you can begin typing in the box and then choose from the menu.

Important

In some cases, you might want an account that is not in the same AWS organization as your account to clone a cluster. In these cases, for security reasons the console doesn't report who owns that account ID or whether the account exists.

Be careful entering account numbers that are not in the same AWS organization as your AWS account. Immediately verify that you shared with the intended account.

5. On the confirmation page, verify that the account ID that you specified is correct. Enter `share` in the confirmation box to confirm.

On the **Details** page, an entry appears that shows the specified AWS account ID under **Accounts that this DB cluster is shared with**. The **Status** column initially shows a status of **Pending**.

6. Contact the owner of the other AWS account, or sign in to that account if you own both of them. Instruct the owner of the other account to accept the sharing invitation and clone the DB cluster, as described following.

AWS CLI

To grant permission to clone your cluster

1. Gather the information for the required parameters. You need the ARN for your cluster and the numeric ID for the other AWS account.
2. Run the AWS RAM CLI command `create-resource-share`.

For Linux, macOS, or Unix:

```
aws ram create-resource-share --name descriptive_name \  
  --region region \  
  --resource-arns cluster_arn \  
  --principals other_account_ids
```

For Windows:

```
aws ram create-resource-share --name descriptive_name ^  
  --region region ^  
  --resource-arns cluster_arn ^  
  --principals other_account_ids
```

To include multiple account IDs for the `--principals` parameter, separate IDs from each other with spaces. To specify whether the permitted account IDs can be outside your AWS organization, include the `--allow-external-principals` or `--no-allow-external-principals` parameter for `create-resource-share`.

AWS RAM API

To grant permission to clone your cluster

1. Gather the information for the required parameters. You need the ARN for your cluster and the numeric ID for the other AWS account.
2. Call the AWS RAM API operation `CreateResourceShare`, and specify the following values:
 - Specify the account ID for one or more AWS accounts as the `principals` parameter.
 - Specify the ARN for one or more Aurora DB clusters as the `resourceArns` parameter.
 - Specify whether the permitted account IDs can be outside your AWS organization by including a Boolean value for the `allowExternalPrincipals` parameter.

Recreating a cluster that uses the default RDS key

If the encrypted cluster that you plan to share uses the default RDS key, make sure to recreate the cluster. To do this, create a manual snapshot of your DB cluster, use an AWS KMS key, and then restore the cluster to a new cluster. Then share the new cluster. To perform this process, take the following steps.

To recreate an encrypted cluster that uses the default RDS key

1. Sign in to the AWS Management Console and open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. Choose **Snapshots** from the navigation pane.
3. Choose your snapshot.
4. For **Actions**, choose **Copy Snapshot**, and then choose **Enable encryption**.
5. For **AWS KMS key**, choose the new encryption key that you want to use.
6. Restore the copied snapshot. To do so, follow the procedure in [Restoring from a DB cluster snapshot \(p. 497\)](#). The new DB instance uses your new encryption key.
7. (Optional) Delete the old DB cluster if you no longer need it. To do so, follow the procedure in [Deleting a DB cluster snapshot \(p. 539\)](#). Before you do, confirm that your new cluster has all necessary data and that your application can access it successfully.

Checking if a cluster that you own is shared with other AWS accounts

You can check if other users have permission to share a cluster. Doing so can help you understand whether the cluster is approaching the limit for the maximum number of cross-account clones.

For the procedures to share resources using the AWS RAM console, see [Sharing resources owned by you in the AWS RAM User Guide](#).

AWS CLI

To find out if a cluster that you own is shared with other AWS accounts

- Call the AWS RAM CLI command `list-principals`, using your account ID as the resource owner and the ARN of your cluster as the resource ARN. You can see all shares with the following command. The results indicate which AWS accounts are allowed to clone the cluster.

```
aws ram list-principals \
    --resource-arns your_cluster_arn \
    --principals your_aws_id
```

AWS RAM API

To find out if a cluster that you own is shared with other AWS accounts

- Call the AWS RAM API operation `ListPrincipals`. Use your account ID as the resource owner and the ARN of your cluster as the resource ARN.

Cloning a cluster that is owned by another AWS account

To clone a cluster that's owned by another AWS account, use AWS RAM to get permission to make the clone. After you have the required permission, use the standard procedure for cloning an Aurora cluster.

You can also check whether a cluster that you own is a clone of a cluster owned by a different AWS account.

For the procedures to work with resources owned by others in the AWS RAM console, see [Accessing resources shared with you](#) in the *AWS RAM User Guide*.

Topics

- [Viewing invitations to clone clusters that are owned by other AWS accounts \(p. 419\)](#)
- [Accepting invitations to share clusters owned by other AWS accounts \(p. 419\)](#)
- [Cloning an Aurora cluster that is owned by another AWS account \(p. 420\)](#)
- [Checking if a DB cluster is a cross-account clone \(p. 423\)](#)

Viewing invitations to clone clusters that are owned by other AWS accounts

To work with invitations to clone clusters owned by AWS accounts in other AWS organizations, use the AWS CLI, the AWS RAM console, or the AWS RAM API. Currently, you can't perform this procedure using the Amazon RDS console.

For the procedures to work with invitations in the AWS RAM console, see [Accessing resources shared with you](#) in the *AWS RAM User Guide*.

AWS CLI

To see invitations to clone clusters that are owned by other AWS accounts

1. Run the AWS RAM CLI command [get-resource-share-invitations](#).

```
aws ram get-resource-share-invitations --region region_name
```

The results from the preceding command show all invitations to clone clusters, including any that you already accepted or rejected.

2. (Optional) Filter the list so you see only the invitations that require action from you. To do so, add the parameter `--query 'resourceShareInvitations[?status==`PENDING`]`.

AWS RAM API

To see invitations to clone clusters that are owned by other AWS accounts

1. Call the AWS RAM API operation [GetResourceShareInvitations](#). This operation returns all such invitations, including any that you already accepted or rejected.
2. (Optional) Find only the invitations that require action from you by checking the `resourceShareAssociations` return field for a `status` value of PENDING.

Accepting invitations to share clusters owned by other AWS accounts

You can accept invitations to share clusters owned by other AWS accounts that are in different AWS organizations. To work with these invitations, use the AWS CLI, the AWS RAM and RDS APIs, or the AWS RAM console. Currently, you can't perform this procedure using the RDS console.

For the procedures to work with invitations in the AWS RAM console, see [Accessing resources shared with you](#) in the *AWS RAM User Guide*.

Console

To accept an invitation to share a cluster from another AWS account

1. Find the invitation ARN by running the AWS RAM CLI command [get-resource-share-invitations](#), as shown preceding.

2. Accept the invitation by calling the AWS RAM CLI command [accept-resource-share-invitation](#), as shown following.

For Linux, macOS, or Unix:

```
aws ram accept-resource-share-invitation \
--resource-share-invitation-arn invitation_arn \
--region region
```

For Windows:

```
aws ram accept-resource-share-invitation ^
--resource-share-invitation-arn invitation_arn ^
--region region
```

AWS RAM and RDS API

To accept invitations to share somebody's cluster

1. Find the invitation ARN by calling the AWS RAM API operation [GetResourceShareInvitations](#), as shown preceding.
2. Pass that ARN as the `resourceShareInvitationArn` parameter to the RDS API operation [AcceptResourceShareInvitation](#).

Cloning an Aurora cluster that is owned by another AWS account

After you accept the invitation from the AWS account that owns the DB cluster, as shown preceding, you can clone the cluster.

Console

To clone an Aurora cluster that is owned by another AWS account

1. Sign in to the AWS Management Console and open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. In the navigation pane, choose **Databases**.

At the top of the database list, you should see one or more items with a **Role** value of `Shared from account #account_id`. For security reasons, you can only see limited information about the original clusters. The properties that you can see are the ones such as database engine and version that must be the same in your cloned cluster.

3. Choose the cluster that you intend to clone.
4. For **Actions**, choose **Create clone**.
5. Follow the procedure in [Console \(p. 407\)](#) to finish setting up the cloned cluster.
6. As needed, enable encryption for the cloned cluster. If the cluster that you are cloning is encrypted, you must enable encryption for the cloned cluster. The AWS account that shared the cluster with you must also share the KMS key that was used to encrypt the cluster. You can use the same KMS key to encrypt the clone, or your own KMS key. You can't create a cross-account clone for a cluster that is encrypted with the default KMS key.

The account that owns the encryption key must grant permission to use the key to the destination account by using a key policy. This process is similar to how encrypted snapshots are shared, by using a key policy that grants permission to the destination account to use the key.

AWS CLI

To clone an Aurora cluster owned by another AWS account

1. Accept the invitation from the AWS account that owns the DB cluster, as shown preceding.
2. Clone the cluster by specifying the full ARN of the source cluster in the `source-db-cluster-identifier` parameter of the RDS CLI command `restore-db-cluster-to-point-in-time`, as shown following.

If the ARN passed as the `source-db-cluster-identifier` hasn't been shared, the same error is returned as if the specified cluster doesn't exist.

For Linux, macOS, or Unix:

```
aws rds restore-db-cluster-to-point-in-time \
--source-db-cluster-identifier=arn:aws:rds:arn_details \
--db-cluster-identifier=new_cluster_id \
--restore-type=copy-on-write \
--use-latest-restorable-time
```

For Windows:

```
aws rds restore-db-cluster-to-point-in-time ^
--source-db-cluster-identifier=arn:aws:rds:arn_details ^
--db-cluster-identifier=new_cluster_id ^
--restore-type=copy-on-write ^
--use-latest-restorable-time
```

3. If the cluster that you are cloning is encrypted, encrypt your cloned cluster by including a `kms-key-id` parameter. This `kms-key-id` value can be the same one used to encrypt the original DB cluster, or your own KMS key. Your account must have permission to use that encryption key.

For Linux, macOS, or Unix:

```
aws rds restore-db-cluster-to-point-in-time \
--source-db-cluster-identifier=arn:aws:rds:arn_details \
--db-cluster-identifier=new_cluster_id \
--restore-type=copy-on-write \
--use-latest-restorable-time \
--kms-key-id=arn:aws:kms:arn_details
```

For Windows:

```
aws rds restore-db-cluster-to-point-in-time ^
--source-db-cluster-identifier=arn:aws:rds:arn_details ^
--db-cluster-identifier=new_cluster_id ^
--restore-type=copy-on-write ^
--use-latest-restorable-time ^
--kms-key-id=arn:aws:kms:arn_details
```

The account that owns the encryption key must grant permission to use the key to the destination account by using a key policy. This process is similar to how encrypted snapshots are shared, by using a key policy that grants permission to the destination account to use the key. An example of a key policy follows.

```
{  
  "Id": "key-policy-1",  
  "Version": "2012-10-17",  
  "Statement": [
```

```
"Statement": [
    {
        "Sid": "Allow use of the key",
        "Effect": "Allow",
        "Principal": {"AWS": [
            "arn:aws:iam::account_id:user/KeyUser",
            "arn:aws:iam::account_id:root"
        ]},
        "Action": [
            "kms>CreateGrant",
            "kms:Encrypt",
            "kms:Decrypt",
            "kms:ReEncrypt",
            "kms:GenerateDataKey*",
            "kms:DescribeKey"
        ],
        "Resource": "*"
    },
    {
        "Sid": "Allow attachment of persistent resources",
        "Effect": "Allow",
        "Principal": {"AWS": [
            "arn:aws:iam::account_id:user/KeyUser",
            "arn:aws:iam::account_id:root"
        ]},
        "Action": [
            "kms>CreateGrant",
            "kms>ListGrants",
            "kms:RevokeGrant"
        ],
        "Resource": "*",
        "Condition": {"Bool": {"kms:GrantIsForAWSResource": true}}
    }
]
```

Note

The [restore-db-cluster-to-point-in-time](#) AWS CLI command restores only the DB cluster, not the DB instances for that DB cluster. To create DB instances for the restored DB cluster, invoke the [create-db-instance](#) command. Specify the identifier of the restored DB cluster in `--db-cluster-identifier`.

You can create DB instances only after the [restore-db-cluster-to-point-in-time](#) command has completed and the DB cluster is available.

RDS API

To clone an Aurora cluster owned by another AWS account

1. Accept the invitation from the AWS account that owns the DB cluster, as shown preceding.
2. Clone the cluster by specifying the full ARN of the source cluster in the `SourceDBClusterIdentifier` parameter of the RDS API operation [RestoreDBClusterToPointInTime](#).

If the ARN passed as the `SourceDBClusterIdentifier` hasn't been shared, then the same error is returned as if the specified cluster doesn't exist.

3. If the cluster that you are cloning is encrypted, include a `KmsKeyId` parameter to encrypt your cloned cluster. This `kms-key-id` value can be the same one used to encrypt the original DB cluster, or your own KMS key. Your account must have permission to use that encryption key.

When you clone a volume, the destination account must have permission to use the encryption key used to encrypt the source cluster. Aurora encrypts the new cloned cluster with the encryption key specified in `KmsKeyId`.

The account that owns the encryption key must grant permission to use the key to the destination account by using a key policy. This process is similar to how encrypted snapshots are shared, by using a key policy that grants permission to the destination account to use the key. An example of a key policy follows.

```
{
  "Id": "key-policy-1",
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "Allow use of the key",
      "Effect": "Allow",
      "Principal": {"AWS": [
        "arn:aws:iam::account_id:user/KeyUser",
        "arn:aws:iam::account_id:root"
      ]},
      "Action": [
        "kms>CreateGrant",
        "kms:Encrypt",
        "kms:Decrypt",
        "kms:ReEncrypt",
        "kms:GenerateDataKey*",
        "kms:DescribeKey"
      ],
      "Resource": "*"
    },
    {
      "Sid": "Allow attachment of persistent resources",
      "Effect": "Allow",
      "Principal": {"AWS": [
        "arn:aws:iam::account_id:user/KeyUser",
        "arn:aws:iam::account_id:root"
      ]},
      "Action": [
        "kms>CreateGrant",
        "kms>ListGrants",
        "kms:RevokeGrant"
      ],
      "Resource": "*",
      "Condition": {"Bool": {"kms:GrantIsForAWSResource": true}}
    }
  ]
}
```

Note

The [RestoreDBClusterToPointInTime](#) RDS API operation restores only the DB cluster, not the DB instances for that DB cluster. To create DB instances for the restored DB cluster, invoke the [CreateDBInstance](#) RDS API operation. Specify the identifier of the restored DB cluster in `DBClusterIdentifier`. You can create DB instances only after the `RestoreDBClusterToPointInTime` operation has completed and the DB cluster is available.

Checking if a DB cluster is a cross-account clone

The `DBClusters` object identifies whether each cluster is a cross-account clone. You can see the clusters that you have permission to clone by using the `include-shared` option when you run the RDS CLI

command `describe-db-clusters`. However, you can't see most of the configuration details for such clusters.

AWS CLI

To check if a DB cluster is a cross-account clone

- Call the RDS CLI command `describe-db-clusters`.

The following example shows how actual or potential cross-account clone DB clusters appear in `describe-db-clusters` output. For existing clusters owned by your AWS account, the `CrossAccountClone` field indicates whether the cluster is a clone of a DB cluster that is owned by another AWS account.

In some cases, an entry might have a different AWS account number than yours in the `DBClusterArn` field. In this case, that entry represents a cluster that is owned by a different AWS account and that you can clone. Such entries have few fields other than `DBClusterArn`. When creating the cloned cluster, specify the same `StorageEncrypted`, `Engine`, and `EngineVersion` values as in the original cluster.

```
$ aws rds describe-db-clusters --include-shared --region us-east-1
{
  "DBClusters": [
    {
      "EarliestRestorableTime": "2019-05-01T21:17:54.106Z",
      "Engine": "aurora",
      "EngineVersion": "5.6.10a",
      "CrossAccountClone": false,
      ...
    },
    {
      "EarliestRestorableTime": "2019-04-09T16:01:07.398Z",
      "Engine": "aurora",
      "EngineVersion": "5.6.10a",
      "CrossAccountClone": true,
      ...
    },
    {
      "StorageEncrypted": false,
      "DBClusterArn": "arn:aws:rds:us-east-1:12345678:cluster:cluster-abcdefg",
      "Engine": "aurora",
      "EngineVersion": "5.6.10a",
    }
  ]
}
```

RDS API

To check if a DB cluster is a cross-account clone

- Call the RDS API operation `DescribeDBClusters`.

For existing clusters owned by your AWS account, the `CrossAccountClone` field indicates whether the cluster is a clone of a DB cluster owned by another AWS account. Entries with a different AWS account number in the `DBClusterArn` field represent clusters that you can clone and that are owned by other AWS accounts. These entries have few fields other than `DBClusterArn`. When creating the cloned cluster, specify the same `StorageEncrypted`, `Engine`, and `EngineVersion` values as in the original cluster.

The following example shows a return value that demonstrates both actual and potential cloned clusters.

```
{  
    "DBClusters": [  
        {  
            "EarliestRestorableTime": "2019-05-01T21:17:54.106Z",  
            "Engine": "aurora",  
            "EngineVersion": "5.6.10a",  
            "CrossAccountClone": false,  
            ...  
        },  
        {  
            "EarliestRestorableTime": "2019-04-09T16:01:07.398Z",  
            "Engine": "aurora",  
            "EngineVersion": "5.6.10a",  
            "CrossAccountClone": true,  
            ...  
        },  
        {  
            "StorageEncrypted": false,  
            "DBClusterArn": "arn:aws:rds:us-east-1:12345678:cluster:cluster-abcdefgh",  
            "Engine": "aurora",  
            "EngineVersion": "5.6.10a"  
        }  
    ]  
}
```

Integrating Aurora with other AWS services

Integrate Amazon Aurora with other AWS services so that you can extend your Aurora DB cluster to use additional capabilities in the AWS Cloud.

Topics

- [Integrating AWS services with Amazon Aurora MySQL \(p. 426\)](#)
- [Integrating AWS services with Amazon Aurora PostgreSQL \(p. 426\)](#)
- [Using Amazon Aurora Auto Scaling with Aurora replicas \(p. 427\)](#)
- [Using machine learning \(ML\) capabilities with Amazon Aurora \(p. 442\)](#)

Integrating AWS services with Amazon Aurora MySQL

Amazon Aurora MySQL integrates with other AWS services so that you can extend your Aurora MySQL DB cluster to use additional capabilities in the AWS Cloud. Your Aurora MySQL DB cluster can use AWS services to do the following:

- Synchronously or asynchronously invoke an AWS Lambda function using the native functions `lambda_sync` or `lambda_async`. Or, asynchronously invoke an AWS Lambda function using the `mysql.lambda_async` procedure.
- Load data from text or XML files stored in an Amazon S3 bucket into your DB cluster using the `LOAD DATA FROM S3` or `LOAD XML FROM S3` command.
- Save data to text files stored in an Amazon S3 bucket from your DB cluster using the `SELECT INTO OUTFILE S3` command.
- Automatically add or remove Aurora Replicas with Application Auto Scaling. For more information, see [Using Amazon Aurora Auto Scaling with Aurora replicas \(p. 427\)](#).

For more information about integrating Aurora MySQL with other AWS services, see [Integrating Amazon Aurora MySQL with other AWS services \(p. 984\)](#).

Integrating AWS services with Amazon Aurora PostgreSQL

Amazon Aurora PostgreSQL integrates with other AWS services so that you can extend your Aurora PostgreSQL DB cluster to use additional capabilities in the AWS Cloud. Your Aurora PostgreSQL DB cluster can use AWS services to do the following:

- Quickly collect, view, and assess performance on your relational database workloads with Performance Insights.
- Automatically add or remove Aurora Replicas with Aurora Auto Scaling. For more information, see [Using Amazon Aurora Auto Scaling with Aurora replicas \(p. 427\)](#).

For more information about integrating Aurora PostgreSQL with other AWS services, see [Integrating Amazon Aurora PostgreSQL with other AWS services \(p. 1437\)](#).

Using Amazon Aurora Auto Scaling with Aurora replicas

To meet your connectivity and workload requirements, Aurora Auto Scaling dynamically adjusts the number of Aurora Replicas provisioned for an Aurora DB cluster using single-master replication. Aurora Auto Scaling is available for both Aurora MySQL and Aurora PostgreSQL. Aurora Auto Scaling enables your Aurora DB cluster to handle sudden increases in connectivity or workload. When the connectivity or workload decreases, Aurora Auto Scaling removes unnecessary Aurora Replicas so that you don't pay for unused provisioned DB instances.

You define and apply a scaling policy to an Aurora DB cluster. The *scaling policy* defines the minimum and maximum number of Aurora Replicas that Aurora Auto Scaling can manage. Based on the policy, Aurora Auto Scaling adjusts the number of Aurora Replicas up or down in response to actual workloads, determined by using Amazon CloudWatch metrics and target values.

You can use the AWS Management Console to apply a scaling policy based on a predefined metric. Alternatively, you can use either the AWS CLI or Aurora Auto Scaling API to apply a scaling policy based on a predefined or custom metric.

Topics

- [Before you begin \(p. 427\)](#)
- [Aurora Auto Scaling policies \(p. 428\)](#)
- [Adding a scaling policy \(p. 429\)](#)
- [Editing a scaling policy \(p. 438\)](#)
- [Deleting a scaling policy \(p. 440\)](#)
- [DB instance IDs and tagging \(p. 441\)](#)

Before you begin

Before you can use Aurora Auto Scaling with an Aurora DB cluster, you must first create an Aurora DB cluster with a primary instance and at least one Aurora Replica. Although Aurora Auto Scaling manages Aurora Replicas, the Aurora DB cluster must start with at least one Aurora Replica. For more information about creating an Aurora DB cluster, see [Creating an Amazon Aurora DB cluster \(p. 125\)](#).

Aurora Auto Scaling only scales a DB cluster if all Aurora Replicas in a DB cluster are in the available state. If any of the Aurora Replicas are in a state other than available, Aurora Auto Scaling waits until the whole DB cluster becomes available for scaling.

When Aurora Auto Scaling adds a new Aurora Replica, the new Aurora Replica is the same DB instance class as the one used by the primary instance. For more information about DB instance classes, see [Aurora DB instance classes \(p. 54\)](#). Also, the promotion tier for new Aurora Replicas is set to the last priority, which is 15 by default. This means that during a failover, a replica with a better priority, such as one created manually, would be promoted first. For more information, see [Fault tolerance for an Aurora DB cluster \(p. 69\)](#).

Aurora Auto Scaling only removes Aurora Replicas that it created.

To benefit from Aurora Auto Scaling, your applications must support connections to new Aurora Replicas. To do so, we recommend using the Aurora reader endpoint. For Aurora MySQL you can use a driver such as the MariaDB Connector/J utility. For more information, see [Connecting to an Amazon Aurora DB cluster \(p. 281\)](#).

Note

Aurora global databases currently don't support Aurora Auto Scaling for secondary DB clusters.

Aurora Auto Scaling policies

Aurora Auto Scaling uses a scaling policy to adjust the number of Aurora Replicas in an Aurora DB cluster. Aurora Auto Scaling has the following components:

- A service-linked role
- A target metric
- Minimum and maximum capacity
- A cooldown period

Service linked role

Aurora Auto Scaling uses the `AWSServiceRoleForApplicationAutoScaling_RDSCluster` service-linked role. For more information, see [Service-linked roles for Application Auto Scaling](#) in the *Application Auto Scaling User Guide*.

Target metric

In this type of policy, a predefined or custom metric and a target value for the metric is specified in a target-tracking scaling policy configuration. Aurora Auto Scaling creates and manages CloudWatch alarms that trigger the scaling policy and calculates the scaling adjustment based on the metric and target value. The scaling policy adds or removes Aurora Replicas as required to keep the metric at, or close to, the specified target value. In addition to keeping the metric close to the target value, a target-tracking scaling policy also adjusts to fluctuations in the metric due to a changing workload. Such a policy also minimizes rapid fluctuations in the number of available Aurora Replicas for your DB cluster.

For example, take a scaling policy that uses the predefined average CPU utilization metric. Such a policy can keep CPU utilization at, or close to, a specified percentage of utilization, such as 40 percent.

Note

For each Aurora DB cluster, you can create only one Auto Scaling policy for each target metric.

Minimum and maximum capacity

You can specify the maximum number of Aurora Replicas to be managed by Application Auto Scaling. This value must be set to 0–15, and must be equal to or greater than the value specified for the minimum number of Aurora Replicas.

You can also specify the minimum number of Aurora Replicas to be managed by Application Auto Scaling. This value must be set to 0–15, and must be equal to or less than the value specified for the maximum number of Aurora Replicas.

Note

The minimum and maximum capacity are set for an Aurora DB cluster. The specified values apply to all of the policies associated with that Aurora DB cluster.

Cooldown period

You can tune the responsiveness of a target-tracking scaling policy by adding cooldown periods that affect scaling your Aurora DB cluster in and out. A cooldown period blocks subsequent scale-in or scale-out requests until the period expires. These blocks slow the deletions of Aurora Replicas in your Aurora DB cluster for scale-in requests, and the creation of Aurora Replicas for scale-out requests.

You can specify the following cooldown periods:

- A scale-in activity reduces the number of Aurora Replicas in your Aurora DB cluster. A scale-in cooldown period specifies the amount of time, in seconds, after a scale-in activity completes before another scale-in activity can start.

- A scale-out activity increases the number of Aurora Replicas in your Aurora DB cluster. A scale-out cooldown period specifies the amount of time, in seconds, after a scale-out activity completes before another scale-out activity can start.

When a scale-in or a scale-out cooldown period is not specified, the default for each is 300 seconds.

Enable or disable scale-in activities

You can enable or disable scale-in activities for a policy. Enabling scale-in activities allows the scaling policy to delete Aurora Replicas. When scale-in activities are enabled, the scale-in cooldown period in the scaling policy applies to scale-in activities. Disabling scale-in activities prevents the scaling policy from deleting Aurora Replicas.

Note

Scale-out activities are always enabled so that the scaling policy can create Aurora Replicas as needed.

Adding a scaling policy

You can add a scaling policy using the AWS Management Console, the AWS CLI, or the Application Auto Scaling API.

Note

For an example that adds a scaling policy using AWS CloudFormation, see [Declaring a scaling policy for an Aurora DB cluster](#) in the *AWS CloudFormation User Guide*.

Topics

- [Adding a scaling policy using the AWS Management Console \(p. 429\)](#)
- [Adding a scaling policy using the AWS CLI or the Application Auto Scaling API \(p. 432\)](#)

Adding a scaling policy using the AWS Management Console

You can add a scaling policy to an Aurora DB cluster by using the AWS Management Console.

To add an auto scaling policy to an Aurora DB cluster

1. Sign in to the AWS Management Console and open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. In the navigation pane, choose **Databases**.
3. Choose the Aurora DB cluster that you want to add a policy for.
4. Choose the **Logs & events** tab.
5. In the **Auto scaling policies** section, choose **Add**.

The **Add Auto Scaling policy** dialog box appears.

6. For **Policy Name**, type the policy name.
7. For the target metric, choose one of the following:
 - **Average CPU utilization of Aurora Replicas** to create a policy based on the average CPU utilization.
 - **Average connections of Aurora Replicas** to create a policy based on the average number of connections to Aurora Replicas.
8. For the target value, type one of the following:

- If you chose **Average CPU utilization of Aurora Replicas** in the previous step, type the percentage of CPU utilization that you want to maintain on Aurora Replicas.
- If you chose **Average connections of Aurora Replicas** in the previous step, type the number of connections that you want to maintain.

Aurora Replicas are added or removed to keep the metric close to the specified value.

9. (Optional) Open **Additional Configuration** to create a scale-in or scale-out cooldown period.
10. For **Minimum capacity**, type the minimum number of Aurora Replicas that the Aurora Auto Scaling policy is required to maintain.
11. For **Maximum capacity**, type the maximum number of Aurora Replicas the Aurora Auto Scaling policy is required to maintain.
12. Choose **Add policy**.

The following dialog box creates an Auto Scaling policy based an average CPU utilization of 40 percent. The policy specifies a minimum of 5 Aurora Replicas and a maximum of 15 Aurora Replicas.

Add Auto Scaling policy

Define an Auto Scaling policy to automatically add or remove [Aurora Replicas](#). We recommend using the Aurora reader endpoint or the MariaDB Connector to establish connections with new Aurora Replicas. [Learn more](#).

Policy details

Policy name

A name for the policy used to identify it in the console, CLI, API, notifications, and events.

CPUScalingPolicy

Policy name must be 1 to 256 characters.

IAM role

The following service-linked role is used by Aurora Auto Scaling.

AWSServiceRoleForApplicationAutoScaling_RDSCluster

Target metric

Only one Aurora Auto Scaling policy is allowed for one metric.

- Average CPU utilization of Aurora Replicas [View metric](#)
- Average connections of Aurora Replicas [View metric](#)

Target value

Specify the desired value for the selected metric. Aurora Replicas will be added or removed to keep the metric close to the specified value.

40



%

► Additional configuration

Cluster capacity details

Configure the minimum and maximum number of Aurora Replicas you want Aurora Auto Scaling to maintain.

Minimum capacity

Specify the minimum number of Aurora Replicas to maintain.

5



Aurora Replicas

Maximum capacity

Specify the maximum number of Aurora Replicas to maintain. Up to 15 Aurora Replicas are supported.

15



Aurora Replicas

Cancel

Add policy

The following dialog box creates an auto scaling policy based an average number of connections of 100. The policy specifies a minimum of two Aurora Replicas and a maximum of eight Aurora Replicas.

Add Auto Scaling policy

Define an Auto Scaling policy to automatically add or remove [Aurora Replicas](#). We recommend using the Aurora reader endpoint or the MariaDB Connector to establish connections with new Aurora Replicas. [Learn more](#).

Policy details

Policy name

A name for the policy used to identify it in the console, CLI, API, notifications, and events.

ConnectionsScalingPolicy

Policy name must be 1 to 256 characters.

IAM role

The following service-linked role is used by Aurora Auto Scaling.

AWSServiceRoleForApplicationAutoScaling_RDSCluster

Target metric

Only one Aurora Auto Scaling policy is allowed for one metric.

- Average CPU utilization of Aurora Replicas [View metric](#)
- Average connections of Aurora Replicas [View metric](#)

Target value

Specify the desired value for the selected metric. Aurora Replicas will be added or removed to keep the metric close to the specified value.

100



connections

► Additional configuration

Cluster capacity details

Configure the minimum and maximum number of Aurora Replicas you want Aurora Auto Scaling to maintain.

Minimum capacity

Specify the minimum number of Aurora Replicas to maintain.

2



Aurora Replicas

Maximum capacity

Specify the maximum number of Aurora Replicas to maintain. Up to 15 Aurora Replicas are supported.

8



Aurora Replicas

Cancel

Add policy

Adding a scaling policy using the AWS CLI or the Application Auto Scaling API

You can apply a scaling policy based on either a predefined or custom metric. To do so, you can use the AWS CLI or the Application Auto Scaling API. The first step is to register your Aurora DB cluster with Application Auto Scaling.

Registering an Aurora DB cluster

Before you can use Aurora Auto Scaling with an Aurora DB cluster, you register your Aurora DB cluster with Application Auto Scaling. You do so to define the scaling dimension and limits to be applied to that cluster. Application Auto Scaling dynamically scales the Aurora DB cluster along the

`rds:cluster:ReadReplicaCount` scalable dimension, which represents the number of Aurora Replicas.

To register your Aurora DB cluster, you can use either the AWS CLI or the Application Auto Scaling API.

AWS CLI

To register your Aurora DB cluster, use the `register-scalable-target` AWS CLI command with the following parameters:

- `--service-namespace` – Set this value to `rds`.
- `--resource-id` – The resource identifier for the Aurora DB cluster. For this parameter, the resource type is `cluster` and the unique identifier is the name of the Aurora DB cluster, for example `cluster:mscalablecluster`.
- `--scalable-dimension` – Set this value to `rds:cluster:ReadReplicaCount`.
- `--min-capacity` – The minimum number of reader DB instances to be managed by Application Auto Scaling. For information about the relationship between `--min-capacity`, `--max-capacity`, and the number of DB instances in your cluster, see [Minimum and maximum capacity \(p. 428\)](#).
- `--max-capacity` – The maximum number of reader DB instances to be managed by Application Auto Scaling. For information about the relationship between `--min-capacity`, `--max-capacity`, and the number of DB instances in your cluster, see [Minimum and maximum capacity \(p. 428\)](#).

Example

In the following example, you register an Aurora DB cluster named `mscalablecluster`. The registration indicates that the DB cluster should be dynamically scaled to have from one to eight Aurora Replicas.

For Linux, macOS, or Unix:

```
aws application-autoscaling register-scalable-target \
--service-namespace rds \
--resource-id cluster:mscalablecluster \
--scalable-dimension rds:cluster:ReadReplicaCount \
--min-capacity 1 \
--max-capacity 8
```

For Windows:

```
aws application-autoscaling register-scalable-target ^
--service-namespace rds ^
--resource-id cluster:mscalablecluster ^
--scalable-dimension rds:cluster:ReadReplicaCount ^
--min-capacity 1 ^
--max-capacity 8 ^
```

RDS API

To register your Aurora DB cluster with Application Auto Scaling, use the `RegisterScalableTarget` Application Auto Scaling API operation with the following parameters:

- `ServiceNamespace` – Set this value to `rds`.
- `ResourceID` – The resource identifier for the Aurora DB cluster. For this parameter, the resource type is `cluster` and the unique identifier is the name of the Aurora DB cluster, for example `cluster:mscalablecluster`.

- **ScalableDimension** – Set this value to `rds:cluster:ReadReplicaCount`.
- **MinCapacity** – The minimum number of reader DB instances to be managed by Application Auto Scaling. For information about the relationship between `MinCapacity`, `MaxCapacity`, and the number of DB instances in your cluster, see [Minimum and maximum capacity \(p. 428\)](#).
- **MaxCapacity** – The maximum number of reader DB instances to be managed by Application Auto Scaling. For information about the relationship between `MinCapacity`, `MaxCapacity`, and the number of DB instances in your cluster, see [Minimum and maximum capacity \(p. 428\)](#).

Example

In the following example, you register an Aurora DB cluster named `mscalablecluster` with the Application Auto Scaling API. This registration indicates that the DB cluster should be dynamically scaled to have from one to eight Aurora Replicas.

```
POST / HTTP/1.1
Host: autoscaling.us-east-2.amazonaws.com
Accept-Encoding: identity
Content-Length: 219
X-Amz-Target: AnyScaleFrontendService.RegisterScalableTarget
X-Amz-Date: 20160506T182145Z
User-Agent: aws-cli/1.10.23 Python/2.7.11 Darwin/15.4.0 botocore/1.4.8
Content-Type: application/x-amz-json-1.1
Authorization: AUTHPARAMS

{
    "ServiceNamespace": "rds",
    "ResourceId": "cluster:mscalablecluster",
    "ScalableDimension": "rds:cluster:ReadReplicaCount",
    "MinCapacity": 1,
    "MaxCapacity": 8
}
```

Defining a scaling policy for an Aurora DB cluster

A target-tracking scaling policy configuration is represented by a JSON block that the metrics and target values are defined in. You can save a scaling policy configuration as a JSON block in a text file. You use that text file when invoking the AWS CLI or the Application Auto Scaling API. For more information about policy configuration syntax, see [TargetTrackingScalingPolicyConfiguration](#) in the *Application Auto Scaling API Reference*.

The following options are available for defining a target-tracking scaling policy configuration.

Topics

- [Using a predefined metric \(p. 434\)](#)
- [Using a custom metric \(p. 435\)](#)
- [Using cooldown periods \(p. 435\)](#)
- [Disabling scale-in activity \(p. 436\)](#)

Using a predefined metric

By using predefined metrics, you can quickly define a target-tracking scaling policy for an Aurora DB cluster that works well with both target tracking and dynamic scaling in Aurora Auto Scaling.

Currently, Aurora supports the following predefined metrics in Aurora Auto Scaling:

- **RDSReaderAverageCPUUtilization** – The average value of the `CPUUtilization` metric in CloudWatch across all Aurora Replicas in the Aurora DB cluster.

- **RDSReaderAverageDatabaseConnections** – The average value of the DatabaseConnections metric in CloudWatch across all Aurora Replicas in the Aurora DB cluster.

For more information about the CPUUtilization and DatabaseConnections metrics, see [Amazon CloudWatch metrics for Amazon Aurora \(p. 633\)](#).

To use a predefined metric in your scaling policy, you create a target tracking configuration for your scaling policy. This configuration must include a PredefinedMetricSpecification for the predefined metric and a TargetValue for the target value of that metric.

Example

The following example describes a typical policy configuration for target-tracking scaling for an Aurora DB cluster. In this configuration, the RDSReaderAverageCPUUtilization predefined metric is used to adjust the Aurora DB cluster based on an average CPU utilization of 40 percent across all Aurora Replicas.

```
{
    "TargetValue": 40.0,
    "PredefinedMetricSpecification":
    {
        "PredefinedMetricType": "RDSReaderAverageCPUUtilization"
    }
}
```

Using a custom metric

By using custom metrics, you can define a target-tracking scaling policy that meets your custom requirements. You can define a custom metric based on any Aurora metric that changes in proportion to scaling.

Not all Aurora metrics work for target tracking. The metric must be a valid utilization metric and describe how busy an instance is. The value of the metric must increase or decrease in proportion to the number of Aurora Replicas in the Aurora DB cluster. This proportional increase or decrease is necessary to use the metric data to proportionally scale out or in the number of Aurora Replicas.

Example

The following example describes a target-tracking configuration for a scaling policy. In this configuration, a custom metric adjusts an Aurora DB cluster based on an average CPU utilization of 50 percent across all Aurora Replicas in an Aurora DB cluster named `my-db-cluster`.

```
{
    "TargetValue": 50,
    "CustomizedMetricSpecification":
    {
        "MetricName": "CPUUtilization",
        "Namespace": "AWS/RDS",
        "Dimensions": [
            {"Name": "DBClusterIdentifier", "Value": "my-db-cluster"},
            {"Name": "Role", "Value": "READER"}
        ],
        "Statistic": "Average",
        "Unit": "Percent"
    }
}
```

Using cooldown periods

You can specify a value, in seconds, for `ScaleOutCooldown` to add a cooldown period for scaling out your Aurora DB cluster. Similarly, you can add a value, in seconds, for `ScaleInCooldown` to add a

cooldown period for scaling in your Aurora DB cluster. For more information about `ScaleInCooldown` and `ScaleOutCooldown`, see [TargetTrackingScalingPolicyConfiguration](#) in the *Application Auto Scaling API Reference*.

Example

The following example describes a target-tracking configuration for a scaling policy. In this configuration, the `RDSReaderAverageCPUUtilization` predefined metric is used to adjust an Aurora DB cluster based on an average CPU utilization of 40 percent across all Aurora Replicas in that Aurora DB cluster. The configuration provides a scale-in cooldown period of 10 minutes and a scale-out cooldown period of 5 minutes.

```
{  
    "TargetValue": 40.0,  
    "PredefinedMetricSpecification":  
    {  
        "PredefinedMetricType": "RDSReaderAverageCPUUtilization"  
    },  
    "ScaleInCooldown": 600,  
    "ScaleOutCooldown": 300  
}
```

Disabling scale-in activity

You can prevent the target-tracking scaling policy configuration from scaling in your Aurora DB cluster by disabling scale-in activity. Disabling scale-in activity prevents the scaling policy from deleting Aurora Replicas, while still allowing the scaling policy to create them as needed.

You can specify a Boolean value for `DisableScaleIn` to enable or disable scale in activity for your Aurora DB cluster. For more information about `DisableScaleIn`, see [TargetTrackingScalingPolicyConfiguration](#) in the *Application Auto Scaling API Reference*.

Example

The following example describes a target-tracking configuration for a scaling policy. In this configuration, the `RDSReaderAverageCPUUtilization` predefined metric adjusts an Aurora DB cluster based on an average CPU utilization of 40 percent across all Aurora Replicas in that Aurora DB cluster. The configuration disables scale-in activity for the scaling policy.

```
{  
    "TargetValue": 40.0,  
    "PredefinedMetricSpecification":  
    {  
        "PredefinedMetricType": "RDSReaderAverageCPUUtilization"  
    },  
    "DisableScaleIn": true  
}
```

Applying a scaling policy to an Aurora DB cluster

After registering your Aurora DB cluster with Application Auto Scaling and defining a scaling policy, you apply the scaling policy to the registered Aurora DB cluster. To apply a scaling policy to an Aurora DB cluster, you can use the AWS CLI or the Application Auto Scaling API.

AWS CLI

To apply a scaling policy to your Aurora DB cluster, use the `put-scaling-policy` AWS CLI command with the following parameters:

- `--policy-name` – The name of the scaling policy.
- `--policy-type` – Set this value to `TargetTrackingScaling`.

- **--resource-id** – The resource identifier for the Aurora DB cluster. For this parameter, the resource type is `cluster` and the unique identifier is the name of the Aurora DB cluster, for example `cluster:myscalablecluster`.
- **--service-namespace** – Set this value to `rds`.
- **--scalable-dimension** – Set this value to `rds:cluster:ReadReplicaCount`.
- **--target-tracking-scaling-policy-configuration** – The target-tracking scaling policy configuration to use for the Aurora DB cluster.

Example

In the following example, you apply a target-tracking scaling policy named `myscalablepolicy` to an Aurora DB cluster named `myscalablecluster` with Application Auto Scaling. To do so, you use a policy configuration saved in a file named `config.json`.

For Linux, macOS, or Unix:

```
aws application-autoscaling put-scaling-policy \
  --policy-name myscalablepolicy \
  --policy-type TargetTrackingScaling \
  --resource-id cluster:myscalablecluster \
  --service-namespace rds \
  --scalable-dimension rds:cluster:ReadReplicaCount \
  --target-tracking-scaling-policy-configuration file://config.json
```

For Windows:

```
aws application-autoscaling put-scaling-policy ^
  --policy-name myscalablepolicy ^
  --policy-type TargetTrackingScaling ^
  --resource-id cluster:myscalablecluster ^
  --service-namespace rds ^
  --scalable-dimension rds:cluster:ReadReplicaCount ^
  --target-tracking-scaling-policy-configuration file://config.json
```

RDS API

To apply a scaling policy to your Aurora DB cluster with the Application Auto Scaling API, use the [PutScalingPolicy](#) Application Auto Scaling API operation with the following parameters:

- **PolicyName** – The name of the scaling policy.
- **ServiceNamespace** – Set this value to `rds`.
- **ResourceID** – The resource identifier for the Aurora DB cluster. For this parameter, the resource type is `cluster` and the unique identifier is the name of the Aurora DB cluster, for example `cluster:myscalablecluster`.
- **ScalableDimension** – Set this value to `rds:cluster:ReadReplicaCount`.
- **PolicyType** – Set this value to `TargetTrackingScaling`.
- **TargetTrackingScalingPolicyConfiguration** – The target-tracking scaling policy configuration to use for the Aurora DB cluster.

Example

In the following example, you apply a target-tracking scaling policy named `myscalablepolicy` to an Aurora DB cluster named `myscalablecluster` with Application Auto Scaling. You use a policy configuration based on the `RDSReaderAverageCPUUtilization` predefined metric.

```
POST / HTTP/1.1
Host: autoscaling.us-east-2.amazonaws.com
Accept-Encoding: identity
Content-Length: 219
X-Amz-Target: AnyScaleFrontendService.PutScalingPolicy
X-Amz-Date: 20160506T182145Z
User-Agent: aws-cli/1.10.23 Python/2.7.11 Darwin/15.4.0 botocore/1.4.8
Content-Type: application/x-amz-json-1.1
Authorization: AUTHPARAMS

{
    "PolicyName": "myscalablepolicy",
    "ServiceNamespace": "rds",
    "ResourceId": "cluster:myscalablecluster",
    "ScalableDimension": "rds:cluster:ReadReplicaCount",
    "PolicyType": "TargetTrackingScaling",
    "TargetTrackingScalingPolicyConfiguration": {
        "TargetValue": 40.0,
        "PredefinedMetricSpecification":
        {
            "PredefinedMetricType": "RDSReaderAverageCPUUtilization"
        }
    }
}
```

Editing a scaling policy

You can edit a scaling policy using the AWS Management Console, the AWS CLI, or the Application Auto Scaling API.

Editing a scaling policy using the AWS Management Console

You can edit a scaling policy by using the AWS Management Console.

To edit an auto scaling policy for an Aurora DB cluster

1. Sign in to the AWS Management Console and open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. In the navigation pane, choose **Databases**.
3. Choose the Aurora DB cluster whose auto scaling policy you want to edit.
4. Choose the **Logs & events** tab.
5. In the **Auto scaling policies** section, choose the auto scaling policy, and then choose **Edit**.
6. Make changes to the policy.
7. Choose **Save**.

The following is a sample **Edit Auto Scaling policy** dialog box.

Edit Auto Scaling policy

Define an Auto Scaling policy to automatically add or remove [Aurora Replicas](#). We recommend using the Aurora reader endpoint or the MariaDB Connector to establish connections with new Aurora Replicas. [Learn more](#).

Policy details

Policy name

A name for the policy used to identify it in the console, CLI, API, notifications, and events.

CPUScalingPolicy

Policy name must be 1 to 256 characters.

IAM role

The following service-linked role is used by Aurora Auto Scaling.

AWSServiceRoleForApplicationAutoScaling_RDSCluster

Target metric

Only one Aurora Auto Scaling policy is allowed for one metric.

Average CPU utilization of Aurora Replicas [View metric](#)

Average connections of Aurora Replicas [View metric](#)

Target value

Specify the desired value for the selected metric. Aurora Replicas will be added or removed to keep the metric close to the specified value.

50



%

► Additional configuration

Cluster capacity details

Capacity values specified below apply to all the Aurora Auto Scaling policies for the DB cluster.

Minimum capacity

Specify the minimum number of Aurora Replicas to maintain.

1



Aurora Replicas

Maximum capacity

Specify the maximum number of Aurora Replicas to maintain. Up to 15 Aurora Replicas are supported.

6



Aurora Replicas

Changes to the capacity values will be applied to all the Auto Scaling policies for this DB cluster.

Cancel

Save

Editing a scaling policy using the AWS CLI or the Application Auto Scaling API

You can use the AWS CLI or the Application Auto Scaling API to edit a scaling policy in the same way that you apply a scaling policy:

- When using the AWS CLI, specify the name of the policy you want to edit in the `--policy-name` parameter. Specify new values for the parameters you want to change.
- When using the Application Auto Scaling API, specify the name of the policy you want to edit in the `PolicyName` parameter. Specify new values for the parameters you want to change.

For more information, see [Applying a scaling policy to an Aurora DB cluster \(p. 436\)](#).

Deleting a scaling policy

You can delete a scaling policy using the AWS Management Console, the AWS CLI, or the Application Auto Scaling API.

Deleting a scaling policy using the AWS Management Console

You can delete a scaling policy by using the AWS Management Console.

To delete an auto scaling policy for an Aurora DB cluster

1. Sign in to the AWS Management Console and open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. In the navigation pane, choose **Databases**.
3. Choose the Aurora DB cluster whose auto scaling policy you want to delete.
4. Choose the **Logs & events** tab.
5. In the **Auto scaling policies** section, choose the auto scaling policy, and then choose **Delete**.

Deleting a scaling policy using the AWS CLI or the Application Auto Scaling API

You can use the AWS CLI or the Application Auto Scaling API to delete a scaling policy from an Aurora DB cluster.

AWS CLI

To delete a scaling policy from your Aurora DB cluster, use the `delete-scaling-policy` AWS CLI command with the following parameters:

- `--policy-name` – The name of the scaling policy.
- `--resource-id` – The resource identifier for the Aurora DB cluster. For this parameter, the resource type is `cluster` and the unique identifier is the name of the Aurora DB cluster, for example `cluster:myscalablecluster`.
- `--service-namespace` – Set this value to `rds`.
- `--scalable-dimension` – Set this value to `rds:cluster:ReadReplicaCount`.

Example

In the following example, you delete a target-tracking scaling policy named `myscalablepolicy` from an Aurora DB cluster named `myscalablecluster`.

For Linux, macOS, or Unix:

```
aws application-autoscaling delete-scaling-policy \
--policy-name myscalablepolicy \
--resource-id cluster:myscalablecluster \
--service-namespace rds \
```

```
--scalable-dimension rds:cluster:ReadReplicaCount \
```

For Windows:

```
aws application-autoscaling delete-scaling-policy ^
--policy-name myscalablepolicy ^
--resource-id cluster:myscalablecluster ^
--service-namespace rds ^
--scalable-dimension rds:cluster:ReadReplicaCount ^
```

RDS API

To delete a scaling policy from your Aurora DB cluster, use the [DeleteScalingPolicy](#) the Application Auto Scaling API operation with the following parameters:

- **PolicyName** – The name of the scaling policy.
- **ServiceNamespace** – Set this value to `rds`.
- **ResourceId** – The resource identifier for the Aurora DB cluster. For this parameter, the resource type is `cluster` and the unique identifier is the name of the Aurora DB cluster, for example `cluster:myscalablecluster`.
- **ScalableDimension** – Set this value to `rds:cluster:ReadReplicaCount`.

Example

In the following example, you delete a target-tracking scaling policy named `myscalablepolicy` from an Aurora DB cluster named `myscalablecluster` with the Application Auto Scaling API.

```
POST / HTTP/1.1
Host: autoscaling.us-east-2.amazonaws.com
Accept-Encoding: identity
Content-Length: 219
X-Amz-Target: AnyScaleFrontendService.DeleteScalingPolicy
X-Amz-Date: 20160506T182145Z
User-Agent: aws-cli/1.10.23 Python/2.7.11 Darwin/15.4.0 botocore/1.4.8
Content-Type: application/x-amz-json-1.1
Authorization: AUTHPARAMS

{
    "PolicyName": "myscalablepolicy",
    "ServiceNamespace": "rds",
    "ResourceId": "cluster:myscalablecluster",
    "ScalableDimension": "rds:cluster:ReadReplicaCount"
}
```

DB instance IDs and tagging

When a replica is added by Aurora Auto Scaling, its DB instance ID is prefixed by `application-autoscaling-`, for example, `application-autoscaling-61aabbc-4e2f-4c65-b620-ab7421abc123`.

The following tag is automatically added to the DB instance. You can view it on the **Tags** tab of the DB instance detail page.

Tag	Value
application-autoscaling:resourceId	cluster:mynewcluster-cluster

For more information on Amazon RDS resource tags, see [Tagging Amazon RDS resources \(p. 474\)](#).

Using machine learning (ML) capabilities with Amazon Aurora

Following, you can find a description of how to use machine learning (ML) capabilities in your Aurora database applications. This feature simplifies developing database applications that use the Amazon SageMaker and Amazon Comprehend services to perform predictions. In ML terminology, these predictions are known as *inferences*.

This feature is suitable for many kinds of quick predictions. Examples include low-latency, real-time use cases such as fraud detection, ad targeting, and product recommendations. The queries pass customer profile, shopping history, and product catalog data to an SageMaker model. Then your application gets product recommendations returned as query results.

To use this feature, it helps for your organization to already have the appropriate ML models, notebooks, and so on available in the Amazon machine learning services. You can divide the database knowledge and ML knowledge among the members of your team. The database developers can focus on the SQL and database side of your application. The Aurora Machine Learning feature enables the application to use ML processing through the familiar database interface of stored function calls.

Topics

- [Using machine learning \(ML\) with Aurora MySQL \(p. 1020\)](#)
- [Using machine learning \(ML\) with Aurora PostgreSQL \(p. 1494\)](#)

Maintaining an Amazon Aurora DB cluster

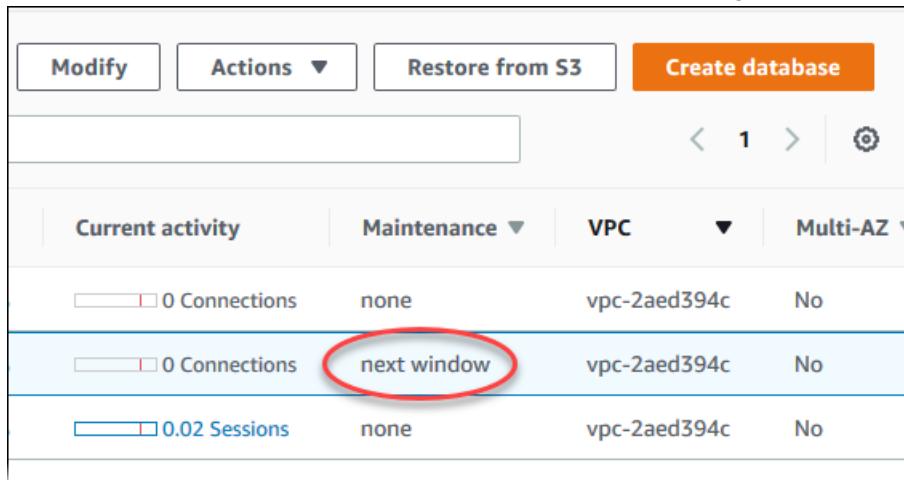
Periodically, Amazon RDS performs maintenance on Amazon RDS resources. Maintenance most often involves updates to the DB cluster's underlying hardware, underlying operating system (OS), or database engine version. Updates to the operating system most often occur for security issues and should be done as soon as possible.

Some maintenance items require that Amazon RDS take your DB cluster offline for a short time. Maintenance items that require a resource to be offline include required operating system or database patching. Required patching is automatically scheduled only for patches that are related to security and instance reliability. Such patching occurs infrequently (typically once every few months) and seldom requires more than a fraction of your maintenance window.

Deferred DB cluster and instance modifications that you have chosen not to apply immediately are also applied during the maintenance window. For example, you might choose to change DB instance classes or cluster or DB parameter groups during the maintenance window. Such modifications that you specify using the **pending reboot** setting don't show up in the **Pending maintenance** list. For information about modifying a DB cluster, see [Modifying an Amazon Aurora DB cluster \(p. 372\)](#).

Viewing pending maintenance

You can view whether a maintenance update is available for your DB cluster by using the RDS console, the AWS CLI, or the Amazon RDS API. If an update is available, it is indicated in the **Maintenance** column for the DB cluster on the Amazon RDS console, as shown following.



The screenshot shows a table in the Amazon RDS console. The columns are: Current activity, Maintenance, VPC, and Multi-AZ. There are three rows of data:

Current activity	Maintenance	VPC	Multi-AZ
0 Connections	none	vpc-2aed394c	No
0 Connections	next window	vpc-2aed394c	No
0.02 Sessions	none	vpc-2aed394c	No

If no maintenance update is available for a DB cluster, the column value is **none** for it.

If a maintenance update is available for a DB cluster, the following column values are possible:

- **required** – The maintenance action will be applied to the resource and can't be deferred indefinitely.
- **available** – The maintenance action is available, but it will not be applied to the resource automatically. You can apply it manually.
- **next window** – The maintenance action will be applied to the resource during the next maintenance window.
- **In progress** – The maintenance action is in the process of being applied to the resource.

If an update is available, you can take one of the actions:

- If the maintenance value is **next window**, defer the maintenance items by choosing **Defer upgrade** from **Actions**. You can't defer a maintenance action if it has already started.
- Apply the maintenance items immediately.
- Schedule the maintenance items to start during your next maintenance window.
- Take no action.

Note

Certain OS updates are marked as **required**. If you defer a required update, you get a notice from Amazon RDS indicating when the update will be performed. Other updates are marked as **available**, and these you can defer indefinitely.

To take an action, choose the DB cluster to show its details, then choose **Maintenance & backups**. The pending maintenance items appear.

The screenshot shows the AWS RDS console with the 'Maintenance & backups' tab selected. In the 'Maintenance' section, it shows that 'Auto minor version upgrade' is 'Enabled'. Below this, under 'Pending maintenance (1)', there is a table with one item:

Description	Type	Status	Apply date
Automatic minor version upgrade to postgres 9.6.11	db-upgrade	next window	February 25th 2019, 3:28:00 am UTC-8 (local)

There are buttons to 'Apply now' or 'Apply at next maintenance window'.

The maintenance window determines when pending operations start, but doesn't limit the total run time of these operations. Maintenance operations aren't guaranteed to finish before the maintenance window ends, and can continue beyond the specified end time. For more information, see [The Amazon RDS maintenance window \(p. 446\)](#).

For information about updates to Amazon Aurora engines and instructions for upgrading and patching them, see [Database engine updates for Amazon Aurora MySQL \(p. 1082\)](#) and [Amazon Aurora PostgreSQL updates \(p. 1597\)](#).

You can also view whether a maintenance update is available for your DB cluster by running the `describe-pending-maintenance-actions` AWS CLI command.

Applying updates for a DB cluster

With Amazon RDS, you can choose when to apply maintenance operations. You can decide when Amazon RDS applies updates by using the RDS console, AWS Command Line Interface (AWS CLI), or RDS API.

Console

To manage an update for a DB cluster

1. Sign in to the AWS Management Console and open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. In the navigation pane, choose **Databases**.
3. Choose the DB cluster that has a required update.
4. For **Actions**, choose one of the following:
 - **Upgrade now**
 - **Upgrade at next window**

Note

If you choose **Upgrade at next window** and later want to delay the update, you can choose **Defer upgrade**. You can't defer a maintenance action if it has already started. To cancel a maintenance action, modify the DB instance and disable **Auto minor version upgrade**.

AWS CLI

To apply a pending update to a DB cluster, use the [apply-pending-maintenance-action](#) AWS CLI command.

Example

For Linux, macOS, or Unix:

```
aws rds apply-pending-maintenance-action \
    --resource-identifier arn:aws:rds:us-west-2:001234567890:db:mysql-db \
    --apply-action system-update \
    --opt-in-type immediate
```

For Windows:

```
aws rds apply-pending-maintenance-action ^
    --resource-identifier arn:aws:rds:us-west-2:001234567890:db:mysql-db ^
    --apply-action system-update ^
    --opt-in-type immediate
```

Note

To defer a maintenance action, specify `undo-opt-in` for `--opt-in-type`. You can't specify `undo-opt-in` for `--opt-in-type` if the maintenance action has already started. To cancel a maintenance action, run the [modify-db-instance](#) AWS CLI command and specify `--no-auto-minor-version-upgrade`.

To return a list of resources that have at least one pending update, use the [describe-pending-maintenance-actions](#) AWS CLI command.

Example

For Linux, macOS, or Unix:

```
aws rds describe-pending-maintenance-actions \
--resource-identifier arn:aws:rds:us-west-2:001234567890:db:mysql-db
```

For Windows:

```
aws rds describe-pending-maintenance-actions ^
--resource-identifier arn:aws:rds:us-west-2:001234567890:db:mysql-db
```

You can also return a list of resources for a DB cluster by specifying the `--filters` parameter of the `describe-pending-maintenance-actions` AWS CLI command. The format for the `--filters` command is `Name=filter-name,Value=resource-id,....`

The following are the accepted values for the `Name` parameter of a filter:

- `db-instance-id` – Accepts a list of DB instance identifiers or Amazon Resource Names (ARNs). The returned list only includes pending maintenance actions for the DB instances identified by these identifiers or ARNs.
- `db-cluster-id` – Accepts a list of DB cluster identifiers or ARNs for Amazon Aurora. The returned list only includes pending maintenance actions for the DB clusters identified by these identifiers or ARNs.

For example, the following example returns the pending maintenance actions for the `sample-cluster1` and `sample-cluster2` DB clusters.

Example

For Linux, macOS, or Unix:

```
aws rds describe-pending-maintenance-actions \
--filters Name=db-cluster-id,Values=sample-cluster1,sample-cluster2
```

For Windows:

```
aws rds describe-pending-maintenance-actions ^
--filters Name=db-cluster-id,Values=sample-cluster1,sample-cluster2
```

RDS API

To apply an update to a DB cluster, call the Amazon RDS API [ApplyPendingMaintenanceAction](#) operation.

To return a list of resources that have at least one pending update, call the Amazon RDS API [DescribePendingMaintenanceActions](#) operation.

The Amazon RDS maintenance window

Every DB cluster has a weekly maintenance window during which any system changes are applied. You can think of the maintenance window as an opportunity to control when modifications and software patching occur, in the event either are requested or required. If a maintenance event is scheduled for a given week, it is initiated during the 30-minute maintenance window you identify. Most maintenance events also complete during the 30-minute maintenance window, although larger maintenance events may take more than 30 minutes to complete.

The 30-minute maintenance window is selected at random from an 8-hour block of time per region. If you don't specify a preferred maintenance window when you create the DB cluster, then Amazon RDS assigns a 30-minute maintenance window on a randomly selected day of the week.

RDS will consume some of the resources on your DB cluster while maintenance is being applied. You might observe a minimal effect on performance. For a DB instance, on rare occasions, a Multi-AZ failover might be required for a maintenance update to complete.

Following, you can find the time blocks for each region from which default maintenance windows are assigned.

Region Name	Region	Time Block
US East (Ohio)	us-east-2	03:00–11:00 UTC
US East (N. Virginia)	us-east-1	03:00–11:00 UTC
US West (N. California)	us-west-1	06:00–14:00 UTC
US West (Oregon)	us-west-2	06:00–14:00 UTC
Africa (Cape Town)	af-south-1	03:00–11:00 UTC
Asia Pacific (Hong Kong)	ap-east-1	06:00–14:00 UTC
Asia Pacific (Jakarta)	ap-southeast-3	08:00–16:00 UTC
Asia Pacific (Mumbai)	ap-south-1	06:00–14:00 UTC
Asia Pacific (Osaka)	ap-northeast-3	22:00–23:59 UTC
Asia Pacific (Seoul)	ap-northeast-2	13:00–21:00 UTC
Asia Pacific (Singapore)	ap-southeast-1	14:00–22:00 UTC
Asia Pacific (Sydney)	ap-southeast-2	12:00–20:00 UTC
Asia Pacific (Tokyo)	ap-northeast-1	13:00–21:00 UTC
Canada (Central)	ca-central-1	03:00–11:00 UTC
China (Beijing)	cn-north-1	06:00–14:00 UTC
China (Ningxia)	cn-northwest-1	06:00–14:00 UTC
Europe (Frankfurt)	eu-central-1	21:00–05:00 UTC
Europe (Ireland)	eu-west-1	22:00–06:00 UTC
Europe (London)	eu-west-2	22:00–06:00 UTC
Europe (Paris)	eu-west-3	23:59–07:29 UTC
Europe (Milan)	eu-south-1	02:00–10:00 UTC
Europe (Stockholm)	eu-north-1	23:00–07:00 UTC
Middle East (Bahrain)	me-south-1	06:00–14:00 UTC
South America (São Paulo)	sa-east-1	00:00–08:00 UTC

Region Name	Region	Time Block
AWS GovCloud (US-East)	us-gov-east-1	17:00–01:00 UTC
AWS GovCloud (US-West)	us-gov-west-1	06:00–14:00 UTC

Adjusting the preferred DB cluster maintenance window

The Aurora DB cluster maintenance window should fall at the time of lowest usage and thus might need modification from time to time. Your DB cluster is unavailable during this time only if the updates that are being applied require an outage. The outage is for the minimum amount of time required to make the necessary updates.

Console

To adjust the preferred DB cluster maintenance window

1. Sign in to the AWS Management Console and open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. In the navigation pane, choose **Databases**.
3. Choose the DB cluster for which you want to change the maintenance window.
4. Choose **Modify**.
5. In the **Maintenance** section, update the maintenance window.
6. Choose **Continue**.

On the confirmation page, review your changes.

7. To apply the changes to the maintenance window immediately, choose **Immediately** in the **Schedule of modifications** section.
8. Choose **Modify cluster** to save your changes.

Alternatively, choose **Back** to edit your changes, or choose **Cancel** to cancel your changes.

AWS CLI

To adjust the preferred DB cluster maintenance window, use the AWS CLI `modify-db-cluster` command with the following parameters:

- `--db-cluster-identifier`
- `--preferred-maintenance-window`

Example

The following code example sets the maintenance window to Tuesdays from 4:00–4:30 AM UTC.

For Linux, macOS, or Unix:

```
aws rds modify-db-cluster \
--db-cluster-identifier my-cluster \
```

```
--preferred-maintenance-window Tue:04:00-Tue:04:30
```

For Windows:

```
aws rds modify-db-cluster ^
--db-cluster-identifier my-cluster ^
--preferred-maintenance-window Tue:04:00-Tue:04:30
```

RDS API

To adjust the preferred DB cluster maintenance window, use the Amazon RDS [ModifyDBCluster](#) API operation with the following parameters:

- `DBClusterIdentifier`
- `PreferredMaintenanceWindow`

Automatic minor version upgrades for Aurora DB clusters

The **Auto minor version upgrade** setting specifies whether Aurora automatically applies upgrades to your cluster. These upgrades include patch levels containing bug fixes, and new minor versions containing additional features. They don't include any incompatible changes.

Note

This setting is enabled by default. For each new cluster, choose the appropriate value for this setting based on its importance, expected lifetime, and the amount of verification testing that you do after each upgrade.

For instructions about turning this setting on or off, see [Settings for Amazon Aurora \(p. 375\)](#). In particular, make sure to apply the same setting to all DB instances in the cluster. If any DB instance in your cluster has this setting turned off, the cluster isn't automatically upgraded.

For more information about engine updates for Aurora PostgreSQL, see [Amazon Aurora PostgreSQL updates \(p. 1597\)](#).

For more information about the **Auto minor version upgrade** setting for Aurora MySQL, see [Enabling automatic upgrades between minor Aurora MySQL versions \(p. 1089\)](#). For general information about engine updates for Aurora MySQL, see [Database engine updates for Amazon Aurora MySQL \(p. 1082\)](#).

Choosing the frequency of Aurora MySQL maintenance updates

You can control whether Aurora MySQL upgrades happen frequently or rarely for each DB cluster. The best choice depends on your usage of Aurora MySQL and the priorities for your applications that run on Aurora. For information about the Aurora MySQL long-term stability (LTS) releases that require less frequent upgrades, see [Aurora MySQL long-term support \(LTS\) releases \(p. 1085\)](#).

You might choose to upgrade an Aurora MySQL cluster rarely if some or all of the following conditions apply:

- Your testing cycle for your application takes a long time for each update to the Aurora MySQL database engine.
- You have many DB clusters or many applications all running on the same Aurora MySQL version. You prefer to upgrade all of your DB clusters and associated applications at the same time.

- You use both Aurora MySQL and RDS for MySQL, and you prefer to keep the Aurora MySQL clusters and RDS for MySQL DB instances compatible with the same level of MySQL.
- Your Aurora MySQL application is in production or is otherwise business-critical. You can't afford downtime for upgrades outside of rare occurrences for critical patches.
- Your Aurora MySQL application isn't limited by performance issues or feature gaps that are addressed in subsequent Aurora MySQL versions.

If the preceding factors apply to your situation, you can limit the number of forced upgrades for an Aurora MySQL DB cluster. You do so by choosing a specific Aurora MySQL version known as the "Long-Term Support" (LTS) version when you create or upgrade that DB cluster. Doing so minimizes the number of upgrade cycles, testing cycles, and upgrade-related outages for that DB cluster.

You might choose to upgrade an Aurora MySQL cluster frequently if some or all of the following conditions apply:

- The testing cycle for your application is straightforward and brief.
- Your application is still in the development stage.
- Your database environment uses a variety of Aurora MySQL versions, or Aurora MySQL and RDS for MySQL versions. Each Aurora MySQL cluster has its own upgrade cycle.
- You are waiting for specific performance or feature improvements before you increase your usage of Aurora MySQL.

If the preceding factors apply to your situation, you can enable Aurora to apply important upgrades more frequently by upgrading an Aurora MySQL DB cluster to a more recent Aurora MySQL version than the LTS version. Doing so makes the latest performance enhancements, bug fixes, and features available to you more quickly.

Rebooting an Amazon Aurora DB cluster or Amazon Aurora DB instance

You might need to reboot your DB cluster or some instances within the cluster, usually for maintenance reasons. For example, suppose that you modify the parameters within a parameter group or associate a different parameter group with your cluster. In these cases, you must reboot the cluster for the changes to take effect. Similarly, you might reboot one or more reader DB instances within the cluster. You can arrange the reboot operations for individual instances to minimize downtime for the entire cluster.

The time required to reboot each DB instance in your cluster depends on the database activity at the time of reboot. It also depends on the recovery process of your specific DB engine. If it's practical, reduce database activity on that particular instance before starting the reboot process. Doing so can reduce the time needed to restart the database.

You can only reboot each DB instance in your cluster when it's in the available state. A DB instance can be unavailable for several reasons. These include the cluster being stopped state, a modification being applied to the instance, and a maintenance-window action such as a version upgrade.

Rebooting a DB instance restarts the database engine process. Rebooting a DB instance results in a momentary outage, during which the DB instance status is set to *rebooting*.

Note

If a DB instance isn't using the latest changes to its associated DB parameter group, the AWS Management Console shows the DB parameter group with a status of **pending-reboot**. The **pending-reboot** parameter groups status doesn't result in an automatic reboot during the next maintenance window. To apply the latest parameter changes to that DB instance, manually reboot the DB instance. For more information about parameter groups, see [Working with DB parameter groups and DB cluster parameter groups \(p. 339\)](#).

Topics

- [Rebooting a DB instance within an Aurora cluster \(p. 451\)](#)
- [Rebooting an Aurora cluster \(Aurora PostgreSQL and Aurora MySQL before version 2.10\) \(p. 452\)](#)
- [Rebooting an Aurora MySQL cluster \(version 2.10 and higher\) \(p. 452\)](#)
- [Checking uptime for Aurora clusters and instances \(p. 453\)](#)
- [Examples of Aurora reboot operations \(p. 455\)](#)

Rebooting a DB instance within an Aurora cluster

This procedure is the most important operation that you take when performing reboots with Aurora. Many of the maintenance procedures involve rebooting one or more Aurora DB instances in a particular order.

Console

To reboot a DB instance

1. Sign in to the AWS Management Console and open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. In the navigation pane, choose **Databases**, and then choose the DB instance that you want to reboot.
3. For **Actions**, choose **Reboot**.
The **Reboot DB Instance** page appears.
4. Choose **Reboot** to reboot your DB instance.

Or choose **Cancel**.

AWS CLI

To reboot a DB instance by using the AWS CLI, call the `reboot-db-instance` command.

Example

For Linux, macOS, or Unix:

```
aws rds reboot-db-instance \  
  --db-instance-identifier mydbinstance
```

For Windows:

```
aws rds reboot-db-instance ^  
  --db-instance-identifier mydbinstance
```

RDS API

To reboot a DB instance by using the Amazon RDS API, call the `RebootDBInstance` operation.

Rebooting an Aurora cluster (Aurora PostgreSQL and Aurora MySQL before version 2.10)

In Aurora PostgreSQL-Compatible Edition, in Aurora MySQL-Compatible Edition version 1, and in Aurora MySQL before version 2.10, you reboot an entire Aurora DB cluster by rebooting the writer DB instance of that cluster. To do so, follow the procedure in [Rebooting a DB instance within an Aurora cluster \(p. 451\)](#).

Rebooting the writer DB instance also initiates a reboot for each reader DB instance in the cluster. That way, any cluster-wide parameter changes are applied to all DB instances at the same time. However, the reboot of all DB instances causes a brief outage for the cluster. The reader DB instances remain unavailable until the writer DB instance finishes rebooting and becomes available.

In the RDS console, the writer DB instance has the value **Writer** under the **Role** column on the **Databases** page. In the RDS CLI, the output of the `describe-db-clusters` command includes a section `DBCClusterMembers`. The `DBCClusterMembers` element representing the writer DB instance has a value of `true` for the `IsClusterWriter` field.

Important

In Aurora MySQL 2.10 and higher, the reboot behavior is different: the reader DB instances typically remain available while you reboot the writer instance. Then you can reboot the reader instances at a convenient time. You can reboot the reader instances on a staggered schedule if you want some reader instances to always be available. For more information, see [Rebooting an Aurora MySQL cluster \(version 2.10 and higher\) \(p. 452\)](#).

Rebooting an Aurora MySQL cluster (version 2.10 and higher)

In Aurora MySQL version 2.10 and higher, you can reboot the writer instance of your Aurora MySQL cluster without rebooting the reader instances in the cluster. Doing so can help maintain high availability

of the cluster for read operations while you reboot the writer instance. You can reboot the reader instances later, on a schedule that's convenient for you. For example, for a production cluster, you might reboot the reader instances one at a time, starting only after the reboot of the primary instance is finished. For each DB instance that you reboot, follow the procedure in [Rebooting a DB instance within an Aurora cluster \(p. 451\)](#).

Before Aurora MySQL 2.10, rebooting the primary instance caused a reboot for each reader instance at the same time. If your Aurora MySQL cluster is running an older version, use the reboot procedure in [Rebooting an Aurora cluster \(Aurora PostgreSQL and Aurora MySQL before version 2.10\) \(p. 452\)](#) instead.

Important

The change to reboot behavior in Aurora MySQL 2.10 and higher is different for Aurora global databases. If you reboot the writer instance for the primary cluster in an Aurora global database, the reader instances in the primary cluster remain available. However, the DB instances in any secondary clusters reboot at the same time.

You frequently reboot the cluster after making changes to cluster parameter groups. You make parameter changes by following the procedures in [Working with DB parameter groups and DB cluster parameter groups \(p. 339\)](#). Suppose that you reboot the writer DB instance in an Aurora MySQL cluster to apply changes to cluster parameters. Some or all of the reader DB instances might continue using the old parameter settings. However, the different parameter settings don't affect the data integrity of the cluster. Any cluster parameters that affect the organization of data files are only used by the writer DB instance. For example, you can update cluster parameters such as `binlog_format` and `innodb_purge_threads` on the writer instance before the reader instances. Only the writer instance is writing binary logs and purging undo records.

For parameters that change how queries interpret SQL statements or query output, you might need to take care to reboot the reader instances immediately. You do this to avoid unexpected application behavior during queries. For example, suppose that you change the `lower_case_table_names` parameter and reboot the writer instance. In this case, the reader instances might not be able to access a newly created table until they are all rebooted.

For a list of all the Aurora MySQL cluster parameters, see [Cluster-level parameters \(p. 1043\)](#).

Tip

Aurora MySQL might still reboot some of the reader instances along with the writer instance if your cluster is processing a workload with high throughput.

The reduction in the number of reboots applies during failover operations also. Aurora MySQL only restarts the writer DB instance and the failover target during a failover. Other reader DB instances in the cluster remain available to continue processing queries through connections to the reader endpoint. Thus, you can improve availability during a failover by having more than one reader DB instance in a cluster.

Checking uptime for Aurora clusters and instances

You can check and monitor the length of time since the last reboot for each DB instance in your Aurora cluster. The Amazon CloudWatch metric `EngineUptime` reports the number of seconds since the last time a DB instance was started. You can examine this metric at a point in time to find out the uptime for the DB instance. You can also monitor this metric over time to detect when the instance is rebooted.

You can also examine the `EngineUptime` metric at the cluster level. The `Minimum` and `Maximum` dimensions report the smallest and largest uptime values for all DB instances in the cluster. To check the most recent time when any reader instance in a cluster was rebooted, or restarted for another reason, monitor the cluster-level metric using the `Minimum` dimension. To check which instance in the cluster has gone the longest without a reboot, monitor the cluster-level metric using the `Maximum` dimension. For example, you might want to confirm that all DB instances in the cluster were rebooted after a configuration change.

Tip

For long-term monitoring, we recommend monitoring the `EngineUptime` metric for individual instances instead of at the cluster level. The cluster-level `EngineUptime` metric is set to zero when a new DB instance is added to the cluster. Such cluster changes can happen as part of maintenance and scaling operations such as those performed by Auto Scaling.

The following CLI examples show how to examine the `EngineUptime` metric for the writer and reader instances in a cluster. The examples use a cluster named `tpch100g`. This cluster has a writer DB instance `instance-1234`. It also has two reader DB instances, `instance-7448` and `instance-6305`.

First, the `reboot-db-instance` command reboots one of the reader instances. The `wait` command waits until the instance is finished rebooting.

```
$ aws rds reboot-db-instance --db-instance-identifier instance-6305
{
    "DBInstance": {
        "DBInstanceIdentifier": "instance-6305",
        "DBInstanceState": "rebooting",
    }
...
$ aws rds wait db-instance-available --db-instance-id instance-6305
```

The CloudWatch `get-metric-statistics` command examines the `EngineUptime` metric over the last five minutes at one-minute intervals. The uptime for the `instance-6305` instance is reset to zero and begins counting upwards again. This AWS CLI example for Linux uses `$()` variable substitution to insert the appropriate timestamps into the CLI commands. It also uses the Linux `sort` command to order the output by the time the metric was collected. That timestamp value is the third field in each line of output.

```
$ aws cloudwatch get-metric-statistics --metric-name "EngineUptime" \
--start-time "$(date -d '5 minutes ago')" --end-time "$(date -d 'now')" \
--period 60 --namespace "AWS/RDS" --statistics Maximum \
--dimensions Name=DBInstanceIdentifier,Value=instance-6305 --output text \
| sort -k 3
EngineUptime
DATAPOINTS 231.0 2021-03-16T18:19:00+00:00 Seconds
DATAPOINTS 291.0 2021-03-16T18:20:00+00:00 Seconds
DATAPOINTS 351.0 2021-03-16T18:21:00+00:00 Seconds
DATAPOINTS 411.0 2021-03-16T18:22:00+00:00 Seconds
DATAPOINTS 471.0 2021-03-16T18:23:00+00:00 Seconds
```

The minimum uptime for the cluster is reset to zero because one of the instances in the cluster was rebooted. The maximum uptime for the cluster isn't reset because at least one of the DB instances in the cluster remained available.

```
$ aws cloudwatch get-metric-statistics --metric-name "EngineUptime" \
--start-time "$(date -d '5 minutes ago')" --end-time "$(date -d 'now')" \
--period 60 --namespace "AWS/RDS" --statistics Minimum \
--dimensions Name=DBClusterIdentifier,Value=tpch100g --output text \
| sort -k 3
EngineUptime
DATAPOINTS 63099.0 2021-03-16T18:12:00+00:00 Seconds
DATAPOINTS 63159.0 2021-03-16T18:13:00+00:00 Seconds
DATAPOINTS 63219.0 2021-03-16T18:14:00+00:00 Seconds
DATAPOINTS 63279.0 2021-03-16T18:15:00+00:00 Seconds
DATAPOINTS 51.0 2021-03-16T18:16:00+00:00 Seconds

$ aws cloudwatch get-metric-statistics --metric-name "EngineUptime" \
--start-time "$(date -d '5 minutes ago')" --end-time "$(date -d 'now')" \
--period 60 --namespace "AWS/RDS" --statistics Maximum \
--dimensions Name=DBClusterIdentifier,Value=tpch100g --output text \
```

```
| sort -k 3
EngineUptime
DATAPOINTS 63389.0 2021-03-16T18:16:00+00:00 Seconds
DATAPOINTS 63449.0 2021-03-16T18:17:00+00:00 Seconds
DATAPOINTS 63509.0 2021-03-16T18:18:00+00:00 Seconds
DATAPOINTS 63569.0 2021-03-16T18:19:00+00:00 Seconds
DATAPOINTS 63629.0 2021-03-16T18:20:00+00:00 Seconds
```

Then another `reboot-db-instance` command reboots the writer instance of the cluster. Another `wait` command pauses until the writer instance is finished rebooting.

```
$ aws rds reboot-db-instance --db-instance-identifier instance-1234
{
  "DBInstanceIdentifier": "instance-1234",
  "DBInstanceState": "rebooting",
  ...
$ aws rds wait db-instance-available --db-instance-id instance-1234
```

Now the `EngineUptime` metric for the writer instance shows that the instance `instance-1234` was rebooted recently. The reader instance `instance-6305` was also rebooted automatically along with the writer instance. This cluster is running Aurora MySQL 2.09, which doesn't keep the reader instances running as the writer instance reboots.

```
$ aws cloudwatch get-metric-statistics --metric-name "EngineUptime" \
--start-time "$(date -d '5 minutes ago')" --end-time "$(date -d 'now')" \
--period 60 --namespace "AWS/RDS" --statistics Maximum \
--dimensions Name=DBInstanceIdentifier,Value=instance-1234 --output text \
| sort -k 3
EngineUptime
DATAPOINTS 63749.0 2021-03-16T18:22:00+00:00 Seconds
DATAPOINTS 63809.0 2021-03-16T18:23:00+00:00 Seconds
DATAPOINTS 63869.0 2021-03-16T18:24:00+00:00 Seconds
DATAPOINTS 41.0 2021-03-16T18:25:00+00:00 Seconds
DATAPOINTS 101.0 2021-03-16T18:26:00+00:00 Seconds

$ aws cloudwatch get-metric-statistics --metric-name "EngineUptime" \
--start-time "$(date -d '5 minutes ago')" --end-time "$(date -d 'now')" \
--period 60 --namespace "AWS/RDS" --statistics Maximum \
--dimensions Name=DBInstanceIdentifier,Value=instance-6305 --output text \
| sort -k 3
EngineUptime
DATAPOINTS 411.0 2021-03-16T18:22:00+00:00 Seconds
DATAPOINTS 471.0 2021-03-16T18:23:00+00:00 Seconds
DATAPOINTS 531.0 2021-03-16T18:24:00+00:00 Seconds
DATAPOINTS 49.0 2021-03-16T18:26:00+00:00 Seconds
```

Examples of Aurora reboot operations

The following Aurora MySQL examples show different combinations of reboot operations for reader and writer DB instances in an Aurora DB cluster. After each reboot, SQL queries demonstrate the uptime for the instances in the cluster.

Topics

- [Finding the writer and reader instances for an Aurora cluster \(p. 456\)](#)
- [Rebooting a single reader instance \(p. 456\)](#)
- [Rebooting the writer instance \(p. 457\)](#)
- [Rebooting the writer and readers independently \(p. 458\)](#)
- [Applying a cluster parameter change to an Aurora MySQL version 2.10 cluster \(p. 461\)](#)

Finding the writer and reader instances for an Aurora cluster

In an Aurora MySQL cluster with multiple DB instances, it's important to know which one is the writer and which ones are the readers. The writer and reader instances also can switch roles when a failover operation happens. Thus, it's best to perform a check like the following before doing any operation that requires a writer or reader instance. In this case, the `False` values for `IsClusterWriter` identify the reader instances, `instance-6305` and `instance-7448`. The `True` value identifies the writer instance, `instance-1234`.

```
$ aws rds describe-db-clusters --db-cluster-id tpch100g \
    --query "[].['Cluster:',DBClusterIdentifier,DBClusterMembers[*]].
    ['Instance:',DBInstanceIdentifier,IsClusterWriter]]" \
    --output text
Cluster:      tpch100g
Instance:     instance-6305      False
Instance:     instance-7448      False
Instance:     instance-1234      True
```

Before we start the examples of rebooting, the writer instance has an uptime of approximately one week. The SQL query in this example shows a MySQL-specific way to check the uptime. You might use this technique in a database application. For another technique that uses the AWS CLI and works for both Aurora engines, see [Checking uptime for Aurora clusters and instances \(p. 453\)](#).

```
$ mysql -h instance-7448.a12345.us-east-1.rds.amazonaws.com -P 3306 -u my-user -p
...
mysql> select date_sub(now(), interval variable_value second) "Last Startup",
       -> time_format(sec_to_time(variable_value),'%H %im') as "Uptime"
       -> from performance_schema.global_status
       -> where variable_name='Uptime';
+-----+-----+
| Last Startup          | Uptime   |
+-----+-----+
| 2021-03-08 17:49:06.000000 | 174h 42m|
+-----+-----+
```

Rebooting a single reader instance

This example reboots one of the reader DB instances. Perhaps this instance was overloaded by a huge query or many concurrent connections. Or perhaps it fell behind the writer instance because of a network issue. After starting the reboot operation, the example uses a `wait` command to pause until the instance becomes available. By that point, the instance has an uptime of a few minutes.

```
$ aws rds reboot-db-instance --db-instance-identifier instance-6305
{
    "DBInstance": {
        "DBInstanceIdentifier": "instance-6305",
        "DBInstanceState": "rebooting",
        ...
    }
}
$ aws rds wait db-instance-available --db-instance-id instance-6305
$ mysql -h instance-6305.a12345.us-east-1.rds.amazonaws.com -P 3306 -u my-user -p
...
mysql> select date_sub(now(), interval variable_value second) "Last Startup",
       -> time_format(sec_to_time(variable_value),'%H %im') as "Uptime"
       -> from performance_schema.global_status
       -> where variable_name='Uptime';
+-----+-----+
| Last Startup          | Uptime   |
+-----+-----+
```

```
+-----+-----+
| 2021-03-16 00:35:02.000000 | 00h 03m |
+-----+-----+
```

Rebooting the reader instance didn't affect the uptime of the writer instance. It still has an uptime of about one week.

```
$ mysql -h instance-7448.a12345.us-east-1.rds.amazonaws.com -P 3306 -u my-user -p
...
mysql> select date_sub(now(), interval variable_value second) "Last Startup",
   -> time_format(sec_to_time(variable_value),'%Hh %im') as "Uptime"
   -> from performance_schema.global_status where variable_name='Uptime';
+-----+-----+
| Last Startup           | Uptime    |
+-----+-----+
| 2021-03-08 17:49:06.000000 | 174h 49m |
+-----+-----+
```

Rebooting the writer instance

This example reboots the writer instance. This cluster is running Aurora MySQL version 2.09. Because the Aurora MySQL version is lower than 2.10, rebooting the writer instance also reboots any reader instances in the cluster.

A wait command pauses until the reboot is finished. Now the uptime for that instance is reset to zero. It's possible that a reboot operation might take substantially different times for writer and reader DB instances. The writer and reader DB instances perform different kinds of cleanup operations depending on their roles.

```
$ aws rds reboot-db-instance --db-instance-identifier instance-1234
{
    "DBInstance": {
        "DBInstanceIdentifier": "instance-1234",
        "DBInstanceStatus": "rebooting",
        ...
    }
}
$ aws rds wait db-instance-available --db-instance-id instance-1234
$ mysql -h instance-1234.a12345.us-east-1.rds.amazonaws.com -P 3306 -u my-user -p
...
mysql> select date_sub(now(), interval variable_value second) "Last Startup",
   -> time_format(sec_to_time(variable_value),'%Hh %im') as "Uptime"
   -> from performance_schema.global_status where variable_name='Uptime';
+-----+-----+
| Last Startup           | Uptime    |
+-----+-----+
| 2021-03-16 00:40:27.000000 | 00h 00m |
+-----+-----+
```

After the reboot for the writer DB instance, both of the reader DB instances also have their uptime reset. Rebooting the writer instance caused the reader instances to reboot also. This behavior applies to Aurora PostgreSQL clusters and to Aurora MySQL clusters before version 2.10.

```
$ mysql -h instance-7448.a12345.us-east-1.rds.amazonaws.com -P 3306 -u my-user -p
...
mysql> select date_sub(now(), interval variable_value second) "Last Startup",
   -> time_format(sec_to_time(variable_value),'%Hh %im') as "Uptime"
   -> from performance_schema.global_status where variable_name='Uptime';
+-----+-----+
| Last Startup           | Uptime    |
+-----+-----+
```

```
+-----+-----+
| 2021-03-16 00:40:35.000000 | 00h 00m |
+-----+-----+
$ mysql -h instance-6305.a12345.us-east-1.rds.amazonaws.com -P 3306 -u my-user -p
...
mysql> select date_sub(now(), interval variable_value second) "Last Startup",
-> time_format(sec_to_time(variable_value),'%Hh %im') as "Uptime"
-> from performance_schema.global_status where variable_name='Uptime';
+-----+-----+
| Last Startup           | Uptime   |
+-----+-----+
| 2021-03-16 00:40:33.000000 | 00h 01m |
+-----+-----+
```

Rebooting the writer and readers independently

These next examples show a cluster that runs Aurora MySQL version 2.10. In this Aurora MySQL version and higher, you can reboot the writer instance without causing reboots for all the reader instances. That way, your query-intensive applications don't experience any outage when you reboot the writer instance. You can reboot the reader instances later. You might do those reboots at a time of low query traffic. You might also reboot the reader instances one at a time. That way, at least one reader instance is always available for the query traffic of your application.

The following example uses a cluster named `cluster-2393`, running Aurora MySQL version `5.7.mysql_aurora.2.10.0`. This cluster has a writer instance named `instance-9404` and three reader instances named `instance-6772`, `instance-2470`, and `instance-5138`.

```
$ aws rds describe-db-clusters --db-cluster-id cluster-2393 \
--query "[*].[['Cluster:',DBClusterIdentifier,DBClusterMembers[*]].\n['Instance:',DBInstanceIdentifier,IsClusterWriter]]" \
--output text
Cluster:      cluster-2393
Instance:     instance-5138      False
Instance:     instance-2470      False
Instance:     instance-6772      False
Instance:     instance-9404      True
```

Checking the `uptime` value of each database instance through the `mysql` command shows that each one has roughly the same uptime. For example, here is the uptime for `instance-5138`.

```
mysql> SHOW GLOBAL STATUS LIKE 'uptime';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| Uptime       | 3866  |
+-----+-----+
```

By using CloudWatch, we can get the corresponding uptime information without actually logging into the instances. That way, an administrator can monitor the database but can't view or change any table data. In this case, we specify a time period spanning five minutes, and check the uptime value every minute. The increasing uptime values demonstrate that the instances weren't restarted during that period.

```
$ aws cloudwatch get-metric-statistics --metric-name "EngineUptime" \
--start-time "$(date -d '5 minutes ago')" --end-time "$(date -d 'now')" --period 60 \
--namespace "AWS/RDS" --statistics Minimum --dimensions
Name=DBInstanceIdentifier,Value=instance-9404 \
--output text | sort -k 3
```

```

EngineUptime
DATAPOINTS 4648.0 2021-03-17T23:42:00+00:00 Seconds
DATAPOINTS 4708.0 2021-03-17T23:43:00+00:00 Seconds
DATAPOINTS 4768.0 2021-03-17T23:44:00+00:00 Seconds
DATAPOINTS 4828.0 2021-03-17T23:45:00+00:00 Seconds
DATAPOINTS 4888.0 2021-03-17T23:46:00+00:00 Seconds

$ aws cloudwatch get-metric-statistics --metric-name "EngineUptime" \
--start-time "$(date -d '5 minutes ago')" --end-time "$(date -d 'now')" --period 60 \
--namespace "AWS/RDS" --statistics Minimum --dimensions
Name=DBInstanceIdentifier,Value=instance-6772 \
--output text | sort -k 3
EngineUptime
DATAPOINTS 4315.0 2021-03-17T23:42:00+00:00 Seconds
DATAPOINTS 4375.0 2021-03-17T23:43:00+00:00 Seconds
DATAPOINTS 4435.0 2021-03-17T23:44:00+00:00 Seconds
DATAPOINTS 4495.0 2021-03-17T23:45:00+00:00 Seconds
DATAPOINTS 4555.0 2021-03-17T23:46:00+00:00 Seconds

```

Now we reboot one of the reader instances, `instance-5138`. We wait for the instance to become available again after the reboot. Now monitoring the uptime over a five-minute period shows that the uptime was reset to zero during that time. The most recent uptime value was measured five seconds after the reboot finished.

```

$ aws rds reboot-db-instance --db-instance-identifier instance-5138
{
    "DBInstanceIdentifier": "instance-5138",
    "DBInstanceState": "rebooting"
}
$ aws rds wait db-instance-available --db-instance-id instance-5138

$ aws cloudwatch get-metric-statistics --metric-name "EngineUptime" \
--start-time "$(date -d '5 minutes ago')" --end-time "$(date -d 'now')" --period 60 \
--namespace "AWS/RDS" --statistics Minimum --dimensions
Name=DBInstanceIdentifier,Value=instance-5138 \
--output text | sort -k 3
EngineUptime
DATAPOINTS 4500.0 2021-03-17T23:46:00+00:00 Seconds
DATAPOINTS 4560.0 2021-03-17T23:47:00+00:00 Seconds
DATAPOINTS 4620.0 2021-03-17T23:48:00+00:00 Seconds
DATAPOINTS 4680.0 2021-03-17T23:49:00+00:00 Seconds
DATAPOINTS 5.0 2021-03-17T23:50:00+00:00 Seconds

```

Next, we perform a reboot for the writer instance, `instance-9404`. We compare the uptime values for the writer instance and one of the reader instances. By doing so, we can see that rebooting the writer didn't cause a reboot for the readers. In versions before Aurora MySQL 2.10, the uptime values for all the readers would be reset at the same time as the writer.

```

$ aws rds reboot-db-instance --db-instance-identifier instance-9404
{
    "DBInstanceIdentifier": "instance-9404",
    "DBInstanceState": "rebooting"
}
$ aws rds wait db-instance-available --db-instance-id instance-9404

$ aws cloudwatch get-metric-statistics --metric-name "EngineUptime" \
--start-time "$(date -d '5 minutes ago')" --end-time "$(date -d 'now')" --period 60 \
--namespace "AWS/RDS" --statistics Minimum --dimensions
Name=DBInstanceIdentifier,Value=instance-9404 \
--output text | sort -k 3
EngineUptime
DATAPOINTS 371.0 2021-03-17T23:57:00+00:00 Seconds
DATAPOINTS 431.0 2021-03-17T23:58:00+00:00 Seconds

```

```

DATAPOINTS 491.0 2021-03-17T23:59:00+00:00 Seconds
DATAPOINTS 551.0 2021-03-18T00:00:00+00:00 Seconds
DATAPOINTS 37.0 2021-03-18T00:01:00+00:00 Seconds

$ aws cloudwatch get-metric-statistics --metric-name "EngineUptime" \
--start-time "$(date -d '5 minutes ago')" --end-time "$(date -d 'now')" --period 60 \
--namespace "AWS/RDS" --statistics Minimum --dimensions
Name=DBInstanceIdentifier,Value=instance-6772 \
--output text | sort -k 3
EngineUptime
DATAPOINTS 5215.0 2021-03-17T23:57:00+00:00 Seconds
DATAPOINTS 5275.0 2021-03-17T23:58:00+00:00 Seconds
DATAPOINTS 5335.0 2021-03-17T23:59:00+00:00 Seconds
DATAPOINTS 5395.0 2021-03-18T00:00:00+00:00 Seconds
DATAPOINTS 5455.0 2021-03-18T00:01:00+00:00 Seconds

```

To make sure that all the reader instances have all the same changes to configuration parameters as the writer instance, reboot all the reader instances after the writer. This example reboots all the readers and then waits until all of them are available before proceeding.

```

$ aws rds reboot-db-instance --db-instance-identifier instance-6772
{
  "DBInstanceIdentifier": "instance-6772",
  "DBInstanceState": "rebooting"
}

$ aws rds reboot-db-instance --db-instance-identifier instance-2470
{
  "DBInstanceIdentifier": "instance-2470",
  "DBInstanceState": "rebooting"
}

$ aws rds reboot-db-instance --db-instance-identifier instance-5138
{
  "DBInstanceIdentifier": "instance-5138",
  "DBInstanceState": "rebooting"
}

$ aws rds wait db-instance-available --db-instance-id instance-6772
$ aws rds wait db-instance-available --db-instance-id instance-2470
$ aws rds wait db-instance-available --db-instance-id instance-5138

```

Now we can see that the writer DB instance has the highest uptime. This instance's uptime value increased steadily throughout the monitoring period. The reader DB instances were all rebooted after the reader. We can see the point within the monitoring period when each reader was rebooted and its uptime was reset to zero.

```

$ aws cloudwatch get-metric-statistics --metric-name "EngineUptime" \
--start-time "$(date -d '5 minutes ago')" --end-time "$(date -d 'now')" --period 60 \
--namespace "AWS/RDS" --statistics Minimum --dimensions
Name=DBInstanceIdentifier,Value=instance-9404 \
--output text | sort -k 3
EngineUptime
DATAPOINTS 457.0 2021-03-18T00:08:00+00:00 Seconds
DATAPOINTS 517.0 2021-03-18T00:09:00+00:00 Seconds
DATAPOINTS 577.0 2021-03-18T00:10:00+00:00 Seconds
DATAPOINTS 637.0 2021-03-18T00:11:00+00:00 Seconds
DATAPOINTS 697.0 2021-03-18T00:12:00+00:00 Seconds

$ aws cloudwatch get-metric-statistics --metric-name "EngineUptime" \
--start-time "$(date -d '5 minutes ago')" --end-time "$(date -d 'now')" --period 60 \
--namespace "AWS/RDS" --statistics Minimum --dimensions
Name=DBInstanceIdentifier,Value=instance-2470 \

```

```

--output text | sort -k 3
EngineUptime
DATAPOINTS 5819.0 2021-03-18T00:08:00+00:00 Seconds
DATAPOINTS 35.0 2021-03-18T00:09:00+00:00 Seconds
DATAPOINTS 95.0 2021-03-18T00:10:00+00:00 Seconds
DATAPOINTS 155.0 2021-03-18T00:11:00+00:00 Seconds
DATAPOINTS 215.0 2021-03-18T00:12:00+00:00 Seconds

$ aws cloudwatch get-metric-statistics --metric-name "EngineUptime" \
--start-time "$(date -d '5 minutes ago')" --end-time "$(date -d 'now')" --period 60 \
--namespace "AWS/RDS" --statistics Minimum --dimensions
Name=DBInstanceIdentifier,Value=instance-5138 \
--output text | sort -k 3
EngineUptime
DATAPOINTS 1085.0 2021-03-18T00:08:00+00:00 Seconds
DATAPOINTS 1145.0 2021-03-18T00:09:00+00:00 Seconds
DATAPOINTS 1205.0 2021-03-18T00:10:00+00:00 Seconds
DATAPOINTS 49.0 2021-03-18T00:11:00+00:00 Seconds
DATAPOINTS 109.0 2021-03-18T00:12:00+00:00 Seconds

```

Applying a cluster parameter change to an Aurora MySQL version 2.10 cluster

The following example demonstrates how to apply a parameter change to all DB instances in your Aurora MySQL 2.10 cluster. With this Aurora MySQL version, you reboot the writer instance and all the reader instances independently.

The example uses the MySQL configuration parameter `lower_case_table_names` for illustration. When this parameter setting is different between the writer and reader DB instances, a query might not be able to access a table declared with an uppercase or mixed-case name. Or if two table names differ only in terms of uppercase and lowercase letters, a query might access the wrong table.

This example shows how to determine the writer and reader instances in the cluster by examining the `IsClusterWriter` attribute of each instance. The cluster is named `cluster-2393`. The cluster has a writer instance named `instance-9404`. The reader instances in the cluster are named `instance-5138` and `instance-2470`.

```

$ aws rds describe-db-clusters --db-cluster-id cluster-2393 \
--query '*[].[DBClusterIdentifier,DBClusterMembers[*]]. \
[DBInstanceIdentifier,IsClusterWriter]' \
--output text
cluster-2393
instance-5138      False
instance-2470      False
instance-9404      True

```

To demonstrate the effects of changing the `lower_case_table_names` parameter, we set up two DB cluster parameter groups. The `lower-case-table-names-0` parameter group has this parameter set to 0. The `lower-case-table-names-1` parameter group has this parameter group set to 1.

```

$ aws rds create-db-cluster-parameter-group --description 'lower-case-table-names-0' \
--db-parameter-group-family aurora-mysql5.7 \
--db-cluster-parameter-group-name lower-case-table-names-0
{
    "DBClusterParameterGroup": {
        "DBClusterParameterGroupName": "lower-case-table-names-0",
        "DBParameterGroupFamily": "aurora-mysql5.7",
        "Description": "lower-case-table-names-0"
    }
}

```

```
$ aws rds create-db-cluster-parameter-group --description 'lower-case-table-names-1' \
--db-parameter-group-family aurora-mysql5.7 \
--db-cluster-parameter-group-name lower-case-table-names-1
{
    "DBClusterParameterGroup": {
        "DBClusterParameterGroupName": "lower-case-table-names-1",
        "DBParameterGroupFamily": "aurora-mysql5.7",
        "Description": "lower-case-table-names-1"
    }
}

$ aws rds modify-db-cluster-parameter-group \
--db-cluster-parameter-group-name lower-case-table-names-0 \
--parameters ParameterName=lower_case_table_names,ParameterValue=0,ApplyMethod=pending-
reboot
{
    "DBClusterParameterGroupName": "lower-case-table-names-0"
}

$ aws rds modify-db-cluster-parameter-group \
--db-cluster-parameter-group-name lower-case-table-names-1 \
--parameters ParameterName=lower_case_table_names,ParameterValue=1,ApplyMethod=pending-
reboot
{
    "DBClusterParameterGroupName": "lower-case-table-names-1"
}
```

The default value of `lower_case_table_names` is 0. With this parameter setting, the table `foo` is distinct from the table `FOO`. This example verifies that the parameter is still at its default setting. Then the example creates three tables that differ only in uppercase and lowercase letters in their names.

```
mysql> create database lctn;
Query OK, 1 row affected (0.07 sec)

mysql> use lctn;
Database changed
mysql> select @@lower_case_table_names;
+-----+
| @@lower_case_table_names |
+-----+
|          0           |
+-----+

mysql> create table foo (s varchar(128));
mysql> insert into foo values ('Lowercase table name foo');

mysql> create table Foo (s varchar(128));
mysql> insert into Foo values ('Mixed-case table name Foo');

mysql> create table FOO (s varchar(128));
mysql> insert into FOO values ('Uppercase table name FOO');

mysql> select * from foo;
+-----+
| s           |
+-----+
| Lowercase table name foo |
+-----+

mysql> select * from Foo;
+-----+
| s           |
+-----+
```

```
| Mixed-case table name Foo |
+-----+
mysql> select * from FOO;
+-----+
| s |
+-----+
| Uppercase table name FOO |
+-----+
```

Next, we associate the DB parameter group with the cluster to set the `lower_case_table_names` parameter to 1. This change only takes effect after each DB instance is rebooted.

```
$ aws rds modify-db-cluster --db-cluster-identifier cluster-2393 \
--db-cluster-parameter-group-name lower-case-table-names-1
{
  "DBClusterIdentifier": "cluster-2393",
  "DBClusterParameterGroup": "lower-case-table-names-1",
  "Engine": "aurora-mysql",
  "EngineVersion": "5.7.mysql_aurora.2.10.0"
}
```

The first reboot we do is for the writer DB instance. Then we wait for the instance to become available again. At that point, we connect to the writer endpoint and verify that the writer instance has the changed parameter value. The `SHOW TABLES` command confirms that the database contains the three different tables. However, queries that refer to tables named `foo`, `Foo`, or `FOO` all access the table whose name is all-lowercase, `foo`.

```
# Rebooting the writer instance
$ aws rds reboot-db-instance --db-instance-identifier instance-9404
$ aws rds wait db-instance-available --db-instance-id instance-9404
```

Now, queries using the cluster endpoint show the effects of the parameter change. Whether the table name in the query is uppercase, lowercase, or mixed case, the SQL statement accesses the table whose name is all lowercase.

```
mysql> select @@lower_case_table_names;
+-----+
| @@lower_case_table_names |
+-----+
| 1 |
+-----+

mysql> use lctn;
mysql> show tables;
+-----+
| Tables_in_lctn |
+-----+
| FOO           |
| Foo          |
| foo          |
+-----+

mysql> select * from foo;
+-----+
| s |
+-----+
| Lowercase table name foo |
+-----+

mysql> select * from Foo;
+-----+
```

```

| s
+-----+
| Lowercase table name foo |
+-----+

mysql> select * from FOO;
+-----+
| s
+-----+
| Lowercase table name foo |
+-----+

```

The next example shows the same queries as the previous one. In this case, the queries use the reader endpoint and run on one of the reader DB instances. Those instances haven't been rebooted yet. Thus, they still have the original setting for the `lower_case_table_names` parameter. That means that queries can access each of the `foo`, `Foo`, and `FOO` tables.

```

mysql> select @@lower_case_table_names;
+-----+
| @@lower_case_table_names |
+-----+
| 0 |
+-----+

mysql> use lctn;

mysql> select * from foo;
+-----+
| s
+-----+
| Lowercase table name foo |
+-----+

mysql> select * from Foo;
+-----+
| s
+-----+
| Mixed-case table name Foo |
+-----+

mysql> select * from FOO;
+-----+
| s
+-----+
| Uppercase table name FOO |
+-----+

```

Next, we reboot one of the reader instances and wait for it to become available again.

```

$ aws rds reboot-db-instance --db-instance-identifier instance-2470
{
    "DBInstanceIdentifier": "instance-2470",
    "DBInstanceState": "rebooting"
}
$ aws rds wait db-instance-available --db-instance-id instance-2470

```

While connected to the instance endpoint for `instance-2470`, a query shows that the new parameter is in effect.

```

mysql> select @@lower_case_table_names;
+-----+
| @@lower_case_table_names |
+-----+

```

```
+-----+
|      1 |
+-----+
```

At this point, the two reader instances in the cluster are running with different `lower_case_table_names` settings. Thus, any connection to the reader endpoint of the cluster uses a value for this setting that's unpredictable. It's important to immediately reboot the other reader instance so that they both have consistent settings.

```
$ aws rds reboot-db-instance --db-instance-identifier instance-5138
{
  "DBInstanceIdentifier": "instance-5138",
  "DBInstanceState": "rebooting"
}
$ aws rds wait db-instance-available --db-instance-id instance-5138
```

The following example confirms that all the reader instances have the same setting for the `lower_case_table_names` parameter. The commands check the `lower_case_table_names` setting value on each reader instance. Then the same command using the reader endpoint demonstrates that each connection to the reader endpoint uses one of the reader instances, but which one isn't predictable.

```
# Check lower_case_table_names setting on each reader instance.

$ mysql -h instance-5138.a12345.us-east-1.rds.amazonaws.com \
  -u my-user -p -e 'select @@aurora_server_id, @@lower_case_table_names'
+-----+
| @@aurora_server_id | @@lower_case_table_names |
+-----+
| instance-5138      |                      1 |
+-----+

$ mysql -h instance-2470.a12345.us-east-1.rds.amazonaws.com \
  -u my-user -p -e 'select @@aurora_server_id, @@lower_case_table_names'
+-----+
| @@aurora_server_id | @@lower_case_table_names |
+-----+
| instance-2470      |                      1 |
+-----+

# Check lower_case_table_names setting on the reader endpoint of the cluster.

$ mysql -h cluster-2393.cluster-ro-a12345.us-east-1.rds.amazonaws.com \
  -u my-user -p -e 'select @@aurora_server_id, @@lower_case_table_names'
+-----+
| @@aurora_server_id | @@lower_case_table_names |
+-----+
| instance-5138      |                      1 |
+-----+

# Run query on writer instance

$ mysql -h cluster-2393.cluster-a12345.us-east-1.rds.amazonaws.com \
  -u my-user -p -e 'select @@aurora_server_id, @@lower_case_table_names'
+-----+
| @@aurora_server_id | @@lower_case_table_names |
+-----+
| instance-9404      |                      1 |
+-----+
```

With the parameter change applied everywhere, we can see the effect of setting `lower_case_table_names=1`. Whether the table is referred to as `foo`, `Foo`, or `FOO` the query converts the name to `foo` and accesses the same table in each case.

```
mysql> use lctn;
mysql> select * from foo;
+-----+
| s |
+-----+
| Lowercase table name foo |
+-----+

mysql> select * from Foo;
+-----+
| s |
+-----+
| Lowercase table name foo |
+-----+

mysql> select * from FOO;
+-----+
| s |
+-----+
| Lowercase table name foo |
+-----+
```

Deleting Aurora DB clusters and DB instances

You can delete an Aurora DB cluster when you don't need it any longer. Doing so removes the cluster volume containing all your data. Before deleting the cluster, you can save a snapshot of your data. You can restore the snapshot later to create a new cluster containing the same data.

You can also delete DB instances from a cluster, while preserving the cluster itself and the data that it contains. Doing so can help you reduce charges if the cluster isn't busy and doesn't need the computation capacity of multiple DB instances.

Topics

- [Deleting an Aurora DB cluster \(p. 467\)](#)
- [Deletion protection for Aurora clusters \(p. 471\)](#)
- [Deleting a stopped Aurora cluster \(p. 472\)](#)
- [Deleting Aurora MySQL clusters that are read replicas \(p. 472\)](#)
- [The final snapshot when deleting a cluster \(p. 472\)](#)
- [Deleting a DB instance from an Aurora DB cluster \(p. 472\)](#)

Deleting an Aurora DB cluster

Aurora doesn't provide a single-step method to delete a DB cluster. This design choice is intended to prevent you from accidentally losing data or taking your application offline. Aurora applications are typically mission-critical and require high availability. Thus, Aurora makes it easy to scale the capacity of the cluster up and down by adding and removing DB instances. However, removing the cluster itself requires you to make a separate choice.

Use the following general procedure to remove all the DB instances from a cluster and then delete the cluster itself.

1. Delete any reader instances in the cluster. Use the procedure in [Deleting a DB instance from an Aurora DB cluster \(p. 472\)](#). If the cluster has any reader instances, deleting one of the instances just reduces the computation capacity of the cluster. Deleting the reader instances first ensures that the cluster remains available throughout the procedure and doesn't perform unnecessary failover operations.
2. Delete the writer instance from the cluster. Again, use the procedure in [Deleting a DB instance from an Aurora DB cluster \(p. 472\)](#).

If you use the AWS Management Console, this is the final step. Deleting the final DB instance in a DB cluster through the console automatically deletes the DB cluster and the data in the cluster volume. At this point, Aurora prompts you to optionally create a snapshot before deleting the cluster. Aurora also requires you to confirm that you intend to delete the cluster.

3. CLI and API only: If you delete the DB instances using the AWS CLI or the RDS API, the cluster and its associated cluster volume remain even after you delete all the DB instances. To delete the cluster itself, you call the `delete-db-cluster` CLI command or `DeleteDBCluster` API operation when the cluster has zero associated DB instances. At this point, you choose whether to create a snapshot of the cluster volume. Doing so preserves the data from the cluster if you might need it later.

Topics

- [Deleting an empty Aurora cluster \(p. 468\)](#)
- [Deleting an Aurora cluster with a single DB instance \(p. 468\)](#)
- [Deleting an Aurora cluster with multiple DB instances \(p. 469\)](#)

Deleting an empty Aurora cluster

If you use the AWS Management Console, Aurora automatically deletes your cluster when you delete the last DB instance in that cluster. Thus, the procedures for deleting an empty cluster only apply when you use the AWS CLI or the RDS API.

Tip

You can keep a cluster with no DB instances to preserve your data without incurring CPU charges for the cluster. You can quickly start using the cluster again by creating one or more new DB instances for the cluster. You can perform Aurora-specific administrative operations on the cluster while it doesn't have any associated DB instances. You just can't access the data or perform any operations that require connecting to a DB instance.

To delete an empty Aurora DB cluster by using the AWS CLI, call the [delete-db-cluster](#) command.

To delete an empty Aurora DB cluster by using the Amazon RDS API, call the [DeleteDBInstance](#) operation.

Suppose that the empty cluster `deleteme-zero-instances` was only used for development and testing and doesn't contain any important data. In that case, you don't need to preserve a snapshot of the cluster volume when you delete the cluster. The following example demonstrates that a cluster doesn't contain any DB instances, and then deletes the empty cluster without creating a final snapshot.

```
$ aws rds describe-db-clusters --db-cluster-identifier deleteme-zero-instances --output text \
  --query '*[].[{"Cluster":DBClusterIdentifier,DBClusterMembers[*].["Instance":DBInstanceIdentifier,IsClusterWriter]}]
Cluster:          deleteme-zero-instances

$ aws rds delete-db-cluster --db-cluster-identifier deleteme-zero-instances --skip-final-snapshot
{
  "DBClusterIdentifier": "deleteme-zero-instances",
  "Status": "available",
  "Engine": "aurora-mysql"
}
```

Deleting an Aurora cluster with a single DB instance

If you try to delete the last DB instance in your Aurora cluster, the behavior depends on the method you use. You can delete the last DB instance using the AWS Management Console. Doing so also deletes the DB cluster. You can also delete the last DB instance through the AWS CLI or API, even if the DB cluster has deletion protection enabled. In this case, the DB cluster itself still exists and your data is preserved. You can access the data again by attaching a new DB instance to the cluster.

The following example shows how the `delete-db-cluster` command doesn't work when the cluster still has any associated DB instances. This cluster has a single writer DB instance. When we examine the DB instances in the cluster, we check the `IsClusterWriter` attribute of each one. The cluster could have zero or one writer DB instance. A value of `true` signifies a writer DB instance. A value of `false` signifies a reader DB instance. The cluster could have zero, one, or many reader DB instances. In this case, we delete the writer DB instance using the `delete-db-instance` command. As soon as the DB instance goes into deleting state, we can delete the cluster also. For this example also, suppose that the cluster doesn't contain any data worth preserving and so we don't create a snapshot of the cluster volume.

```
$ aws rds delete-db-cluster --db-cluster-identifier deleteme-writer-only --skip-final-snapshot
An error occurred (InvalidDBClusterStateException) when calling the DeleteDBCluster operation:
  Cluster cannot be deleted, it still contains DB instances in non-deleting state.

$ aws rds describe-db-clusters --db-cluster-identifier deleteme-writer-only \
```

```

--query '*[].[DBClusterIdentifier,Status,DBClusterMembers[*].[DBInstanceIdentifier,IsClusterWriter]]'
[
  [
    [
      "deleteme-writer-only",
      "available",
      [
        [
          "instance-2130",
          true
        ]
      ]
    ]
  ]
]

$ aws rds delete-db-instance --db-instance-identifier instance-2130
{
  "DBInstanceIdentifier": "instance-2130",
  "DBInstanceState": "deleting",
  "Engine": "aurora-mysql"
}

$ aws rds delete-db-cluster --db-cluster-identifier deleteme-writer-only --skip-final-snapshot
{
  "DBClusterIdentifier": "deleteme-writer-only",
  "Status": "available",
  "Engine": "aurora-mysql"
}

```

Deleting an Aurora cluster with multiple DB instances

If your cluster contains multiple DB instances, typically there is a single writer instance and one or more reader instances. The reader instances help with high availability, by being on standby to take over if the writer instance encounters a problem. You can also use reader instances to scale the cluster up to handle a read-intensive workload without adding overhead to the writer instance.

To delete a cluster with multiple reader DB instances, you delete the reader instances first and then the writer instance. If you use the AWS Management Console, deleting the writer instance automatically deletes the cluster afterwards. If you use the AWS CLI or RDS API, deleting the writer instance leaves the cluster and its data in place. In that case, you delete the cluster through a separate command or API operation.

- For the procedure to delete an Aurora DB instance, see [Deleting a DB instance from an Aurora DB cluster \(p. 472\)](#).
- For the procedure to delete the writer DB instance in an Aurora cluster, see [Deleting an Aurora cluster with a single DB instance \(p. 468\)](#).
- For the procedure to delete an empty Aurora cluster, see [Deleting an empty Aurora cluster \(p. 468\)](#).

This example shows how to delete a cluster containing both a writer DB instance and a single reader DB instance. The `describe-db-clusters` output shows that `instance-7384` is the writer instance and `instance-1039` is the reader instance. The example deletes the reader instance first, because deleting the writer instance while a reader instance still existed would cause a failover operation. It doesn't make sense to promote the reader instance to a writer if you plan to delete that instance also. Again, suppose that these `db.t2.small` instances are only used for development and testing, and so the delete operation skips the final snapshot.

```
$ aws rds delete-db-cluster --db-cluster-identifier deleteme-writer-and-reader --skip-final-snapshot
```

```
An error occurred (InvalidDBClusterStateException) when calling the DeleteDBCluster operation:  
Cluster cannot be deleted, it still contains DB instances in non-deleting state.

$ aws rds describe-db-clusters --db-cluster-identifier deleteme-writer-and-reader --output  
text \  
--query '*[].[{"Cluster":DBClusterIdentifier,DBClusterMembers[*].  
["Instance":DBInstanceIdentifier,IsClusterWriter]}]  
Cluster:      deleteme-writer-and-reader  
Instance:     instance-1039  False  
Instance:     instance-7384  True

$ aws rds delete-db-instance --db-instance-identifier instance-1039  
{  
  "DBInstanceIdentifier": "instance-1039",  
  "DBInstanceState": "deleting",  
  "Engine": "aurora-mysql"  
}

$ aws rds delete-db-instance --db-instance-identifier instance-7384  
{  
  "DBInstanceIdentifier": "instance-7384",  
  "DBInstanceState": "deleting",  
  "Engine": "aurora-mysql"  
}

$ aws rds delete-db-cluster --db-cluster-identifier deleteme-writer-and-reader --skip-  
final-snapshot  
{  
  "DBClusterIdentifier": "deleteme-writer-and-reader",  
  "Status": "available",  
  "Engine": "aurora-mysql"  
}
```

The following example shows how to delete a DB cluster containing a writer DB instance and multiple reader DB instances. It uses concise output from the `describe-db-clusters` command to get a report of the writer and reader instances. Again, we delete all reader DB instances before deleting the writer DB instance. It doesn't matter what order we delete the reader DB instances in. Suppose that this cluster with several DB instances does contain data worth preserving. Thus, the `delete-db-cluster` command in this example includes the `--no-skip-final-snapshot` and `--final-db-snapshot-identifier` parameters to specify the details of the snapshot to create.

```
$ aws rds describe-db-clusters --db-cluster-identifier deleteme-multiple-readers --output  
text \  
--query '*[].[{"Cluster":DBClusterIdentifier,DBClusterMembers[*].  
["Instance":DBInstanceIdentifier,IsClusterWriter]}]  
Cluster:      deleteme-multiple-readers  
Instance:     instance-1010  False  
Instance:     instance-5410  False  
Instance:     instance-9948  False  
Instance:     instance-8451  True

$ aws rds delete-db-instance --db-instance-identifier instance-1010  
{  
  "DBInstanceIdentifier": "instance-1010",  
  "DBInstanceState": "deleting",  
  "Engine": "aurora-mysql"  
}

$ aws rds delete-db-instance --db-instance-identifier instance-5410  
{  
  "DBInstanceIdentifier": "instance-5410",  
  "DBInstanceState": "deleting",  
  "Engine": "aurora-mysql"  
}
```

```
$ aws rds delete-db-instance --db-instance-identifier instance-9948
{
    "DBInstanceIdentifier": "instance-9948",
    "DBInstanceState": "deleting",
    "Engine": "aurora-mysql"
}

$ aws rds delete-db-instance --db-instance-identifier instance-8451
{
    "DBInstanceIdentifier": "instance-8451",
    "DBInstanceState": "deleting",
    "Engine": "aurora-mysql"
}

$ aws rds delete-db-cluster --db-cluster-identifier deleteme-multiple-readers --no-skip-
final-snapshot \
    --final-db-snapshot-identifier deleteme-multiple-readers-snapshot-11-7087
{
    "DBClusterIdentifier": "deleteme-multiple-readers",
    "Status": "available",
    "Engine": "aurora-mysql"
}
```

The following example shows how to confirm that Aurora created the requested snapshot. You can request details for the specific snapshot by specifying its identifier **deleteme-multiple-readers-snapshot-11-7087**. You can also get a report of all snapshots for the cluster that was deleted by specifying the cluster identifier **deleteme-multiple-readers**. Both of those commands return information about the same snapshot.

```
$ aws rds describe-db-cluster-snapshots --db-cluster-snapshot-identifier deleteme-multiple-
readers-snapshot-11-7087
{
    "DBClusterSnapshots": [
        {
            "AvailabilityZones": [],
            "DBClusterSnapshotIdentifier": "deleteme-multiple-readers-snapshot-11-7087",
            "DBClusterIdentifier": "deleteme-multiple-readers",
            "SnapshotCreateTime": "11T01:40:07.354000+00:00",
            "Engine": "aurora-mysql",
            ...
        }
    ]
}

$ aws rds describe-db-cluster-snapshots --db-cluster-identifier deleteme-multiple-readers
{
    "DBClusterSnapshots": [
        {
            "AvailabilityZones": [],
            "DBClusterSnapshotIdentifier": "deleteme-multiple-readers-snapshot-11-7087",
            "DBClusterIdentifier": "deleteme-multiple-readers",
            "SnapshotCreateTime": "11T01:40:07.354000+00:00",
            "Engine": "aurora-mysql",
            ...
        }
    ]
}
```

Deletion protection for Aurora clusters

You can't delete clusters that have deletion protection enabled. You can delete DB instances within the cluster, but not the cluster itself. That way, the cluster volume containing all your data is safe from accidental deletion. Aurora enforces deletion protection for a DB cluster whether you try to delete the cluster using the console, the AWS CLI, or the RDS API.

Deletion protection is enabled by default when you create a production DB cluster using the AWS Management Console. However, deletion protection is disabled by default if you create a cluster using

the AWS CLI or API. Enabling or disabling deletion protection doesn't cause an outage. To be able to delete the cluster, modify the cluster and disable deletion protection. For more information about turning deletion protection on and off, see [Modifying the DB cluster by using the console, CLI, and API \(p. 372\)](#).

Tip

Even if all the DB instances are deleted, you can access the data by creating a new DB instance in the cluster.

Deleting a stopped Aurora cluster

You can't delete a cluster if it's in the `stopped` state. In this case, start the cluster before deleting it. For more information, see [Starting an Aurora DB cluster \(p. 370\)](#).

Deleting Aurora MySQL clusters that are read replicas

For Aurora MySQL, you can't delete a DB instance in a DB cluster if both of the following conditions are true:

- The DB cluster is a read replica of another Aurora DB cluster.
- The DB instance is the only instance in the DB cluster.

To delete a DB instance in this case, first promote the DB cluster so that it's no longer a read replica. After the promotion completes, you can delete the final DB instance in the DB cluster. For more information, see [Replicating Amazon Aurora MySQL DB clusters across AWS Regions \(p. 922\)](#).

The final snapshot when deleting a cluster

Throughout this section, the examples show how you can choose whether to take a final snapshot when you delete an Aurora cluster. If you choose to take a final snapshot but the name you specify matches an existing snapshot, the operation stops with an error. In this case, examine the snapshot details to confirm if it represents your current detail or if it is an older snapshot. If the existing snapshot doesn't have the latest data that you want to preserve, rename the snapshot and try again, or specify a different name for the `final_snapshot` parameter.

Deleting a DB instance from an Aurora DB cluster

You can delete a DB instance from an Aurora DB cluster as part of the process of deleting the entire cluster. If your cluster contains a certain number of DB instances, then deleting the cluster requires deleting each of those DB instances. You can also delete one or more reader instances from a cluster while leaving the cluster running. You might do so to reduce compute capacity and associated charges if your cluster isn't busy.

To delete a DB instance, you specify the name of the instance.

You can delete a DB instance using the AWS Management Console, the AWS CLI, or the RDS API.

For Aurora DB clusters, deleting a DB instance doesn't necessarily delete the entire cluster. You can delete a DB instance in an Aurora cluster to reduce compute capacity and associated charges when the cluster isn't busy. For information about the special circumstances for Aurora clusters that have one DB instance or zero DB instances, see [Deleting an Aurora cluster with a single DB instance \(p. 468\)](#) and [Deleting an empty Aurora cluster \(p. 468\)](#).

Note

You can't delete a DB cluster when deletion protection is enabled for it. For more information, see [Deletion protection for Aurora clusters \(p. 471\)](#).

You can disable deletion protection by modifying the DB cluster. For more information, see [Modifying an Amazon Aurora DB cluster \(p. 372\)](#).

Console

To delete a DB instance

1. Sign in to the AWS Management Console and open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. In the navigation pane, choose **Databases**, and then choose the DB instance that you want to delete.
3. For **Actions**, choose **Delete**.
4. Enter **delete me** in the box.
5. Choose **Delete**.

AWS CLI

To delete a DB instance by using the AWS CLI, call the `delete-db-instance` command and specify the `--db-instance-identifier` value.

Example

For Linux, macOS, or Unix:

```
aws rds delete-db-instance \
--db-instance-identifier mydbinstance
```

For Windows:

```
aws rds delete-db-instance ^
--db-instance-identifier mydbinstance
```

RDS API

To delete a DB instance by using the Amazon RDS API, call the `DeleteDBInstance` operation and specify the `DBInstanceIdentifier` parameter.

Note

When the status for a DB instance is `deleting`, its CA certificate value doesn't appear in the RDS console or in output for AWS CLI commands or RDS API operations. For more information about CA certificates, see [Using SSL/TLS to encrypt a connection to a DB cluster \(p. 1714\)](#).

Tagging Amazon RDS resources

You can use Amazon RDS tags to add metadata to your Amazon RDS resources. You can use the tags to add your own notations about database instances, snapshots, Aurora clusters, and so on. Doing so can help you to document your Amazon RDS resources. You can also use the tags with automated maintenance procedures.

In particular, you can use these tags with IAM policies to manage access to Amazon RDS resources and to control what actions can be applied to the Amazon RDS resources. You can also use these tags to track costs by grouping expenses for similarly tagged resources.

You can tag the following Amazon RDS resources:

- DB instances
- DB clusters
- Read replicas
- DB snapshots
- DB cluster snapshots
- Reserved DB instances
- Event subscriptions
- DB option groups
- DB parameter groups
- DB cluster parameter groups
- DB security groups
- DB subnet groups
- RDS Proxies
- RDS Proxy endpoints

Note

Currently, you can't tag RDS Proxies and RDS Proxy endpoints by using the AWS Management Console.

Topics

- [Overview of Amazon RDS resource tags \(p. 474\)](#)
- [Using tags for access control with IAM \(p. 475\)](#)
- [Using tags to produce detailed billing reports \(p. 475\)](#)
- [Adding, listing, and removing tags \(p. 476\)](#)
- [Using the AWS Tag Editor \(p. 478\)](#)
- [Copying tags to DB cluster snapshots \(p. 478\)](#)
- [Tutorial: Use tags to specify which Aurora DB clusters to stop \(p. 479\)](#)

Overview of Amazon RDS resource tags

An Amazon RDS tag is a name-value pair that you define and associate with an Amazon RDS resource. The name is referred to as the key. Supplying a value for the key is optional. You can use tags to assign arbitrary information to an Amazon RDS resource. You can use a tag key, for example, to define a category, and the tag value might be an item in that category. For example, you might define a tag key

of "project" and a tag value of "Salix", indicating that the Amazon RDS resource is assigned to the Salix project. You can also use tags to designate Amazon RDS resources as being used for test or production by using a key such as `environment=test` or `environment=production`. We recommend that you use a consistent set of tag keys to make it easier to track metadata associated with Amazon RDS resources.

In addition, you can use conditions in your IAM policies to control access to AWS resources based on the tags on that resource. You can do this by using the global `aws:ResourceTag/tag-key` condition key. For more information, see [Controlling access to AWS resources](#) in the *AWS Identity and Access Management User Guide*.

Each Amazon RDS resource has a tag set, which contains all the tags that are assigned to that Amazon RDS resource. A tag set can contain as many as 50 tags, or it can be empty. If you add a tag to an Amazon RDS resource that has the same key as an existing tag on resource, the new value overwrites the old value.

AWS does not apply any semantic meaning to your tags; tags are interpreted strictly as character strings. Amazon RDS can set tags on a DB instance or other Amazon RDS resources, depending on the settings that you use when you create the resource. For example, Amazon RDS might add a tag indicating that a DB instance is for production or for testing.

- The tag key is the required name of the tag. The string value can be from 1 to 128 Unicode characters in length and cannot be prefixed with `aws:` or `rds:`. The string can contain only the set of Unicode letters, digits, white-space, `'`, `!`, `:`, `/`, `=`, `+`, `-`, `@` (Java regex: `^([\u0000-\uFFFF]\u0000-\uFFFF]*$)`).
- The tag value is an optional string value of the tag. The string value can be from 1 to 256 Unicode characters in length and cannot be prefixed with `aws:`. The string can contain only the set of Unicode letters, digits, white-space, `'`, `!`, `:`, `/`, `=`, `+`, `-`, `@` (Java regex: `^([\u0000-\uFFFF]\u0000-\uFFFF]*$)`).

Values do not have to be unique in a tag set and can be null. For example, you can have a key-value pair in a tag set of `project=Trinity` and `cost-center=Trinity`.

You can use the AWS Management Console, the command line interface, or the Amazon RDS API to add, list, and delete tags on Amazon RDS resources. When using the command line interface or the Amazon RDS API, you must provide the Amazon Resource Name (ARN) for the Amazon RDS resource you want to work with. For more information about constructing an ARN, see [Constructing an ARN for Amazon RDS \(p. 482\)](#).

Tags are cached for authorization purposes. Because of this, additions and updates to tags on Amazon RDS resources can take several minutes before they are available.

Using tags for access control with IAM

You can use tags with IAM policies to manage access to Amazon RDS resources and to control what actions can be applied to the Amazon RDS resources.

For information on managing access to tagged resources with IAM policies, see [Identity and access management in Amazon Aurora \(p. 1724\)](#).

Using tags to produce detailed billing reports

You can also use tags to track costs by grouping expenses for similarly tagged resources.

Use tags to organize your AWS bill to reflect your own cost structure. To do this, sign up to get your AWS account bill with tag key values included. Then, to see the cost of combined resources, organize your

billing information according to resources with the same tag key values. For example, you can tag several resources with a specific application name, and then organize your billing information to see the total cost of that application across several services. For more information, see [Using Cost Allocation Tags](#) in the *AWS Billing and Cost Management User Guide*.

Note

You can add a tag to a snapshot, however, your bill will not reflect this grouping.

Adding, listing, and removing tags

The following procedures show how to perform typical tagging operations on resources related to DB instances and Aurora DB clusters.

Console

The process to tag an Amazon RDS resource is similar for all resources. The following procedure shows how to tag an Amazon RDS DB instance.

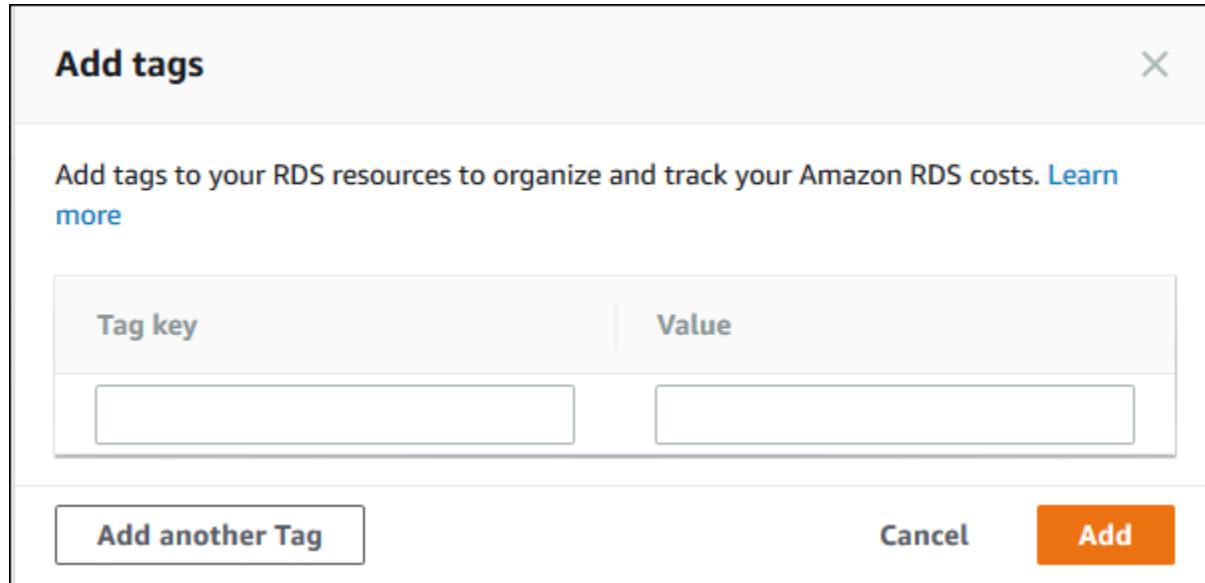
To add a tag to a DB instance

1. Sign in to the AWS Management Console and open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. In the navigation pane, choose **Databases**.

Note

To filter the list of DB instances in the **Databases** pane, enter a text string for **Filter databases**. Only DB instances that contain the string appear.

3. Choose the name of the DB instance that you want to tag to show its details.
4. In the details section, scroll down to the **Tags** section.
5. Choose **Add**. The **Add tags** window appears.



6. Enter a value for **Tag key** and **Value**.
7. To add another tag, you can choose **Add another Tag** and enter a value for its **Tag key** and **Value**.
Repeat this step as many times as necessary.
8. Choose **Add**.

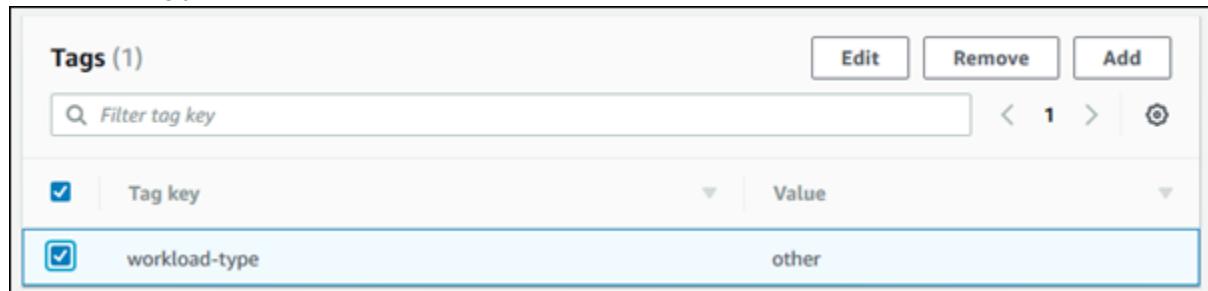
To delete a tag from a DB instance

1. Sign in to the AWS Management Console and open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. In the navigation pane, choose **Databases**.

Note

To filter the list of DB instances in the **Databases** pane, enter a text string in the **Filter databases** box. Only DB instances that contain the string appear.

3. Choose the name of the DB instance to show its details.
4. In the details section, scroll down to the **Tags** section.
5. Choose the tag you want to delete.



6. Choose **Delete**, and then choose **Delete** in the **Delete tags** window.

AWS CLI

You can add, list, or remove tags for a DB instance using the AWS CLI.

- To add one or more tags to an Amazon RDS resource, use the AWS CLI command [add-tags-to-resource](#).
- To list the tags on an Amazon RDS resource, use the AWS CLI command [list-tags-for-resource](#).
- To remove one or more tags from an Amazon RDS resource, use the AWS CLI command [remove-tags-from-resource](#).

To learn more about how to construct the required ARN, see [Constructing an ARN for Amazon RDS \(p. 482\)](#).

RDS API

You can add, list, or remove tags for a DB instance using the Amazon RDS API.

- To add a tag to an Amazon RDS resource, use the [AddTagsToResource](#) operation.
- To list tags that are assigned to an Amazon RDS resource, use the [ListTagsForResource](#).
- To remove tags from an Amazon RDS resource, use the [RemoveTagsFromResource](#) operation.

To learn more about how to construct the required ARN, see [Constructing an ARN for Amazon RDS \(p. 482\)](#).

When working with XML using the Amazon RDS API, tags use the following schema:

```
<Tagging>
  <TagSet>
    <Tag>
      <Key>Project</Key>
```

```

        <Value>Trinity</Value>
    </Tag>
    <Tag>
        <Key>User</Key>
        <Value>Jones</Value>
    </Tag>
</TagSet>
</Tagging>

```

The following table provides a list of the allowed XML tags and their characteristics. Values for Key and Value are case-dependent. For example, project=Trinity and PROJECT=Trinity are two distinct tags.

Tagging element	Description
TagSet	A tag set is a container for all tags assigned to an Amazon RDS resource. There can be only one tag set per resource. You work with a TagSet only through the Amazon RDS API.
Tag	A tag is a user-defined key-value pair. There can be from 1 to 50 tags in a tag set.
Key	A key is the required name of the tag. The string value can be from 1 to 128 Unicode characters in length and cannot be prefixed with <code>aws:</code> or <code>rds:</code> . The string can only contain only the set of Unicode letters, digits, white-space, '_', '.', '/', '=', '+', '-' (Java regex: " <code>^([\u00p{L}\u00p{Z}\u00p{N}_:/=-]+\u00p{-})*\$</code> "). Keys must be unique to a tag set. For example, you cannot have a key-pair in a tag set with the key the same but with different values, such as <code>project/Trinity</code> and <code>project/Xanadu</code> .
Value	A value is the optional value of the tag. The string value can be from 1 to 256 Unicode characters in length and cannot be prefixed with <code>aws:</code> or <code>rds:</code> . The string can only contain only the set of Unicode letters, digits, white-space, '_', '.', '/', '=', '+', '-' (Java regex: " <code>^([\u00p{L}\u00p{Z}\u00p{N}_:/=-]+\u00p{-})*\$</code> "). Values do not have to be unique in a tag set and can be null. For example, you can have a key-value pair in a tag set of <code>project/Trinity</code> and <code>cost-center/Trinity</code> .

Using the AWS Tag Editor

You can browse and edit the tags on your RDS resources in the AWS Management Console by using the AWS Tag editor. For more information, see [Tag Editor](#) in the *AWS Resource Groups User Guide*.

Copying tags to DB cluster snapshots

When you create or restore a DB cluster, you can specify that the tags from the DB cluster are copied to snapshots of the DB cluster. Copying tags ensures that the metadata for the DB snapshots matches that of the source DB cluster and any access policies for the DB snapshot also match those of the source DB cluster. Tags are not copied by default.

You can specify that tags are copied to DB snapshots for the following actions:

- Creating a DB cluster.
- Restoring a DB cluster.
- Creating a read replica.

- Copying a DB cluster snapshot.

Note

If you include a value for the `--tag-key` parameter of the `create-db-cluster-snapshot` AWS CLI command (or supply at least one tag to the `CreateDBClusterSnapshot` API operation) then RDS doesn't copy tags from the source DB cluster to the new DB snapshot. This functionality applies even if the source DB cluster has the `--copy-tags-to-snapshot` (`CopyTagsToSnapshot`) option enabled. If you take this approach, you can create a copy of a DB cluster from a DB cluster snapshot and avoid adding tags that don't apply to the new DB cluster. Once you have created your DB cluster snapshot using the AWS CLI `create-db-cluster-snapshot` command (or the `CreateDBSnapshot` Amazon RDS API operation) you can then add tags as described later in this topic.

Tutorial: Use tags to specify which Aurora DB clusters to stop

Suppose that you're creating a number of Aurora DB clusters in a development or test environment. You need to keep all of these clusters for several days. Some of the clusters run tests overnight. Other clusters can be stopped overnight and started again the next day. The following example shows how to assign a tag to those clusters that are suitable to stop overnight. Then the example shows how a script can detect which clusters have that tag and then stop those clusters. In this example, the value portion of the key-value pair doesn't matter. The presence of the `stopable` tag signifies that the cluster has this user-defined property.

To specify which Aurora DB clusters to stop

1. Determine the ARN of a cluster that you want to designate as stoppable.

The commands and APIs for tagging work with ARNs. That way, they can work seamlessly across AWS Regions, AWS accounts, and different types of resources that might have identical short names. You can specify the ARN instead of the cluster ID in CLI commands that operate on clusters. Substitute the name of your own cluster for `dev-test-cluster`. In subsequent commands that use ARN parameters, substitute the ARN of your own cluster. The ARN includes your own AWS account ID and the name of the AWS Region where your cluster is located.

```
$ aws rds describe-db-clusters --db-cluster-identifier dev-test-cluster \
--query "[].{DBClusterArn:DBClusterArn}" --output text
arn:aws:rds:us-east-1:123456789:cluster:dev-test-cluster
```

2. Add the tag `stopable` to this cluster.

The name for this tag is chosen by you. Using a tag like this is an alternative to devising a naming convention that encodes all the relevant information in the name of the cluster, DB instance, and so on. Because this example treats the tag as an attribute that is either present or absent, it omits the `Value=` part of the `--tags` parameter.

```
$ aws rds add-tags-to-resource \
--resource-name arn:aws:rds:us-east-1:123456789:cluster:dev-test-cluster \
--tags Key=stopable
```

3. Confirm that the tag is present in the cluster.

These commands retrieve the tag information for the cluster in JSON format and in plain tab-separated text.

```
$ aws rds list-tags-for-resource \
```

```
--resource-name arn:aws:rds:us-east-1:123456789:cluster:dev-test-cluster
{
    "TagList": [
        {
            "Key": "stoppable",
            "Value": ""
        }
    ]
}
$ aws rds list-tags-for-resource \
--resource-name arn:aws:rds:us-east-1:123456789:cluster:dev-test-cluster --output
text
TAGLIST stoppable
```

- To stop all the clusters that are designated as stoppable, prepare a list of all your clusters. Loop through the list and check if each cluster is tagged with the relevant attribute.

This Linux example uses shell scripting to save the list of cluster ARNs to a temporary file and then perform CLI commands for each cluster.

```
$ aws rds describe-db-clusters --query "*[].[DBClusterArn]" --output text >/tmp/
cluster_arns.lst
$ for arn in $(cat /tmp/cluster_arns.lst)
do
    match=$(aws rds list-tags-for-resource --resource-name $arn --output text | grep
'TAGLIST\|tstopable')
    if [[ ! -z "$match" ]]
    then
        echo "Cluster $arn is tagged as stoppable. Stopping it now."
    # Note that you can specify the full ARN value as the parameter instead of the short ID
    'dev-test-cluster'.
        aws rds stop-db-cluster --db-cluster-identifier $arn
    fi
done

Cluster arn:aws:rds:us-east-1:123456789:cluster:dev-test-cluster is tagged as
stoppable. Stopping it now.
{
    "DBCluster": {
        "AllocatedStorage": 1,
        "AvailabilityZones": [
            "us-east-1e",
            "us-east-1c",
            "us-east-1d"
        ],
        "BackupRetentionPeriod": 1,
        "DBClusterIdentifier": "dev-test-cluster",
        ...
    }
}
```

You can run a script like this at the end of each day to make sure that nonessential clusters are stopped. You might also schedule a job using a utility such as cron to perform such a check each night, in case some clusters were left running by mistake. In that case, you might fine-tune the command that prepares the list of clusters to check. The following command produces a list of your clusters, but only the ones in available state. The script can ignore clusters that are already stopped, because they will have different status values such as stopped or stopping.

```
$ aws rds describe-db-clusters \
--query '*[].[DBClusterArn:DBClusterArn,Status:Status]|?Status == `available`|[]'.
{DBClusterArn:DBClusterArn} \
--output text
arn:aws:rds:us-east-1:123456789:cluster:cluster-2447
```

```
arn:aws:rds:us-east-1:123456789:cluster:cluster-3395
arn:aws:rds:us-east-1:123456789:cluster:dev-test-cluster
arn:aws:rds:us-east-1:123456789:cluster:pg2-cluster
```

Tip

After you're familiar with the general procedure of assigning tags and finding clusters that have those tags, you can use the same technique to reduce costs in other ways. For example, in this scenario with Aurora DB clusters used for development and testing, you might designate some clusters to be deleted at the end of each day, or to have only their reader DB instances deleted, or to have their DB instances changed to a small DB instance classes during times of expected low usage.

Working with Amazon Resource Names (ARNs) in Amazon RDS

Resources created in Amazon Web Services are each uniquely identified with an Amazon Resource Name (ARN). For certain Amazon RDS operations, you must uniquely identify an Amazon RDS resource by specifying its ARN. For example, when you create an RDS DB instance read replica, you must supply the ARN for the source DB instance.

Constructing an ARN for Amazon RDS

Resources created in Amazon Web Services are each uniquely identified with an Amazon Resource Name (ARN). You can construct an ARN for an Amazon RDS resource using the following syntax.

`arn:aws:rds:<region>:<account number>:<resourcetype>:<name>`

Region Name	Region	Endpoint	Protocol	
US East (Ohio)	us-east-2	rds.us-east-2.amazonaws.com rds-fips.us-east-2.amazonaws.com rds-fips.us-east-2.amazonaws.com	HTTPS HTTPS HTTPS	
US East (N. Virginia)	us-east-1	rds.us-east-1.amazonaws.com rds-fips.us-east-1.amazonaws.com rds-fips.us-east-1.amazonaws.com	HTTPS HTTPS HTTPS	
US West (N. California)	us-west-1	rds.us-west-1.amazonaws.com rds-fips.us-west-1.amazonaws.com rds-fips.us-west-1.amazonaws.com	HTTPS HTTPS HTTPS	
US West (Oregon)	us-west-2	rds.us-west-2.amazonaws.com rds-fips.us-west-2.amazonaws.com rds-fips.us-west-2.amazonaws.com	HTTPS HTTPS HTTPS	
Africa (Cape Town)	af-south-1	rds.af-south-1.amazonaws.com	HTTPS	
Asia Pacific (Hong Kong)	ap-east-1	rds.ap-east-1.amazonaws.com	HTTPS	
Asia Pacific (Jakarta)	ap-southeast-3	rds.ap-southeast-3.amazonaws.com	HTTPS	

Region Name	Region	Endpoint	Protocol	
Asia Pacific (Mumbai)	ap-south-1	rds.ap-south-1.amazonaws.com	HTTPS	
Asia Pacific (Osaka)	ap-northeast-3	rds.ap-northeast-3.amazonaws.com	HTTPS	
Asia Pacific (Seoul)	ap-northeast-2	rds.ap-northeast-2.amazonaws.com	HTTPS	
Asia Pacific (Singapore)	ap-southeast-1	rds.ap-southeast-1.amazonaws.com	HTTPS	
Asia Pacific (Sydney)	ap-southeast-2	rds.ap-southeast-2.amazonaws.com	HTTPS	
Asia Pacific (Tokyo)	ap-northeast-1	rds.ap-northeast-1.amazonaws.com	HTTPS	
Canada (Central)	ca-central-1	rds.ca-central-1.amazonaws.com	HTTPS	
		rds-fips.ca-central-1.amazonaws.com	HTTPS	
		rds-fips.ca-central-1.amazonaws.com	HTTPS	
Europe (Frankfurt)	eu-central-1	rds.eu-central-1.amazonaws.com	HTTPS	
Europe (Ireland)	eu-west-1	rds.eu-west-1.amazonaws.com	HTTPS	
Europe (London)	eu-west-2	rds.eu-west-2.amazonaws.com	HTTPS	
Europe (Milan)	eu-south-1	rds.eu-south-1.amazonaws.com	HTTPS	
Europe (Paris)	eu-west-3	rds.eu-west-3.amazonaws.com	HTTPS	
Europe (Stockholm)	eu-north-1	rds.eu-north-1.amazonaws.com	HTTPS	
Middle East (Bahrain)	me-south-1	rds.me-south-1.amazonaws.com	HTTPS	
South America (São Paulo)	sa-east-1	rds.sa-east-1.amazonaws.com	HTTPS	

Region Name	Region	Endpoint	Protocol	
AWS GovCloud (US-East)	us-gov-east-1	rds.us-gov-east-1.amazonaws.com	HTTPS	
AWS GovCloud (US-West)	us-gov-west-1	rds.us-gov-west-1.amazonaws.com	HTTPS	

The following table shows the format that you should use when constructing an ARN for a particular Amazon RDS resource type.

Resource type	ARN format
DB instance	<p>arn:aws:rds:<region>:<account>:db:<name></p> <p>For example:</p> <pre>arn:aws:rds:us-east-2:123456789012:db:my-mysql-instance-1</pre>
DB cluster	<p>arn:aws:rds:<region>:<account>:cluster:<name></p> <p>For example:</p> <pre>arn:aws:rds:us-east-2:123456789012:cluster:my-aurora-cluster-1</pre>
Event subscription	<p>arn:aws:rds:<region>:<account>:es:<name></p> <p>For example:</p> <pre>arn:aws:rds:us-east-2:123456789012:es:my-subscription</pre>
DB parameter group	<p>arn:aws:rds:<region>:<account>:pg:<name></p> <p>For example:</p> <pre>arn:aws:rds:us-east-2:123456789012:pg:my-param-enable-logs</pre>
DB cluster parameter group	<p>arn:aws:rds:<region>:<account>:cluster-pg:<name></p> <p>For example:</p> <pre>arn:aws:rds:us-east-2:123456789012:cluster-pg:my-cluster-param-timezone</pre>
Reserved DB instance	<p>arn:aws:rds:<region>:<account>:ri:<name></p> <p>For example:</p>

Resource type	ARN format
	<code>arn:aws:rds:us-east-2:123456789012:ri:my-reserved-postgresql</code>
DB security group	<p><code>arn:aws:rds:<region>:<account>:secgrp:<name></code></p> <p>For example:</p> <pre>arn:aws:rds:us-east-2:123456789012:secgrp:my-public</pre>
Automated DB snapshot	<p><code>arn:aws:rds:<region>:<account>:snapshot:rds:<name></code></p> <p>For example:</p> <pre>arn:aws:rds:us-east-2:123456789012:snapshot:rds:my-mysql-db-2019-07-22-07-23</pre>
Automated DB cluster snapshot	<p><code>arn:aws:rds:<region>:<account>:cluster-snapshot:rds:<name></code></p> <p>For example:</p> <pre>arn:aws:rds:us-east-2:123456789012:cluster-snapshot:rds:my-aurora-cluster-2019-07-22-16-16</pre>
Manual DB snapshot	<p><code>arn:aws:rds:<region>:<account>:snapshot:<name></code></p> <p>For example:</p> <pre>arn:aws:rds:us-east-2:123456789012:snapshot:my-mysql-db-snap</pre>
Manual DB cluster snapshot	<p><code>arn:aws:rds:<region>:<account>:cluster-snapshot:<name></code></p> <p>For example:</p> <pre>arn:aws:rds:us-east-2:123456789012:cluster-snapshot:my-aurora-cluster-snap</pre>
DB subnet group	<p><code>arn:aws:rds:<region>:<account>:subgrp:<name></code></p> <p>For example:</p> <pre>arn:aws:rds:us-east-2:123456789012:subgrp:my-subnet-10</pre>

Getting an existing ARN

You can get the ARN of an RDS resource by using the AWS Management Console, AWS Command Line Interface (AWS CLI), or RDS API.

Console

To get an ARN from the AWS Management Console, navigate to the resource you want an ARN for, and view the details for that resource. For example, you can get the ARN for a DB instance from the **Configuration** tab of the DB instance details, as shown following.

The screenshot shows the AWS Management Console interface for a DB instance named "oracle-instance1". The top navigation bar has tabs: Connectivity, Monitoring, Logs & events, and Configuration. The Configuration tab is highlighted with a red oval. Below the tabs, the page title is "Instance" and the section title is "Configuration". The configuration details listed are:

- DB instance id: oracle-instance1
- Engine version: 12.1.0.2.v14
- Storage type: General Purpose (SSD)
- IOPS: -
- Storage: 20 GiB
- DB name: ORCL
- License model: Bring Your Own License
- Character set: AL32UTF8
- Option groups: default:oracle-ee-12-1
- ARN: arn:aws:rds:us-west-2:XXXXXXXXXXXX:db:oracle-instance1
- Resource id: (redacted)

AWS CLI

To get an ARN from the AWS CLI for a particular RDS resource, you use the `describe` command for that resource. The following table shows each AWS CLI command, and the ARN property used with the command to get an ARN.

AWS CLI command	ARN property
describe-event-subscriptions	EventSubscriptionArn
describe-certificates	CertificateArn
describe-db-parameter-groups	DBParameterGroupArn
describe-db-cluster-parameter-groups	DBClusterParameterGroupArn
describe-db-instances	DBInstanceArn
describe-db-security-groups	DBSecurityGroupArn
describe-db-snapshots	DBSnapshotArn
describe-events	SourceArn
describe-reserved-db-instances	ReservedDBInstanceArn
describe-db-subnet-groups	DBSubnetGroupArn
describe-db-clusters	DBClusterArn
describe-db-cluster-snapshots	DBClusterSnapshotArn

For example, the following AWS CLI command gets the ARN for a DB instance.

Example

For Linux, macOS, or Unix:

```
aws rds describe-db-instances \
--db-instance-identifier DBInstanceIdentifier \
--region us-west-2 \
--query "[].{DBInstanceIdentifier:DBInstanceIdentifier,DBInstanceArn:DBInstanceArn}"
```

For Windows:

```
aws rds describe-db-instances ^
--db-instance-identifier DBInstanceIdentifier ^
--region us-west-2 ^
--query "[].{DBInstanceIdentifier:DBInstanceIdentifier,DBInstanceArn:DBInstanceArn}"
```

The output of that command is like the following:

```
[  
  {  
    "DBInstanceArn": "arn:aws:rds:us-west-2:account_id:db:instance_id",  
    "DBInstanceIdentifier": "instance_id"  
  }  
]
```

RDS API

To get an ARN for a particular RDS resource, you can call the following RDS API operations and use the ARN properties shown following.

RDS API operation	ARN property
DescribeEventSubscriptions	EventSubscriptionArn
DescribeCertificates	CertificateArn
DescribeDBParameterGroups	DBParameterGroupArn
DescribeDBClusterParameterGroups	DBClusterParameterGroupArn
DescribeDBInstances	DBInstanceArn
DescribeDBSecurityGroups	DBSecurityGroupArn
DescribeDBSnapshots	DBSnapshotArn
DescribeEvents	SourceArn
DescribeReservedDBInstances	ReservedDBInstanceArn
DescribeDBSubnetGroups	DBSubnetGroupArn
DescribeDBClusters	DBClusterArn
DescribeDBClusterSnapshots	DBClusterSnapshotArn

Amazon Aurora updates

Amazon Aurora releases updates regularly. Updates are applied to Amazon Aurora DB clusters during system maintenance windows. The timing when updates are applied depends on the region and maintenance window setting for the DB cluster, and also the type of update. Updates require a database restart, so you typically experience 20 to 30 seconds of downtime. After this downtime, you can resume using your DB cluster or clusters. You can view or change your maintenance window settings from the [AWS Management Console](#).

Note

The time required to reboot your DB instance depends on the crash recovery process, database activity at the time of reboot, and the behavior of your specific DB engine. To improve the reboot time, we recommend that you reduce database activity as much as possible during the reboot process. Reducing database activity reduces rollback activity for in-transit transactions.

Following, you can find information on general updates to Amazon Aurora. Some of the updates applied to Amazon Aurora are specific to a database engine supported by Aurora. For more information about database engine updates for Aurora, see the following table.

Database engine	Updates
Amazon Aurora MySQL	See Database engine updates for Amazon Aurora MySQL (p. 1082)
Amazon Aurora PostgreSQL	See Amazon Aurora PostgreSQL updates (p. 1597)

Identifying your Amazon Aurora version

Amazon Aurora includes certain features that are general to Aurora and available to all Aurora DB clusters. Aurora includes other features that are specific to a particular database engine that Aurora supports. These features are available only to those Aurora DB clusters that use that database engine, such as Aurora PostgreSQL.

An Aurora DB instance provides two version numbers, the Aurora version number and the Aurora database engine version number. Aurora version numbers use the following format.

```
<major version>.<minor version>.<patch version>
```

To get the Aurora version number from an Aurora DB instance using a particular database engine, use one of the following queries.

Database engine	Queries
Amazon Aurora MySQL	<pre>SELECT AURORA_VERSION();</pre>
Amazon Aurora PostgreSQL	<pre>SHOW @@aurora_version;</pre>

Backing up and restoring an Amazon Aurora DB cluster

This section shows how to back up and restore Amazon Aurora DB clusters.

Topics

- [Overview of backing up and restoring an Aurora DB cluster \(p. 491\)](#)
- [Understanding Aurora backup storage usage \(p. 494\)](#)
- [Creating a DB cluster snapshot \(p. 495\)](#)
- [Restoring from a DB cluster snapshot \(p. 497\)](#)
- [Copying a DB cluster snapshot \(p. 500\)](#)
- [Sharing a DB cluster snapshot \(p. 510\)](#)
- [Exporting DB snapshot data to Amazon S3 \(p. 518\)](#)
- [Restoring a DB cluster to a specified time \(p. 537\)](#)
- [Deleting a DB cluster snapshot \(p. 539\)](#)

Overview of backing up and restoring an Aurora DB cluster

In the following sections, you can find information about Aurora backups and how to restore your Aurora DB cluster using the AWS Management Console.

Tip

The Aurora high availability features and automatic backup capabilities help to keep your data safe without requiring extensive setup from you. Before you implement a backup strategy, learn about the ways that Aurora maintains multiple copies of your data and helps you to access them across multiple DB instances and AWS Regions. For details, see [High availability for Amazon Aurora \(p. 68\)](#).

Backups

Aurora backs up your cluster volume automatically and retains restore data for the length of the *backup retention period*. Aurora backups are continuous and incremental so you can quickly restore to any point within the backup retention period. No performance impact or interruption of database service occurs as backup data is being written. You can specify a backup retention period, from 1 to 35 days, when you create or modify a DB cluster. Aurora backups are stored in Amazon S3.

If you want to retain a backup beyond the backup retention period, you can also take a snapshot of the data in your cluster volume. Because Aurora retains incremental restore data for the entire backup retention period, you only need to create a snapshot for data that you want to retain beyond the backup retention period. You can create a new DB cluster from the snapshot.

Note

- For Amazon Aurora DB clusters, the default backup retention period is one day regardless of how the DB cluster is created.
- You can't disable automated backups on Aurora. The backup retention period for Aurora is managed by the DB cluster.

Your costs for backup storage depend upon the amount of Aurora backup and snapshot data you keep and how long you keep it. For information about the storage associated with Aurora backups and snapshots, see [Understanding Aurora backup storage usage \(p. 494\)](#). For pricing information about Aurora backup storage, see [Amazon RDS for Aurora pricing](#). After the Aurora cluster associated with a snapshot is deleted, storing that snapshot incurs the standard backup storage charges for Aurora.

Note

You can also use AWS Backup to manage backups of Amazon Aurora DB clusters. Backups managed by AWS Backup are considered manual DB cluster snapshots, but don't count toward the DB cluster snapshot quota for Aurora. Backups that were created with AWS Backup have names ending in `awsbackup:AWS-Backup-job-number`. For information about AWS Backup, see the [AWS Backup Developer Guide](#).

Backup window

Automated backups occur daily during the preferred backup window. If the backup requires more time than allotted to the backup window, the backup continues after the window ends, until it finishes. The backup window can't overlap with the weekly maintenance window for the DB cluster.

Aurora backups are continuous and incremental, but the backup window is used to create a daily system backup that is preserved within the backup retention period. You can copy it to preserve it outside of the retention period.

Note

When you create a DB cluster using the AWS Management Console, you can't specify a backup window. However, you can specify a backup window when you create a DB cluster using the AWS CLI or RDS API.

If you don't specify a preferred backup window when you create the DB cluster, Aurora assigns a default 30-minute backup window. This window is selected at random from an 8-hour block of time for each AWS Region. The following table lists the time blocks for each AWS Region from which the default backup windows are assigned.

Region Name	Region	Time Block
US East (Ohio)	us-east-2	03:00–11:00 UTC
US East (N. Virginia)	us-east-1	03:00–11:00 UTC
US West (N. California)	us-west-1	06:00–14:00 UTC
US West (Oregon)	us-west-2	06:00–14:00 UTC
Africa (Cape Town)	af-south-1	03:00–11:00 UTC
Asia Pacific (Hong Kong)	ap-east-1	06:00–14:00 UTC
Asia Pacific (Jakarta)	ap-southeast-3	08:00–16:00 UTC
Asia Pacific (Mumbai)	ap-south-1	16:30–00:30 UTC
Asia Pacific (Osaka)	ap-northeast-3	00:00–08:00 UTC
Asia Pacific (Seoul)	ap-northeast-2	13:00–21:00 UTC
Asia Pacific (Singapore)	ap-southeast-1	14:00–22:00 UTC
Asia Pacific (Sydney)	ap-southeast-2	12:00–20:00 UTC
Asia Pacific (Tokyo)	ap-northeast-1	13:00–21:00 UTC
Canada (Central)	ca-central-1	03:00–11:00 UTC
China (Beijing)	cn-north-1	06:00–14:00 UTC
China (Ningxia)	cn-northwest-1	06:00–14:00 UTC
Europe (Frankfurt)	eu-central-1	20:00–04:00 UTC
Europe (Ireland)	eu-west-1	22:00–06:00 UTC
Europe (London)	eu-west-2	22:00–06:00 UTC
Europe (Paris)	eu-west-3	07:29–14:29 UTC
Europe (Milan)	eu-south-1	02:00–10:00 UTC
Europe (Stockholm)	eu-north-1	23:00–07:00 UTC
Middle East (Bahrain)	me-south-1	06:00–14:00 UTC
South America (São Paulo)	sa-east-1	23:00–07:00 UTC

Region Name	Region	Time Block
AWS GovCloud (US-East)	us-gov-east-1	17:00–01:00 UTC
AWS GovCloud (US-West)	us-gov-west-1	06:00–14:00 UTC

Restoring data

You can recover your data by creating a new Aurora DB cluster from the backup data that Aurora retains, or from a DB cluster snapshot that you have saved. You can quickly restore a new copy of a DB cluster created from backup data to any point in time during your backup retention period. The continuous and incremental nature of Aurora backups during the backup retention period means you don't need to take frequent snapshots of your data to improve restore times.

To determine the latest or earliest restorable time for a DB cluster, look for the `Latest restore time` or `Earliest restorable time` values on the RDS console. For information about viewing these values, see [Viewing an Amazon Aurora DB cluster \(p. 547\)](#). The latest restorable time for a DB cluster is the most recent point at which you can restore your DB cluster, typically within 5 minutes of the current time. The earliest restorable time specifies how far back within the backup retention period that you can restore your cluster volume.

You can determine when the restore of a DB cluster is complete by checking the `Latest Restorable Time` and `Earliest Restorable Time` values. The `Latest Restorable Time` and `Earliest Restorable Time` values return `NULL` until the restore operation is complete. You can't request a backup or restore operation if `Latest Restorable Time` or `Earliest Restorable Time` returns `NULL`.

For information about restoring a DB cluster to a specified time, see [Restoring a DB cluster to a specified time \(p. 537\)](#).

Database cloning for Aurora

You can also use database cloning to clone the databases of your Aurora DB cluster to a new DB cluster, instead of restoring a DB cluster snapshot. The clone databases use only minimal additional space when first created. Data is copied only as data changes, either on the source databases or the clone databases. You can make multiple clones from the same DB cluster, or create additional clones even from other clones. For more information, see [Cloning a volume for an Aurora DB cluster \(p. 402\)](#).

Backtrack

Aurora MySQL now supports "rewinding" a DB cluster to a specific time, without restoring data from a backup. For more information, see [Backtracking an Aurora DB cluster \(p. 816\)](#).

Understanding Aurora backup storage usage

Aurora stores continuous backups (within the backup retention period) and snapshots in Aurora backup storage. To control your backup storage usage, you can reduce the backup retention interval, remove old manual snapshots when they are no longer needed, or both. For general information about Aurora backups, see [Backups \(p. 491\)](#). For pricing information about Aurora backup storage, see the [Amazon Aurora pricing](#) webpage.

To control your costs, you can monitor the amount of storage consumed by continuous backups and manual snapshots that persist beyond the retention period. Then you can reduce the backup retention interval and remove manual snapshots when they are no longer needed.

You can use the Amazon CloudWatch metrics `TotalBackupStorageBilled`, `SnapshotStorageUsed`, and `BackupRetentionPeriodStorageUsed` to review and monitor the amount of storage used by your Aurora backups, as follows:

- `BackupRetentionPeriodStorageUsed` represents the amount of backup storage used, in bytes, for storing continuous backups at the current time. This value depends on the size of the cluster volume and the amount of changes you make during the retention period. However, for billing purposes it doesn't exceed the cumulative cluster volume size during the retention period. For example, if your cluster's `VolumeBytesUsed` size is 107,374,182,400 bytes (100 GiB), and your retention period is two days, the maximum value for `BackupRetentionPeriodStorageUsed` is 214,748,364,800 bytes (100 GiB + 100 GiB).
- `SnapshotStorageUsed` represents the amount of backup storage used, in bytes, for storing manual snapshots beyond the backup retention period. Manual snapshots don't count against your snapshot backup storage while their creation timestamp is within the retention period. All automatic snapshots also don't count against your snapshot backup storage. The size of each snapshot is the size of the cluster volume at the time you take the snapshot. The `SnapshotStorageUsed` value depends on the number of snapshots you keep and the size of each snapshot. For example, suppose you have one manual snapshot outside the retention period, and the cluster's `VolumeBytesUsed` size was 100 GiB when that snapshot was taken. The amount of `SnapshotStorageUsed` is 107,374,182,400 bytes (100 GiB).
- `TotalBackupStorageBilled` represents the sum, in bytes, of `BackupRetentionPeriodStorageUsed` and `SnapshotStorageUsed`, minus an amount of free backup storage, which equals the size of the cluster volume for one day. The free backup storage is equal to the latest volume size. For example if your cluster's `VolumeBytesUsed` size is 100 GiB, your retention period is two days, and you have one manual snapshot outside the retention period, the `TotalBackupStorageBilled` is 214,748,364,800 bytes (200 GiB + 100 GiB - 100 GiB).
- These metrics are computed independently for each Aurora DB cluster.

You can monitor your Aurora clusters and build reports using CloudWatch metrics through the [CloudWatch console](#). For more information about how to use CloudWatch metrics, see [Availability of Aurora metrics in the Amazon RDS console \(p. 649\)](#).

The backtrack setting for an Aurora DB cluster doesn't affect the volume of backup data for that cluster. Amazon bills the storage for backtrack data separately. You can also find the backtrack pricing information on the [Amazon Aurora pricing](#) web page.

If you share a snapshot with another user, you are still the owner of that snapshot. The storage costs apply to the snapshot owner. If you delete a shared snapshot that you own, nobody can access it. To keep access to a shared snapshot owned by someone else, you can copy that snapshot. Doing so makes you the owner of the new snapshot. Any storage costs for the copied snapshot apply to your account.

Creating a DB cluster snapshot

Amazon RDS creates a storage volume snapshot of your DB cluster, backing up the entire DB cluster and not just individual databases. When you create a DB cluster snapshot, you need to identify which DB cluster you are going to back up, and then give your DB cluster snapshot a name so you can restore from it later. The amount of time it takes to create a DB cluster snapshot varies with the size of your databases. Because the snapshot includes the entire storage volume, the size of files, such as temporary files, also affects the amount of time it takes to create the snapshot.

Unlike automated backups, manual snapshots aren't subject to the backup retention period. Snapshots don't expire.

For very long-term backups, we recommend exporting snapshot data to Amazon S3. If the major version of your DB engine is no longer supported, you can't restore to that version from a snapshot. For more information, see [Exporting DB snapshot data to Amazon S3 \(p. 518\)](#).

You can create a DB cluster snapshot using the AWS Management Console, the AWS CLI, or the RDS API.

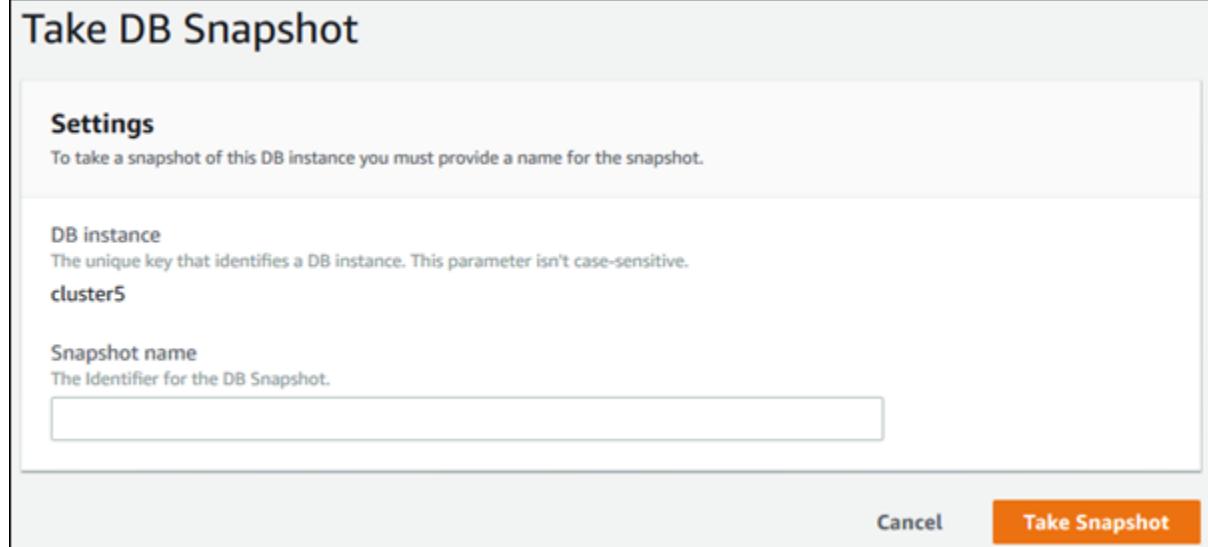
Console

To create a DB cluster snapshot

1. Sign in to the AWS Management Console and open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. In the navigation pane, choose **Databases**.
3. In the list of DB instances, choose a writer instance for the DB cluster.
4. Choose **Actions**, and then choose **Take snapshot**.

The **Take DB Snapshot** window appears.

5. Enter the name of the DB cluster snapshot in the **Snapshot name** box.



6. Choose **Take Snapshot**.

AWS CLI

When you create a DB cluster snapshot using the AWS CLI, you need to identify which DB cluster you are going to back up, and then give your DB cluster snapshot a name so you can restore from it later.

You can do this by using the AWS CLI [create-db-cluster-snapshot](#) command with the following parameters:

- `--db-cluster-identifier`
- `--db-cluster-snapshot-identifier`

In this example, you create a DB cluster snapshot named `mydbclustersnapshot` for a DB cluster called `mydbcluster`.

Example

For Linux, macOS, or Unix:

```
aws rds create-db-cluster-snapshot \
--db-cluster-identifier mydbcluster \
--db-cluster-snapshot-identifier mydbclustersnapshot
```

For Windows:

```
aws rds create-db-cluster-snapshot ^
--db-cluster-identifier mydbcluster ^
--db-cluster-snapshot-identifier mydbclustersnapshot
```

RDS API

When you create a DB cluster snapshot using the Amazon RDS API, you need to identify which DB cluster you are going to back up, and then give your DB cluster snapshot a name so you can restore from it later. You can do this by using the Amazon RDS API [CreateDBClusterSnapshot](#) command with the following parameters:

- `DBClusterIdentifier`
- `DBClusterSnapshotIdentifier`

Determining whether the DB cluster snapshot is available

You can check that the DB cluster snapshot is available by looking under **Snapshots** on the **Maintenance & backups** tab on the detail page for the cluster in the AWS Management Console, by using the [describe-db-cluster-snapshots](#) CLI command, or by using the [DescribeDBClusterSnapshots](#) API action.

You can also use the [wait db-cluster-snapshot-available](#) CLI command to poll the API every 30 seconds until the snapshot is available.

Restoring from a DB cluster snapshot

Amazon RDS creates a storage volume snapshot of your DB cluster, backing up the entire DB instance and not just individual databases. You can create a new DB cluster by restoring from a DB snapshot. You provide the name of the DB cluster snapshot to restore from, and then provide a name for the new DB cluster that is created from the restore. You can't restore from a DB cluster snapshot to an existing DB cluster; a new DB cluster is created when you restore.

You can use the restored DB cluster as soon as its status is available.

You can use AWS CloudFormation to restore a DB cluster from a DB cluster snapshot. For more information, see [AWS::RDS::DBCluster](#) in the *AWS CloudFormation User Guide*.

Note

Sharing a manual DB cluster snapshot, whether encrypted or unencrypted, enables authorized AWS accounts to directly restore a DB cluster from the snapshot instead of taking a copy of it and restoring from that. For more information, see [Sharing a DB cluster snapshot \(p. 510\)](#).

Parameter group considerations

We recommend that you retain the DB parameter group and DB cluster parameter group for any DB cluster snapshots you create, so that you can associate your restored DB cluster with the correct parameter groups.

The default DB parameter group and DB cluster parameter group are associated with the restored cluster, unless you choose different ones. No custom parameter settings are available in the default parameter groups.

You can specify the parameter groups when you restore the DB cluster.

For more information about DB parameter groups and DB cluster parameter groups, see [Working with DB parameter groups and DB cluster parameter groups \(p. 339\)](#).

Security group considerations

When you restore a DB cluster, the default virtual private cloud (VPC), DB subnet group, and VPC security group are associated with the restored instance, unless you choose different ones.

- If you're using the Amazon RDS console, you can specify a custom VPC security group to associate with the cluster or create a new VPC security group.
- If you're using the AWS CLI, you can specify a custom VPC security group to associate with the cluster by including the `--vpc-security-group-ids` option in the `restore-db-cluster-from-snapshot` command.
- If you're using the Amazon RDS API, you can include the `VpcSecurityGroupIds.VpcSecurityGroupId.N` parameter in the `RestoreDBClusterFromSnapshot` action.

As soon as the restore is complete and your new DB cluster is available, you can also change the VPC settings by modifying the DB cluster. For more information, see [Modifying an Amazon Aurora DB cluster \(p. 372\)](#).

Amazon Aurora considerations

With Aurora, you restore a DB cluster snapshot to a DB cluster.

With both Aurora MySQL and Aurora PostgreSQL, you can also restore a DB cluster snapshot to an Aurora Serverless DB cluster. For more information, see [Restoring an Aurora Serverless v1 DB cluster \(p. 166\)](#).

With Aurora MySQL, you can restore a DB cluster snapshot from a cluster without parallel query to a cluster with parallel query. Because parallel query is typically used with very large tables, the snapshot mechanism is the fastest way to ingest large volumes of data to an Aurora MySQL parallel query-enabled cluster. For more information, see [Working with parallel query for Amazon Aurora MySQL \(p. 881\)](#).

Restoring from a snapshot

You can restore a DB cluster from a DB cluster snapshot using the AWS Management Console, the AWS CLI, or the RDS API.

Console

To restore a DB cluster from a DB cluster snapshot

1. Sign in to the AWS Management Console and open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. In the navigation pane, choose **Snapshots**.
3. Choose the DB cluster snapshot that you want to restore from.
4. For **Actions**, choose **Restore snapshot**.
5. On the **Restore snapshot** page, for **DB instance identifier**, enter the name for your restored DB cluster.
6. Choose **Restore DB instance**.

AWS CLI

To restore a DB cluster from a DB cluster snapshot, use the AWS CLI command [restore-db-cluster-from-snapshot](#).

In this example, you restore from a previously created DB cluster snapshot named `mydbclustersnapshot`. You restore to a new DB cluster named `mynewdbcluster`.

Example

For Linux, macOS, or Unix:

```
aws rds restore-db-cluster-from-snapshot \
--db-cluster-identifier mynewdbcluster \
--snapshot-identifier mydbclustersnapshot \
--engine aurora/aurora-mysql/aurora-postgresql
```

For Windows:

```
aws rds restore-db-cluster-from-snapshot ^
--db-cluster-identifier mynewdbcluster ^
--snapshot-identifier mydbclustersnapshot ^
--engine aurora/aurora-mysql/aurora-postgresql
```

After the DB cluster has been restored, you must add the DB cluster to the security group used by the DB cluster used to create the DB cluster snapshot if you want the same functionality as that of the previous DB cluster.

Important

If you use the console to restore a DB cluster, then Amazon RDS automatically creates the primary instance (writer) for your DB cluster. If you use the AWS CLI to restore a DB cluster, you must explicitly create the primary instance for your DB cluster. The primary instance is the first instance that is created in a DB cluster. Call the [create-db-instance](#) AWS CLI command to create the primary instance for your DB cluster. Include the name of the DB cluster as the --db-cluster-identifier option value.

RDS API

To restore a DB cluster from a DB cluster snapshot, call the RDS API operation [RestoreDBClusterFromSnapshot](#) with the following parameters:

- `DBClusterIdentifier`
- `SnapshotIdentifier`

Important

If you use the console to restore a DB cluster, then Amazon RDS automatically creates the primary instance (writer) for your DB cluster. If you use the RDS API to restore a DB cluster, you must explicitly create the primary instance for your DB cluster. The primary instance is the first instance that is created in a DB cluster. Call the RDS API operation [CreateDBInstance](#) to create the primary instance for your DB cluster. Include the name of the DB cluster as the `DBClusterIdentifier` parameter value.

Copying a DB cluster snapshot

With Amazon RDS, you can copy automated backups or manual DB cluster snapshots. After you copy a snapshot, the copy is a manual snapshot. You can make multiple copies of an automated backup or manual snapshot, but each copy must have a unique identifier.

You can copy a snapshot within the same AWS Region, you can copy a snapshot across AWS Regions, and you can copy shared snapshots.

You can't copy a DB cluster snapshot across Regions and accounts in a single step. Perform one step for each of these copy actions. As an alternative to copying, you can also share manual snapshots with other AWS accounts. For more information, see [Sharing a DB cluster snapshot \(p. 510\)](#).

Note

Amazon bills you based upon the amount of Amazon Aurora backup and snapshot data you keep and the period of time that you keep it. For information about the storage associated with Aurora backups and snapshots, see [Understanding Aurora backup storage usage \(p. 494\)](#). For pricing information about Aurora storage, see [Amazon RDS for Aurora pricing](#).

Limitations

The following are some limitations when you copy snapshots:

- You can't copy a snapshot to or from the China (Beijing) or China (Ningxia) Regions.
- You can copy a snapshot between AWS GovCloud (US-East) and AWS GovCloud (US-West). However, you can't copy a snapshot between these AWS GovCloud (US) Regions and commercial AWS Regions.
- If you delete a source snapshot before the target snapshot becomes available, the snapshot copy might fail. Verify that the target snapshot has a status of `AVAILABLE` before you delete a source snapshot.
- You can have up to five snapshot copy requests in progress to a single destination Region per account.
- Depending on the AWS Regions involved and the amount of data to be copied, a cross-Region snapshot copy can take hours to complete. In some cases, there might be a large number of cross-Region snapshot copy requests from a given source Region. In such cases, Amazon RDS might put new cross-Region copy requests from that source Region into a queue until some in-progress copies complete. No progress information is displayed about copy requests while they are in the queue. Progress information is displayed when the copy starts.

Snapshot retention

Amazon RDS deletes automated backups in several situations:

- At the end of their retention period.
- When you disable automated backups for a DB cluster.
- When you delete a DB cluster.

If you want to keep an automated backup for a longer period, copy it to create a manual snapshot, which is retained until you delete it. Amazon RDS storage costs might apply to manual snapshots if they exceed your default storage space.

For more information about backup storage costs, see [Amazon RDS pricing](#).

Copying shared snapshots

You can copy snapshots shared to you by other AWS accounts. In some cases, you might copy an encrypted snapshot that has been shared from another AWS account. In these cases, you must have access to the AWS KMS key that was used to encrypt the snapshot.

You can only copy a shared DB cluster snapshot, whether encrypted or not, in the same AWS Region. For more information, see [Sharing encrypted snapshots \(p. 511\)](#).

Handling encryption

You can copy a snapshot that has been encrypted using a KMS key. If you copy an encrypted snapshot, the copy of the snapshot must also be encrypted. If you copy an encrypted snapshot within the same AWS Region, you can encrypt the copy with the same KMS key as the original snapshot. Or you can specify a different KMS key.

If you copy an encrypted snapshot across Regions, you must specify a KMS key valid in the destination AWS Region. It can be a Region-specific KMS key, or a multi-Region key. For more information on multi-Region KMS keys, see [Using multi-Region keys in AWS KMS](#).

The source snapshot remains encrypted throughout the copy process. For more information, see [Limitations of Amazon Aurora encrypted DB clusters \(p. 1712\)](#).

Note

For Amazon Aurora DB cluster snapshots, you can't encrypt an unencrypted DB cluster snapshot when you copy the snapshot.

Incremental snapshot copying

Aurora doesn't support incremental snapshot copying. Aurora DB cluster snapshot copies are always full copies. A full snapshot copy contains all of the data and metadata required to restore the DB cluster.

Cross-Region snapshot copying

You can copy DB cluster snapshots across AWS Regions. However, there are certain constraints and considerations for cross-Region snapshot copying.

Cross-Region copying of DB cluster snapshots isn't supported in the following opt-in AWS Regions:

- Africa (Cape Town)
- Asia Pacific (Hong Kong)
- Europe (Milan)
- Middle East (Bahrain)

Depending on the AWS Regions involved and the amount of data to be copied, a cross-Region snapshot copy can take hours to complete.

In some cases, there might be a large number of cross-Region snapshot copy requests from a given source AWS Region. In such cases, Amazon RDS might put new cross-Region copy requests from that source AWS Region into a queue until some in-progress copies complete. No progress information is displayed about copy requests while they are in the queue. Progress information is displayed when the copying starts.

Parameter group considerations

When you copy a snapshot across Regions, the copy doesn't include the parameter group used by the original DB cluster. When you restore a snapshot to create a new DB cluster, that DB cluster gets the default parameter group for the AWS Region it is created in. To give the new DB cluster the same parameters as the original, do the following:

1. In the destination AWS Region, create a DB cluster parameter group with the same settings as the original DB cluster. If one already exists in the new AWS Region, you can use that one.
2. After you restore the snapshot in the destination AWS Region, modify the new DB cluster and add the new or existing parameter group from the previous step.

Copying a DB cluster snapshot

Use the procedures in this topic to copy a DB cluster snapshot. If your source database engine is Aurora, then your snapshot is a DB cluster snapshot.

For each AWS account, you can copy up to five DB cluster snapshots at a time from one AWS Region to another. Copying both encrypted and unencrypted DB cluster snapshots is supported. If you copy a DB cluster snapshot to another AWS Region, you create a manual DB cluster snapshot that is retained in that AWS Region. Copying a DB cluster snapshot out of the source AWS Region incurs Amazon RDS data transfer charges.

For more information about data transfer pricing, see [Amazon RDS pricing](#).

After the DB cluster snapshot copy has been created in the new AWS Region, the DB cluster snapshot copy behaves the same as all other DB cluster snapshots in that AWS Region.

Console

This procedure works for copying encrypted or unencrypted DB cluster snapshots, in the same AWS Region or across Regions.

To cancel a copy operation once it is in progress, delete the target DB cluster snapshot while that DB cluster snapshot is in **copying** status.

To copy a DB cluster snapshot

1. Sign in to the AWS Management Console and open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. In the navigation pane, choose **Snapshots**.
3. Select the check box for the DB cluster snapshot you want to copy.
4. For **Actions**, choose **Copy Snapshot**. The **Make Copy of DB Snapshot** page appears.

Make Copy of DB Snapshot?

Settings

Source DB Snapshot

DB Snapshot Identifier for the automated snapshot being copied.
rds:cluster-1-2020-04-09-00-57

Destination Region [Info](#)

EU (Frankfurt) ▾

New DB Snapshot Identifier

DB Snapshot Identifier for the new snapshot

Copy Tags [Info](#)

i Please note that depending on the amount of data to be copied and the Region you choose, this operation could take several hours to complete and the display on the progress bar could be delayed until setup is complete.

Encryption

Encryption [Info](#)

Enable encryption [Learn more](#) ↗

Select to encrypt the given instance. Master key ids and aliases appear in the list after they have been created using the Key Management Service(KMS) console.

Disable encryption

[Cancel](#)

[Copy Snapshot](#)

5. (Optional) To copy the DB cluster snapshot to a different AWS Region, choose that AWS Region for **Destination Region**.
6. Type the name of the DB cluster snapshot copy in **New DB Snapshot Identifier**.
7. To copy tags and values from the snapshot to the copy of the snapshot, choose **Copy Tags**.
8. Choose **Copy Snapshot**.

Copying an unencrypted DB cluster snapshot by using the AWS CLI or Amazon RDS API

Use the procedures in the following sections to copy an unencrypted DB cluster snapshot by using the AWS CLI or Amazon RDS API.

To cancel a copy operation once it is in progress, delete the target DB cluster snapshot identified by `--target-db-cluster-snapshot-identifier` or `TargetDBClusterSnapshotIdentifier` while that DB cluster snapshot is in **copying** status.

AWS CLI

To copy a DB cluster snapshot, use the AWS CLI [copy-db-cluster-snapshot](#) command. If you are copying the snapshot to another AWS Region, run the command in the AWS Region to which the snapshot will be copied.

The following options are used to copy an unencrypted DB cluster snapshot:

- **--source-db-cluster-snapshot-identifier** – The identifier for the DB cluster snapshot to be copied. If you are copying the snapshot to another AWS Region, this identifier must be in the ARN format for the source AWS Region.
- **--target-db-cluster-snapshot-identifier** – The identifier for the new copy of the DB cluster snapshot.

The following code creates a copy of DB cluster snapshot `arn:aws:rds:us-east-1:123456789012:cluster-snapshot:aurora-cluster1-snapshot-20130805` named `myclustersnapshotcopy` in the AWS Region in which the command is run. When the copy is made, all tags on the original snapshot are copied to the snapshot copy.

Example

For Linux, macOS, or Unix:

```
aws rds copy-db-cluster-snapshot \
--source-db-cluster-snapshot-identifier arn:aws:rds:us-east-1:123456789012:cluster-
snapshot:aurora-cluster1-snapshot-20130805 \
--target-db-cluster-snapshot-identifier myclustersnapshotcopy \
--copy-tags
```

For Windows:

```
aws rds copy-db-cluster-snapshot ^
--source-db-cluster-snapshot-identifier arn:aws:rds:us-east-1:123456789012:cluster-
snapshot:aurora-cluster1-snapshot-20130805 ^
--target-db-cluster-snapshot-identifier myclustersnapshotcopy ^
--copy-tags
```

RDS API

To copy a DB cluster snapshot, use the Amazon RDS API [CopyDBClusterSnapshot](#) operation. If you are copying the snapshot to another AWS Region, perform the action in the AWS Region to which the snapshot will be copied.

The following parameters are used to copy an unencrypted DB cluster snapshot:

- **SourceDBClusterSnapshotIdentifier** – The identifier for the DB cluster snapshot to be copied. If you are copying the snapshot to another AWS Region, this identifier must be in the ARN format for the source AWS Region.
- **TargetDBClusterSnapshotIdentifier** – The identifier for the new copy of the DB cluster snapshot.

The following code creates a copy of a snapshot `arn:aws:rds:us-east-1:123456789012:cluster-snapshot:aurora-cluster1-snapshot-20130805` named `myclustersnapshotcopy` in the US West (N. California) Region. When the copy is made, all tags on the original snapshot are copied to the snapshot copy.

Example

```
https://rds.us-west-1.amazonaws.com/
?Action=CopyDBClusterSnapshot
&CopyTags=true
&SignatureMethod=HmacSHA256
&SignatureVersion=4
&SourceDBSnapshotIdentifier=arn%3Aaws%3Ards%3Aus-east-1%3A123456789012%3Acluster-
snapshot%3Aaurora-cluster1-snapshot-20130805
&TargetDBSnapshotIdentifier=myclustersnapshotcopy
&Version=2013-09-09
&X-Amz-Algorithm=AWS4-HMAC-SHA256
&X-Amz-Credential=AKIADQKE4SARGYLE/20140429/us-west-1/rds/aws4_request
&X-Amz-Date=20140429T175351Z
&X-Amz-SignedHeaders=content-type;host;user-agent;x-amz-content-sha256;x-amz-date
&X-Amz-Signature=9164337efa99caf850e874a1cb7ef62f3cea29d0b448b9e0e7c53b288ddffed2
```

Copying an encrypted DB cluster snapshot by using the AWS CLI or Amazon RDS API

Use the procedures in the following sections to copy an encrypted DB cluster snapshot by using the AWS CLI or Amazon RDS API.

To cancel a copy operation once it is in progress, delete the target DB cluster snapshot identified by `--target-db-cluster-snapshot-identifier` or `TargetDBClusterSnapshotIdentifier` while that DB cluster snapshot is in **copying** status.

AWS CLI

To copy a DB cluster snapshot, use the AWS CLI `copy-db-cluster-snapshot` command. If you are copying the snapshot to another AWS Region, run the command in the AWS Region to which the snapshot will be copied.

The following options are used to copy an encrypted DB cluster snapshot:

- `--source-region` – If you are copying the snapshot to another AWS Region, specify the AWS Region that the encrypted DB cluster snapshot will be copied from.

If you are copying the snapshot to another AWS Region and you don't specify `source-region`, you must specify the `pre-signed-url` option instead. The `pre-signed-url` value must be a URL that contains a Signature Version 4 signed request for the `CopyDBClusterSnapshot` action to be called in the source AWS Region where the DB cluster snapshot is copied from. To learn more about the `pre-signed-url`, see `copy-db-cluster-snapshot`.

- `--source-db-cluster-snapshot-identifier` – The identifier for the encrypted DB cluster snapshot to be copied. If you are copying the snapshot to another AWS Region, this identifier must be in the ARN format for the source AWS Region. If that is the case, the AWS Region specified in `source-db-cluster-snapshot-identifier` must match the AWS Region specified for `--source-region`.
- `--target-db-cluster-snapshot-identifier` – The identifier for the new copy of the encrypted DB cluster snapshot.
- `--kms-key-id` – The KMS key identifier for the key to use to encrypt the copy of the DB cluster snapshot.

You can optionally use this option if the DB cluster snapshot is encrypted, you copy the snapshot in the same AWS Region, and you want to specify a new KMS key to encrypt the copy. Otherwise, the copy of the DB cluster snapshot is encrypted with the same KMS key as the source DB cluster snapshot.

You must use this option if the DB cluster snapshot is encrypted and you are copying the snapshot to another AWS Region. In that case, you must specify a KMS key for the destination AWS Region.

The following code example copies the encrypted DB cluster snapshot from the US West (Oregon) Region to the US East (N. Virginia) Region. The command is called in the US East (N. Virginia) Region.

Example

For Linux, macOS, or Unix:

```
aws rds copy-db-cluster-snapshot \
--source-db-cluster-snapshot-identifier arn:aws:rds:us-west-2:123456789012:cluster-
snapshot:aurora-cluster1-snapshot-20161115 \
--target-db-cluster-snapshot-identifier myclustersnapshotcopy \
--source-region us-west-2 \
--kms-key-id my-us-east-1-key
```

For Windows:

```
aws rds copy-db-cluster-snapshot ^
--source-db-cluster-snapshot-identifier arn:aws:rds:us-west-2:123456789012:cluster-
snapshot:aurora-cluster1-snapshot-20161115 ^
--target-db-cluster-snapshot-identifier myclustersnapshotcopy ^
--source-region us-west-2 ^
--kms-key-id my-us-east-1-key
```

RDS API

To copy a DB cluster snapshot, use the Amazon RDS API [CopyDBClusterSnapshot](#) operation. If you are copying the snapshot to another AWS Region, perform the action in the AWS Region to which the snapshot will be copied.

The following parameters are used to copy an encrypted DB cluster snapshot:

- **SourceDBClusterSnapshotIdentifier** – The identifier for the encrypted DB cluster snapshot to be copied. If you are copying the snapshot to another AWS Region, this identifier must be in the ARN format for the source AWS Region.
- **TargetDBClusterSnapshotIdentifier** – The identifier for the new copy of the encrypted DB cluster snapshot.
- **KmsKeyId** – The KMS key identifier for the key to use to encrypt the copy of the DB cluster snapshot.

You can optionally use this parameter if the DB cluster snapshot is encrypted, you copy the snapshot in the same AWS Region, and you specify a new KMS key to use to encrypt the copy. Otherwise, the copy of the DB cluster snapshot is encrypted with the same KMS key as the source DB cluster snapshot.

You must use this parameter if the DB cluster snapshot is encrypted and you are copying the snapshot to another AWS Region. In that case, you must specify a KMS key for the destination AWS Region.

- **PreSignedUrl** – If you are copying the snapshot to another AWS Region, you must specify the **PreSignedUrl** parameter. The **PreSignedUrl** value must be a URL that contains a Signature Version 4 signed request for the `CopyDBClusterSnapshot` action to be called in the source AWS Region where the DB cluster snapshot is copied from. To learn more about using a presigned URL, see [CopyDBClusterSnapshot](#).

To automatically rather than manually generate a presigned URL, use the AWS CLI `copy-db-cluster-snapshot` command with the `--source-region` option instead.

The following code example copies the encrypted DB cluster snapshot from the US West (Oregon) Region to the US East (N. Virginia) Region. The action is called in the US East (N. Virginia) Region.

Example

```
https://rds.us-east-1.amazonaws.com/
?Action=CopyDBClusterSnapshot
&KmsKeyId=my-us-east-1-key
&PreSignedUrl=https%253A%252F%252Frds.us-west-2.amazonaws.com%252F
%253FAction%253DCopyDBClusterSnapshot
%2526DestinationRegion%253Dus-east-1
%2526KmsKeyId%253Dmy-us-east-1-key
%2526SourceDBClusterSnapshotIdentifier%253Darn%25253Aaws%25253Ards%25253Aus-
west-2%25253A123456789012%25253Acluster-snapshot%25253Aaurora-cluster1-snapshot-20161115
%2526SignatureMethod%253DHmacSHA256
%2526SignatureVersion%253D4
%2526Version%253D2014-10-31
%2526X-Amz-Algorithm%253DAWS4-HMAC-SHA256
%2526X-Amz-Credential%253DAKIADQKE4SARGYLE%252F20161117%252Fus-west-2%252Frds
%252Faws4_request
%2526X-Amz-Date%253D20161117T215409Z
%2526X-Amz-Expires%253D3600
%2526X-Amz-SignedHeaders%253Dcontent-type%253Bhost%253Buser-agent%253Bx-amz-
content-sha256%253Bx-amz-date
%2526X-Amz-Signature
%253D255a0f17b4e717d3b67fad163c3ec26573b882c03a65523522cf890a67fca613
&SignatureMethod=HmacSHA256
&SignatureVersion=4
&SourceDBClusterSnapshotIdentifier=arn%3Aaws%3Ards%3Aus-
west-2%3A123456789012%3Acluster-snapshot%3Aaurora-cluster1-snapshot-20161115
&TargetDBClusterSnapshotIdentifier=myclustersnapshotcopy
&Version=2014-10-31
&X-Amz-Algorithm=AWS4-HMAC-SHA256
&X-Amz-Credential=AKIADQKE4SARGYLE/20161117/us-east-1/rds/aws4_request
&X-Amz-Date=20161117T221704Z
&X-Amz-SignedHeaders=content-type;host;user-agent;x-amz-content-sha256;x-amz-date
&X-Amz-Signature=da4f2da66739d2e722c85fcfd225dc27bba7e2b8dbea8d8612434378e52adccf
```

Copying a DB cluster snapshot across accounts

You can enable other AWS accounts to copy DB cluster snapshots that you specify by using the Amazon RDS API `ModifyDBClusterSnapshotAttribute` and `CopyDBClusterSnapshot` actions. You can only copy DB cluster snapshots across accounts in the same AWS Region. The cross-account copying process works as follows, where Account A is making the snapshot available to copy, and Account B is copying it.

1. Using Account A, call `ModifyDBClusterSnapshotAttribute`, specifying `restore` for the `AttributeName` parameter, and the ID for Account B for the `ValuesToAdd` parameter.
2. (If the snapshot is encrypted) Using Account A, update the key policy for the KMS key, first adding the ARN of Account B as a `Principal`, and then allow the `kms:CreateGrant` action.
3. (If the snapshot is encrypted) Using Account B, choose or create an IAM user and attach an IAM policy to that user that allows it to copy an encrypted DB cluster snapshot using your KMS key.
4. Using Account B, call `CopyDBClusterSnapshot` and use the `SourceDBClusterSnapshotIdentifier` parameter to specify the ARN of the DB cluster snapshot to be copied, which must include the ID for Account A.

To list all of the AWS accounts permitted to restore a DB cluster snapshot, use the `DescribeDBSnapshotAttributes` or `DescribeDBClusterSnapshotAttributes` API operation.

To remove sharing permission for an AWS account, use the `ModifyDBSnapshotAttribute` or `ModifyDBClusterSnapshotAttribute` action with `AttributeName` set to `restore` and the ID of the account to remove in the `ValuesToRemove` parameter.

Copying an unencrypted DB cluster snapshot to another account

Use the following procedure to copy an unencrypted DB cluster snapshot to another account in the same AWS Region.

1. In the source account for the DB cluster snapshot, call `ModifyDBClusterSnapshotAttribute`, specifying `restore` for the `AttributeName` parameter, and the ID for the target account for the `ValuesToAdd` parameter.

Running the following example using the account 987654321 permits two AWS account identifiers, 123451234512 and 123456789012, to restore the DB cluster snapshot named manual-snapshot1.

```
https://rds.us-west-2.amazonaws.com/
?Action=ModifyDBClusterSnapshotAttribute
&AttributeName=restore
&DBClusterSnapshotIdentifier=manual-snapshot1
&SignatureMethod=HmacSHA256&SignatureVersion=4
&ValuesToAdd.member.1=123451234512
&ValuesToAdd.member.2=123456789012
&Version=2014-10-31
&X-Amz-Algorithm=AWS4-HMAC-SHA256
&X-Amz-Credential=AKIADQKE4SARGYLE/20150922/us-west-2/rds/aws4_request
&X-Amz-Date=20150922T220515Z
&X-Amz-SignedHeaders=content-type;host;user-agent;x-amz-content-sha256;x-amz-date
&X-Amz-Signature=ef38f1ce3dab4e1dbf113d8d2a265c67d17ece1999ffd36be85714ed36ddbb3
```

2. In the target account, call `CopyDBClusterSnapshot` and use the `SourceDBClusterSnapshotIdentifier` parameter to specify the ARN of the DB cluster snapshot to be copied, which must include the ID for the source account.

Running the following example using the account 123451234512 copies the DB cluster snapshot aurora-cluster1-snapshot-20130805 from account 987654321 and creates a DB cluster snapshot named dbclustersnapshot1.

```
https://rds.us-west-2.amazonaws.com/
?Action=CopyDBClusterSnapshot
&CopyTags=true
&SignatureMethod=HmacSHA256
&SignatureVersion=4
&SourceDBClusterSnapshotIdentifier=arn:aws:rds:us-west-2:987654321:cluster-
snapshot:aurora-cluster1-snapshot-20130805
&TargetDBClusterSnapshotIdentifier=dbclustersnapshot1
&Version=2013-09-09
&X-Amz-Algorithm=AWS4-HMAC-SHA256
&X-Amz-Credential=AKIADQKE4SARGYLE/20150922/us-west-2/rds/aws4_request
&X-Amz-Date=20140429T175351Z
&X-Amz-SignedHeaders=content-type;host;user-agent;x-amz-content-sha256;x-amz-date
&X-Amz-Signature=9164337efa99caf850e874a1cb7ef62f3cea29d0b448b9e0e7c53b288ddffed2
```

Copying an encrypted DB cluster snapshot to another account

Use the following procedure to copy an encrypted DB cluster snapshot to another account in the same AWS Region.

1. In the source account for the DB cluster snapshot, call `ModifyDBClusterSnapshotAttribute`, specifying `restore` for the `AttributeName` parameter, and the ID for the target account for the `ValuesToAdd` parameter.

Running the following example using the account 987654321 permits two AWS account identifiers, 123451234512 and 123456789012, to restore the DB cluster snapshot named manual-snapshot1.

```
https://rds.us-west-2.amazonaws.com/
?Action=ModifyDBClusterSnapshotAttribute
&AttributeName=restore
&DBClusterSnapshotIdentifier>manual-snapshot1
&SignatureMethod=HmacSHA256&SignatureVersion=4
&ValuesToAdd.member.1=123451234512
&ValuesToAdd.member.2=123456789012
&Version=2014-10-31
&X-Amz-Algorithm=AWS4-HMAC-SHA256
&X-Amz-Credential=AKIADQKE4SARGYLE/20150922/us-west-2/rds/aws4_request
&X-Amz-Date=20150922T220515Z
&X-Amz-SignedHeaders=content-type;host;user-agent;x-amz-content-sha256;x-amz-date
&X-Amz-Signature=ef38f1ce3dab4e1dbf113d8d2a265c67d17ece1999ffd36be85714ed36ddbb3
```

2. In the source account for the DB cluster snapshot, update the key policy for the KMS key, first adding the ARN of the target account as a Principal, and then allow the `kms:CreateGrant` action. For more information, see [Allowing access to an AWS KMS key \(p. 511\)](#).
3. In the target account, choose or create an IAM user and attach an IAM policy to that user that allows it to copy an encrypted DB cluster snapshot using your KMS key. For more information, see [Creating an IAM policy to enable copying of the encrypted snapshot \(p. 512\)](#).
4. In the target account, call `CopyDBClusterSnapshot` and use the `SourceDBClusterSnapshotIdentifier` parameter to specify the ARN of the DB cluster snapshot to be copied, which must include the ID for the source account.

Running the following example using the account 123451234512 copies the DB cluster snapshot aurora-cluster1-snapshot-20130805 from account 987654321 and creates a DB cluster snapshot named dbclustersnapshot1.

```
https://rds.us-west-2.amazonaws.com/
?Action=CopyDBClusterSnapshot
&CopyTags=true
&SignatureMethod=HmacSHA256
&SignatureVersion=4
&SourceDBClusterSnapshotIdentifier=arn:aws:rds:us-west-2:987654321:cluster-
snapshot:aurora-cluster1-snapshot-20130805
&TargetDBClusterSnapshotIdentifier=dbclustersnapshot1
&Version=2013-09-09
&X-Amz-Algorithm=AWS4-HMAC-SHA256
&X-Amz-Credential=AKIADQKE4SARGYLE/20140429T175351Z
&X-Amz-Date=20140429T175351Z
&X-Amz-SignedHeaders=content-type;host;user-agent;x-amz-content-sha256;x-amz-date
&X-Amz-Signature=9164337efa99caf850e874a1cb7ef62f3cea29d0b448b9e0e7c53b288ddffed2
```

Sharing a DB cluster snapshot

Using Amazon RDS, you can share a manual DB cluster snapshot in the following ways:

- Sharing a manual DB cluster snapshot, whether encrypted or unencrypted, enables authorized AWS accounts to copy the snapshot.
- Sharing a manual DB cluster snapshot, whether encrypted or unencrypted, enables authorized AWS accounts to directly restore a DB cluster from the snapshot instead of taking a copy of it and restoring from that.

Note

To share an automated DB cluster snapshot, create a manual DB cluster snapshot by copying the automated snapshot, and then share that copy. This process also applies to AWS Backup-generated resources.

For more information on copying a snapshot, see [Copying a DB cluster snapshot \(p. 500\)](#). For more information on restoring a DB instance from a DB cluster snapshot, see [Restoring from a DB cluster snapshot \(p. 497\)](#).

For more information on restoring a DB cluster from a DB cluster snapshot, see [Overview of backing up and restoring an Aurora DB cluster \(p. 491\)](#).

You can share a manual snapshot with up to 20 other AWS accounts.

The following limitation applies when sharing manual snapshots with other AWS accounts:

- When you restore a DB cluster from a shared snapshot using the AWS Command Line Interface (AWS CLI) or Amazon RDS API, you must specify the Amazon Resource Name (ARN) of the shared snapshot as the snapshot identifier.

Sharing public snapshots

You can also share an unencrypted manual snapshot as public, which makes the snapshot available to all AWS accounts. Make sure when sharing a snapshot as public that none of your private information is included in the public snapshot.

When a snapshot is shared publicly, it gives all AWS accounts permission both to copy the snapshot and to create DB clusters from it.

You aren't billed for the backup storage of public snapshots owned by other accounts. You're billed only for snapshots that you own.

If you copy a public snapshot, you own the copy. You're billed for the backup storage of your snapshot copy. If you create a DB cluster from a public snapshot, you're billed for that DB cluster. For Amazon Aurora pricing information, see the [Aurora pricing page](#).

You can delete only the public snapshots that you own. To delete a shared or public snapshot, make sure to log into the AWS account that owns the snapshot.

Viewing public snapshots owned by other AWS accounts

You can view public snapshots owned by other accounts in a particular AWS Region on the **Public** tab of the **Snapshots** page in the Amazon RDS console. Your snapshots (those owned by your account) don't appear on this tab.

To view public snapshots

1. Open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. In the navigation pane, choose **Snapshots**.
3. Choose the **Public** tab.

The public snapshots appear. You can see which account owns a public snapshot in the **Owner** column.

Note

You might have to modify the page preferences, by selecting the gear icon at the upper right of the **Public snapshots** list, to see this column.

Viewing your own public snapshots

You can use the following AWS CLI command (Unix only) to view the public snapshots owned by your AWS account in a particular AWS Region.

```
aws rds describe-db-cluster-snapshots --snapshot-type public --include-public |  
grep account_number
```

The output returned is similar to the following example if you have public snapshots.

```
"DBClusterSnapshotArn": "arn:aws:rds:us-west-2:123456789012:cluster-  
snapshot:myclustersnapshot1",  
"DBClusterSnapshotArn": "arn:aws:rds:us-west-2:123456789012:cluster-  
snapshot:myclustersnapshot2",
```

Sharing encrypted snapshots

You can share DB cluster snapshots that have been encrypted "at rest" using the AES-256 encryption algorithm, as described in [Encrypting Amazon Aurora resources \(p. 1709\)](#). To do this, take the following steps:

1. Share the AWS KMS key that was used to encrypt the snapshot with any accounts that you want to be able to access the snapshot.

You can share KMS keys with another AWS account by adding the other account to the KMS key policy. For details on updating a key policy, see [Key policies](#) in the *AWS KMS Developer Guide*. For an example of creating a key policy, see [Allowing access to an AWS KMS key \(p. 511\)](#) later in this topic.
2. Use the AWS Management Console, AWS CLI, or Amazon RDS API to share the encrypted snapshot with the other accounts.

These restrictions apply to sharing encrypted snapshots:

- You can't share encrypted snapshots as public.
- You can't share a snapshot that has been encrypted using the default KMS key of the AWS account that shared the snapshot.

Allowing access to an AWS KMS key

For another AWS account to copy an encrypted DB cluster snapshot shared from your account, the account that you share your snapshot with must have access to the AWS KMS key that encrypted the snapshot.

To allow another AWS account access to a KMS key, update the key policy for the KMS key. You update it with the Amazon Resource Name (ARN) of the AWS account that you are sharing to as `Principal` in the KMS key policy. Then you allow the `kms:CreateGrant` action.

After you have given an AWS account access to your KMS key, to copy your encrypted snapshot that AWS account must create an AWS Identity and Access Management (IAM) role or user if it doesn't already have one. In addition, that AWS account must also attach an IAM policy to that IAM role or user that allows the role or user to copy an encrypted DB cluster snapshot using your KMS key. The account must be an IAM user and cannot be a root AWS account identity due to AWS KMS security restrictions.

In the following key policy example, user `111122223333` is the owner of the KMS key, and user `444455556666` is the account that the key is being shared with. This updated key policy gives the AWS account access to the KMS key by including the ARN for the root AWS account identity for user `444455556666` as a `Principal` for the policy, and by allowing the `kms:CreateGrant` action.

```
{
  "Id": "key-policy-1",
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "Allow use of the key",
      "Effect": "Allow",
      "Principal": {"AWS": [
        "arn:aws:iam::111122223333:user/KeyUser",
        "arn:aws:iam::444455556666:root"
      ]},
      "Action": [
        "kms:CreateGrant",
        "kms:Encrypt",
        "kms:Decrypt",
        "kms:ReEncrypt*",
        "kms:GenerateDataKey*",
        "kms:DescribeKey"
      ],
      "Resource": "*"
    },
    {
      "Sid": "Allow attachment of persistent resources",
      "Effect": "Allow",
      "Principal": {"AWS": [
        "arn:aws:iam::111122223333:user/KeyUser",
        "arn:aws:iam::444455556666:root"
      ]},
      "Action": [
        "kms:CreateGrant",
        "kms>ListGrants",
        "kms:RevokeGrant"
      ],
      "Resource": "*",
      "Condition": {"Bool": {"kms:GrantIsForAWSResource": true}}
    }
  ]
}
```

Creating an IAM policy to enable copying of the encrypted snapshot

Once the external AWS account has access to your KMS key, the owner of that AWS account can create a policy that allows an IAM user created for that account to copy an encrypted snapshot encrypted with that KMS key.

The following example shows a policy that can be attached to an IAM user for AWS account `444455556666` that enables the IAM user to copy a shared snapshot from AWS account `111122223333`

that has been encrypted with the KMS key `c989c1dd-a3f2-4a5d-8d96-e793d082ab26` in the `us-west-2` region.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "AllowUseOfTheKey",  
            "Effect": "Allow",  
            "Action": [  
                "kms:Encrypt",  
                "kms:Decrypt",  
                "kms:ReEncrypt*",  
                "kms:GenerateDataKey*",  
                "kms:DescribeKey",  
                "kms:CreateGrant",  
                "kms:RetireGrant"  
            ],  
            "Resource": ["arn:aws:kms:us-west-2:111122223333:key/c989c1dd-a3f2-4a5d-8d96-e793d082ab26"]  
        },  
        {  
            "Sid": "AllowAttachmentOfPersistentResources",  
            "Effect": "Allow",  
            "Action": [  
                "kms:CreateGrant",  
                "kms>ListGrants",  
                "kms:RevokeGrant"  
            ],  
            "Resource": ["arn:aws:kms:us-west-2:111122223333:key/c989c1dd-a3f2-4a5d-8d96-e793d082ab26"],  
            "Condition": {  
                "Bool": {  
                    "kms:GrantIsForAWSResource": true  
                }  
            }  
        }  
    ]  
}
```

For details on updating a key policy, see [Key policies in the AWS KMS Developer Guide](#).

Sharing a snapshot

You can share a DB cluster snapshot using the AWS Management Console, the AWS CLI, or the RDS API.

Console

Using the Amazon RDS console, you can share a manual DB cluster snapshot with up to 20 AWS accounts. You can also use the console to stop sharing a manual snapshot with one or more accounts.

To share a manual DB cluster snapshot by using the Amazon RDS console

1. Sign in to the AWS Management Console and open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. In the navigation pane, choose **Snapshots**.
3. Select the manual snapshot that you want to share.
4. For **Actions**, choose **Share Snapshot**.

5. Choose one of the following options for **DB snapshot visibility**.
 - If the source is unencrypted, choose **Public** to permit all AWS accounts to restore a DB cluster from your manual DB cluster snapshot, or choose **Private** to permit only AWS accounts that you specify to restore a DB cluster from your manual DB cluster snapshot.
6. For **AWS Account ID**, type the AWS account identifier for an account that you want to permit to restore a DB cluster from your manual snapshot, and then choose **Add**. Repeat to include additional AWS account identifiers, up to 20 AWS accounts.

If you make an error when adding an AWS account identifier to the list of permitted accounts, you can delete it from the list by choosing **Delete** at the right of the incorrect AWS account identifier.

Snapshot permissions

Preferences

You are sharing an unencrypted DB snapshot. When you share an unencrypted DB snapshot, you give the other account permission to make a copy of the DB snapshot and to restore a database from your DB snapshot.

DB snapshot
testoracletags-snap

DB snapshot visibility
 Private
 Public

AWS account ID

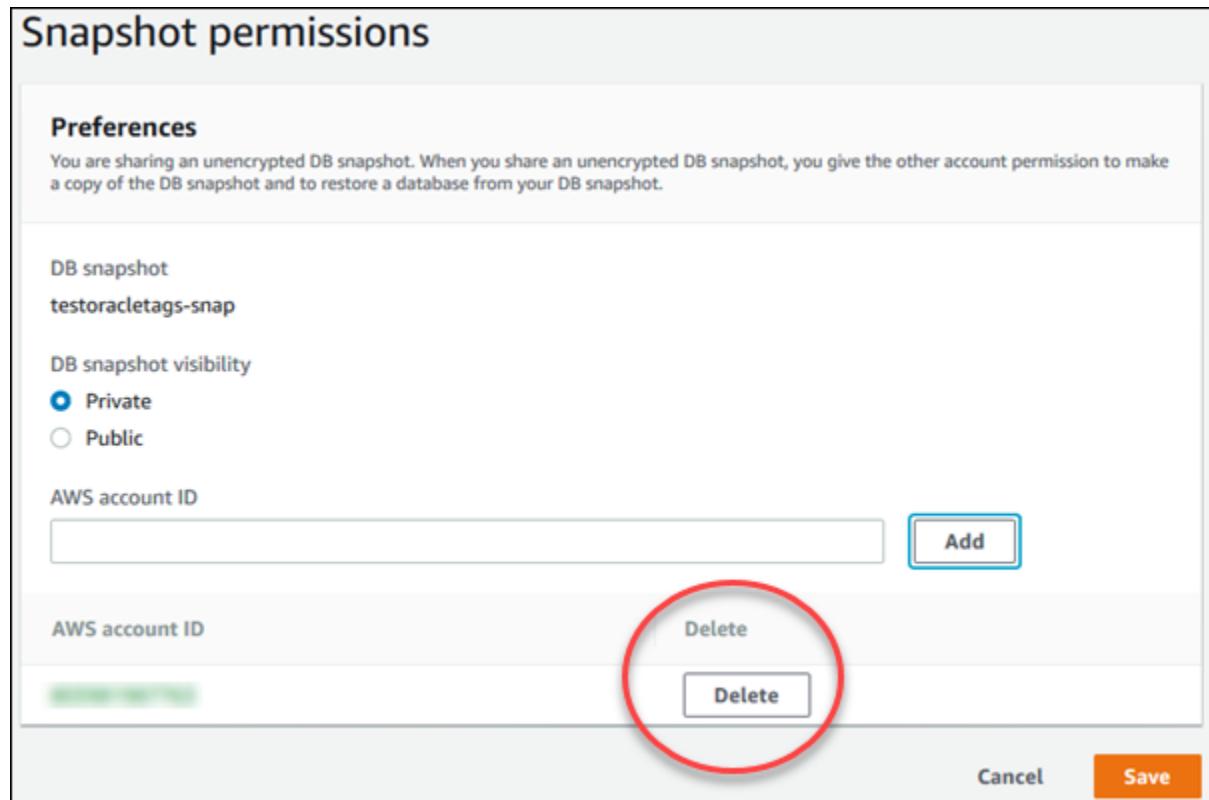
AWS account ID	Delete
Please add AWS account ID	

7. After you have added identifiers for all of the AWS accounts that you want to permit to restore the manual snapshot, choose **Save** to save your changes.

To stop sharing a manual DB cluster snapshot with an AWS account

1. Sign in to the AWS Management Console and open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. In the navigation pane, choose **Snapshots**.
3. Select the manual snapshot that you want to stop sharing.
4. Choose **Actions**, and then choose **Share Snapshot**.

- To remove permission for an AWS account, choose **Delete** for the AWS account identifier for that account from the list of authorized accounts.



- Choose **Save** to save your changes.

AWS CLI

To share a DB cluster snapshot, use the `aws rds modify-db-cluster-snapshot-attribute` command. Use the `--values-to-add` parameter to add a list of the IDs for the AWS accounts that are authorized to restore the manual snapshot.

Example of sharing a snapshot with a single account

The following example enables AWS account identifier 123456789012 to restore the DB cluster snapshot named `cluster-3-snapshot`.

For Linux, macOS, or Unix:

```
aws rds modify-db-cluster-snapshot-attribute \
--db-cluster-snapshot-identifier cluster-3-snapshot \
--attribute-name restore \
--values-to-add 123456789012
```

For Windows:

```
aws rds modify-db-cluster-snapshot-attribute ^
--db-cluster-snapshot-identifier cluster-3-snapshot ^
--attribute-name restore ^
--values-to-add 123456789012
```

Example of sharing a snapshot with multiple accounts

The following example enables two AWS account identifiers, 111122223333 and 444455556666, to restore the DB cluster snapshot named manual-cluster-snapshot1.

For Linux, macOS, or Unix:

```
aws rds modify-db-cluster-snapshot-attribute \
--db-cluster-snapshot-identifier manual-cluster-snapshot1 \
--attribute-name restore \
--values-to-add {"111122223333","444455556666"}
```

For Windows:

```
aws rds modify-db-cluster-snapshot-attribute ^
--db-cluster-snapshot-identifier manual-cluster-snapshot1 ^
--attribute-name restore ^
--values-to-add "[\"111122223333\", \"444455556666\"]"
```

Note

When using the Windows command prompt, you must escape double quotes ("") in JSON code by prefixing them with a backslash (\).

To remove an AWS account identifier from the list, use the --values-to-remove parameter.

Example of stopping snapshot sharing

The following example prevents AWS account ID 444455556666 from restoring the snapshot.

For Linux, macOS, or Unix:

```
aws rds modify-db-cluster-snapshot-attribute \
--db-cluster-snapshot-identifier manual-cluster-snapshot1 \
--attribute-name restore \
--values-to-remove 444455556666
```

For Windows:

```
aws rds modify-db-cluster-snapshot-attribute ^
--db-cluster-snapshot-identifier manual-cluster-snapshot1 ^
--attribute-name restore ^
--values-to-remove 444455556666
```

To list the AWS accounts enabled to restore a snapshot, use the [describe-db-cluster-snapshot-attributes](#) AWS CLI command.

RDS API

You can also share a manual DB cluster snapshot with other AWS accounts by using the Amazon RDS API. To do so, call the [ModifyDBClusterSnapshotAttribute](#) operation. Specify `restore` for `AttributeName`, and use the `ValuesToAdd` parameter to add a list of the IDs for the AWS accounts that are authorized to restore the manual snapshot.

To make a manual snapshot public and restorable by all AWS accounts, use the value `all`. However, take care not to add the `all` value for any manual snapshots that contain private information that you don't want to be available to all AWS accounts. Also, don't specify `all` for encrypted snapshots, because making such snapshots public isn't supported.

To remove sharing permission for an AWS account, use the [ModifyDBClusterSnapshotAttribute](#) operation with `AttributeName` set to `restore` and the `ValuesToRemove` parameter. To mark a manual snapshot as private, remove the value `all` from the values list for the `restore` attribute.

To list all of the AWS accounts permitted to restore a snapshot, use the [DescribeDBClusterSnapshotAttributes](#) API operation.

Exporting DB snapshot data to Amazon S3

You can export DB snapshot data to an Amazon S3 bucket. The export process runs in the background and doesn't affect the performance of your active DB cluster.

When you export a DB snapshot, Amazon Aurora extracts data from the snapshot and stores it in an Amazon S3 bucket. The data is stored in an Apache Parquet format that is compressed and consistent.

You can export manual snapshots and automated system snapshots. By default, all data in the snapshot is exported. However, you can choose to export specific sets of databases, schemas, or tables.

After the data is exported, you can analyze the exported data directly through tools like Amazon Athena or Amazon Redshift Spectrum. For more information on using Athena to read Parquet data, see [Parquet SerDe](#) in the *Amazon Athena User Guide*. For more information on using Redshift Spectrum to read Parquet data, see [COPY from columnar data formats](#) in the *Amazon Redshift Database Developer Guide*.

Amazon RDS supports exporting snapshots in all AWS Regions except the following:

- Asia Pacific (Jakarta)
- AWS GovCloud (US-East)
- AWS GovCloud (US-West)

The following table shows the Aurora MySQL engine versions that are supported for exporting snapshot data to Amazon S3. For more information about Aurora MySQL engine versions, see [Database engine updates for Amazon Aurora MySQL \(p. 1082\)](#).

Aurora MySQL version	MySQL-compatible version
3.01.0 and higher	8.0
2.04.4 and higher	5.7
1.19.2 and higher	5.6

The following table shows the Aurora PostgreSQL engine versions that are supported for exporting snapshot data to Amazon S3. For more information about Aurora PostgreSQL engine versions, see [Amazon Aurora PostgreSQL releases and engine versions \(p. 1599\)](#).

Aurora PostgreSQL version	PostgreSQL-compatible version
13.3 and higher	13.3 and higher
4.0 and higher	12.4 and higher
3.0 and higher	11.4 and higher
2.2 and higher	10.6 and higher
1.4 and higher	9.6.11 and higher

Topics

- [Limitations \(p. 519\)](#)
- [Overview of exporting snapshot data \(p. 519\)](#)

- [Setting up access to an Amazon S3 bucket \(p. 520\)](#)
- [Using a cross-account AWS KMS key for encrypting Amazon S3 exports \(p. 522\)](#)
- [Exporting a snapshot to an Amazon S3 bucket \(p. 523\)](#)
- [Monitoring snapshot exports \(p. 526\)](#)
- [Canceling a snapshot export task \(p. 527\)](#)
- [Failure messages for Amazon S3 export tasks \(p. 528\)](#)
- [Troubleshooting PostgreSQL permissions errors \(p. 529\)](#)
- [File naming convention \(p. 529\)](#)
- [Data conversion when exporting to an Amazon S3 bucket \(p. 530\)](#)

Limitations

Exporting DB snapshot data to Amazon S3 has the following limitations:

- If a database, schema, or table has characters in its name other than the following, partial export isn't supported. However, you can export the entire DB snapshot.
 - Latin letters (A–Z)
 - Digits (0–9)
 - Dollar symbol (\$)
 - Underscore (_)
- Spaces () and certain characters aren't supported in database table column names. Tables with the following characters in column names are skipped during export:

```
, ; { } ( ) \n \t = (space)
```

- If the data contains a large object such as a BLOB or CLOB, close to or greater than 500 MB, the export fails.
- If a table contains a large row close to or greater than 2 GB, the table is skipped during export.

Overview of exporting snapshot data

You use the following process to export DB snapshot data to an Amazon S3 bucket. For more details, see the following sections.

1. Identify the snapshot to export.

Use an existing automated or manual snapshot, or create a manual snapshot of a DB instance.

2. Set up access to the Amazon S3 bucket.

A *bucket* is a container for Amazon S3 objects or files. To provide the information to access a bucket, take the following steps:

- a. Identify the S3 bucket where the snapshot is to be exported to. The S3 bucket must be in the same AWS Region as the snapshot. For more information, see [Identifying the Amazon S3 bucket for export \(p. 520\)](#).
 - b. Create an AWS Identity and Access Management (IAM) role that grants the snapshot export task access to the S3 bucket. For more information, see [Providing access to an Amazon S3 bucket using an IAM role \(p. 520\)](#).
3. Create a symmetric AWS KMS key for the server-side encryption. The KMS key is used by the snapshot export task to set up AWS KMS server-side encryption when writing the export data to S3. For more information, see [Encrypting Amazon Aurora resources \(p. 1709\)](#).

The KMS key is also used for local disk encryption at rest on Amazon EC2. In addition, if you have a deny statement in your KMS key policy, make sure to explicitly exclude the AWS service principal `export.rds.amazonaws.com`.

You can use a KMS key within your AWS account, or you can use a cross-account KMS key. For more information, see [Using a cross-account AWS KMS key for encrypting Amazon S3 exports \(p. 522\)](#).

4. Export the snapshot to Amazon S3 using the console or the `start-export-task` CLI command. For more information, see [Exporting a snapshot to an Amazon S3 bucket \(p. 523\)](#).
5. To access your exported data in the Amazon S3 bucket, see [Uploading, downloading, and managing objects](#) in the *Amazon Simple Storage Service User Guide*.

Setting up access to an Amazon S3 bucket

To export DB snapshot data to an Amazon S3 file, you first give the snapshot permission to access the Amazon S3 bucket. You then create an IAM role to allow the Amazon Aurora service to write to the Amazon S3 bucket.

Topics

- [Identifying the Amazon S3 bucket for export \(p. 520\)](#)
- [Providing access to an Amazon S3 bucket using an IAM role \(p. 520\)](#)
- [Using a cross-account Amazon S3 bucket \(p. 522\)](#)

Identifying the Amazon S3 bucket for export

Identify the Amazon S3 bucket to export the DB snapshot to. Use an existing S3 bucket or create a new S3 bucket.

Note

The S3 bucket to export to must be in the same AWS Region as the snapshot.

For more information about working with Amazon S3 buckets, see the following in the *Amazon Simple Storage Service User Guide*:

- [How do I view the properties for an S3 bucket?](#)
- [How do I enable default encryption for an Amazon S3 bucket?](#)
- [How do I create an S3 bucket?](#)

Providing access to an Amazon S3 bucket using an IAM role

Before you export DB snapshot data to Amazon S3, give the snapshot export tasks write-access permission to the Amazon S3 bucket.

To do this, create an IAM policy that provides access to the bucket. Then create an IAM role and attach the policy to the role. You later assign the IAM role to your snapshot export task.

Important

If you plan to use the AWS Management Console to export your snapshot, you can choose to create the IAM policy and the role automatically when you export the snapshot. For instructions, see [Exporting a snapshot to an Amazon S3 bucket \(p. 523\)](#).

To give DB snapshot tasks access to Amazon S3

1. Create an IAM policy. This policy provides the bucket and object permissions that allow your snapshot export task to access Amazon S3.

Include in the policy the following required actions to allow the transfer of files from Amazon Aurora to an S3 bucket:

- s3:PutObject*
- s3:GetObject*
- s3>ListBucket
- s3>DeleteObject*
- s3:GetBucketLocation

Include in the policy the following resources to identify the S3 bucket and objects in the bucket. The following list of resources shows the Amazon Resource Name (ARN) format for accessing Amazon S3.

- arn:aws:s3:::*your-s3-bucket*
- arn:aws:s3:::*your-s3-bucket*/*

For more information on creating an IAM policy for Amazon Aurora, see [Creating and using an IAM policy for IAM database access \(p. 1746\)](#). See also [Tutorial: Create and attach your first customer managed policy](#) in the *IAM User Guide*.

The following AWS CLI command creates an IAM policy named `ExportPolicy` with these options. It grants access to a bucket named `your-s3-bucket`.

Note

After you create the policy, note the ARN of the policy. You need the ARN for a subsequent step when you attach the policy to an IAM role.

```
aws iam create-policy --policy-name ExportPolicy --policy-document '{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "ExportPolicy",  
            "Effect": "Allow",  
            "Action": [  
                "s3:PutObject*",  
                "s3>ListBucket",  
                "s3:GetObject*",  
                "s3>DeleteObject*",  
                "s3:GetBucketLocation"  
            ],  
            "Resource": [  
                "arn:aws:s3:::your-s3-bucket",  
                "arn:aws:s3:::your-s3-bucket/*"  
            ]  
        }  
    ]  
}'
```

2. Create an IAM role. You do this so that Aurora can assume this IAM role on your behalf to access your Amazon S3 buckets. For more information, see [Creating a role to delegate permissions to an IAM user](#) in the *IAM User Guide*.

The following example shows using the AWS CLI command to create a role named `rds-s3-export-role`.

```
aws iam create-role --role-name rds-s3-export-role --assume-role-policy-document '{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Principal": "rds.amazonaws.com",  
            "Action": "sts:AssumeRole"  
        }  
    ]  
}'
```

```
{  
    "Effect": "Allow",  
    "Principal": {  
        "Service": "export.rds.amazonaws.com"  
    },  
    "Action": "sts:AssumeRole"  
}  
]  
}'
```

3. Attach the IAM policy that you created to the IAM role that you created.

The following AWS CLI command attaches the policy created earlier to the role named `rds-s3-export-role`. Replace `your-policy-arn` with the policy ARN that you noted in an earlier step.

```
aws iam attach-role-policy --policy-arn your-policy-arn --role-name rds-s3-export-role
```

Using a cross-account Amazon S3 bucket

You can use Amazon S3 buckets across AWS accounts. To use a cross-account bucket, add a bucket policy to allow access to the IAM role that you're using for the S3 exports. For more information, see [Example 2: Bucket owner granting cross-account bucket permissions](#).

- Attach a bucket policy to your bucket, as shown in the following example.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Principal": {  
                "AWS": "arn:aws:iam::123456789012:role/Admin"  
            },  
            "Action": [  
                "s3:PutObject*",  
                "s3>ListBucket",  
                "s3:GetObject*",  
                "s3>DeleteObject*",  
                "s3:GetBucketLocation"  
            ],  
            "Resource": [  
                "arn:aws:s3:::mycrossaccountbucket",  
                "arn:aws:s3:::mycrossaccountbucket/*"  
            ]  
        }  
    ]  
}
```

Using a cross-account AWS KMS key for encrypting Amazon S3 exports

You can use a cross-account AWS KMS key to encrypt Amazon S3 exports. First, you add a key policy to the local account, then you add IAM policies in the external account. For more information, see [Allowing users in other accounts to use a KMS key](#).

To use a cross-account KMS key

1. Add a key policy to the local account.

The following example gives `ExampleRole` and `ExampleUser` in the external account 444455556666 permissions in the local account 123456789012.

```
{  
    "Sid": "Allow an external account to use this KMS key",  
    "Effect": "Allow",  
    "Principal": {  
        "AWS": [  
            "arn:aws:iam::444455556666:role/ExampleRole",  
            "arn:aws:iam::444455556666:user/ExampleUser"  
        ]  
    },  
    "Action": [  
        "kms:Encrypt",  
        "kms:Decrypt",  
        "kms:ReEncrypt*",  
        "kms:GenerateDataKey*",  
        "kms>CreateGrant",  
        "kms:DescribeKey",  
        "kms:RetireGrant"  
    ],  
    "Resource": "*"  
}
```

2. Add IAM policies to the external account.

The following example IAM policy allows the principal to use the KMS key in account 123456789012 for cryptographic operations. To give this permission to `ExampleRole` and `ExampleUser` in account 444455556666, [attach the policy](#) to them in that account.

```
{  
    "Sid": "Allow use of KMS key in account 123456789012",  
    "Effect": "Allow",  
    "Action": [  
        "kms:Encrypt",  
        "kms:Decrypt",  
        "kms:ReEncrypt*",  
        "kms:GenerateDataKey*",  
        "kms>CreateGrant",  
        "kms:DescribeKey",  
        "kms:RetireGrant"  
    ],  
    "Resource": "arn:aws:kms:us-  
west-2:123456789012:key/1234abcd-12ab-34cd-56ef-1234567890ab"  
}
```

Exporting a snapshot to an Amazon S3 bucket

You can have up to five concurrent DB snapshot export tasks in progress per account.

Note

Exporting RDS snapshots can take a while depending on your database type and size. The export task first restores and scales the entire database before extracting the data to Amazon S3. The task's progress during this phase displays as **Starting**. When the task switches to exporting data to S3, progress displays as **In progress**.

The time it takes for the export to complete depends on the data stored in the database. For example, tables with well-distributed numeric primary key or index columns export the fastest. Tables that don't contain a column suitable for partitioning and tables with only one index on a string-based column take longer. This longer export time occurs because the export uses a slower single-threaded process.

You can export a DB snapshot to Amazon S3 using the AWS Management Console, the AWS CLI, or the RDS API.

If you use a Lambda function to export a snapshot, add the `kms:DescribeKey` action to the Lambda function policy. For more information, see [AWS Lambda permissions](#).

Console

The **Export to Amazon S3** console option appears only for snapshots that can be exported to Amazon S3. A snapshot might not be available for export because of the following reasons:

- The DB engine isn't supported for S3 export.
- The DB instance version isn't supported for S3 export.
- S3 export isn't supported in the AWS Region where the snapshot was created.

To export a DB snapshot

1. Sign in to the AWS Management Console and open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. In the navigation pane, choose **Snapshots**.
3. From the tabs, choose the type of snapshot that you want to export.
4. In the list of snapshots, choose the snapshot that you want to export.
5. For **Actions**, choose **Export to Amazon S3**.

The **Export to Amazon S3** window appears.

6. For **Export identifier**, enter a name to identify the export task. This value is also used for the name of the file created in the S3 bucket.
7. Choose the data to be exported:
 - Choose **All** to export all data in the snapshot.
 - Choose **Partial** to export specific parts of the snapshot. To identify which parts of the snapshot to export, enter one or more databases, schemas, or tables for **Identifiers**, separated by spaces.

Use the following format:

```
database[.schema][.table] database2[.schema2][.table2] ... databasen[.scheman]  
[.tablen]
```

For example:

```
mydatabase mydatabase2.myschema1 mydatabase2.myschema2.mytable1  
mydatabase2.myschema2.mytable2
```

8. For **S3 bucket**, choose the bucket to export to.

To assign the exported data to a folder path in the S3 bucket, enter the optional path for **S3 prefix**.

9. For **IAM role**, either choose a role that grants you write access to your chosen S3 bucket, or create a new role.

- If you created a role by following the steps in [Providing access to an Amazon S3 bucket using an IAM role \(p. 520\)](#), choose that role.
 - If you didn't create a role that grants you write access to your chosen S3 bucket, choose **Create a new role** to create the role automatically. Next, enter a name for the role in **IAM role name**.
10. For **AWS KMS key**, enter the ARN for the key to use for encrypting the exported data.
11. Choose **Export to Amazon S3**.

AWS CLI

To export a DB snapshot to Amazon S3 using the AWS CLI, use the `start-export-task` command with the following required options:

- `--export-task-identifier`
- `--source-arn`
- `--s3-bucket-name`
- `--iam-role-arn`
- `--kms-key-id`

In the following examples, the snapshot export task is named `my-snapshot-export`, which exports a snapshot to an S3 bucket named `my-export-bucket`.

Example

For Linux, macOS, or Unix:

```
aws rds start-export-task \
  --export-task-identifier my-snapshot-export \
  --source-arn arn:aws:rds:AWS_Region:123456789012:snapshot:snapshot-name \
  --s3-bucket-name my-export-bucket \
  --iam-role-arn iam-role \
  --kms-key-id my-key
```

For Windows:

```
aws rds start-export-task ^
  --export-task-identifier my-snapshot-export ^
  --source-arn arn:aws:rds:AWS_Region:123456789012:snapshot:snapshot-name ^
  --s3-bucket-name my-export-bucket ^
  --iam-role-arn iam-role ^
  --kms-key-id my-key
```

Sample output follows.

```
{  
  "Status": "STARTING",  
  "IamRoleArn": "iam-role",  
  "ExportTime": "2019-08-12T01:23:53.109Z",  
  "S3Bucket": "my-export-bucket",  
  "PercentProgress": 0,  
  "KmsKeyId": "my-key",  
  "ExportTaskIdentifier": "my-snapshot-export",  
  "TotalExtractedDataInGB": 0,  
  "TaskStartTime": "2019-11-13T19:46:00.173Z",  
  "SourceArn": "arn:aws:rds:AWS_Region:123456789012:snapshot:snapshot-name"  
}
```

To provide a folder path in the S3 bucket for the snapshot export, include the `--s3-prefix` option in the [start-export-task](#) command.

RDS API

To export a DB snapshot to Amazon S3 using the Amazon RDS API, use the [StartExportTask](#) operation with the following required parameters:

- `ExportTaskIdentifier`
- `SourceArn`
- `S3BucketName`
- `IamRoleArn`
- `KmsKeyId`

Monitoring snapshot exports

You can monitor DB snapshot exports using the AWS Management Console, the AWS CLI, or the RDS API.

Console

To monitor DB snapshot exports

1. Sign in to the AWS Management Console and open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. In the navigation pane, choose **Snapshots**.
3. To view the list of snapshot exports, choose the **Exports in Amazon S3** tab.
4. To view information about a specific snapshot export, choose the export task.

AWS CLI

To monitor DB snapshot exports using the AWS CLI, use the [describe-export-tasks](#) command.

The following example shows how to display current information about all of your snapshot exports.

Example

```
aws rds describe-export-tasks
{
    "ExportTasks": [
        {
            "Status": "CANCELED",
            "TaskEndTime": "2019-11-01T17:36:46.961Z",
            "S3Prefix": "something",
            "ExportTime": "2019-10-24T20:23:48.364Z",
            "S3Bucket": "examplebucket",
            "PercentProgress": 0,
            "KmsKeyId": "arn:aws:kms:AWS_Region:123456789012:key/K7MDENG/bPxRfiCYEXAMPLEKEY",
            "ExportTaskIdentifier": "anewtest",
            "IamRoleArn": "arn:aws:iam::123456789012:role/export-to-s3",
            "TotalExtractedDataInGB": 0,
            "TaskStartTime": "2019-10-25T19:10:58.885Z",
            "SourceArn": "arn:aws:rds:AWS_Region:123456789012:snapshot:parameter-groups-test"
    ]
}
```

```

        },
        {
            "Status": "COMPLETE",
            "TaskEndTime": "2019-10-31T21:37:28.312Z",
            "WarningMessage": "{\"skippedTables\":[],\"skippedObjectives\":[],\"general\":[" +
                "{\"reason\":\"FAILED_TO_EXTRACT_TABLES_LIST_FOR_DATABASE\"}]}",
            "S3Prefix": "",
            "ExportTime": "2019-10-31T06:44:53.452Z",
            "S3Bucket": "examplebucket1",
            "PercentProgress": 100,
            "KmsKeyId": "arn:aws:kms:AWS_Region:123456789012:key/2Zp9Utk/
h3yCo8nvbEXAMPLEKEY",
            "ExportTaskIdentifier": "thursday-events-test",
            "IamRoleArn": "arn:aws:iam::123456789012:role/export-to-s3",
            "TotalExtractedDataInGB": 263,
            "TaskStartTime": "2019-10-31T20:58:06.998Z",
            "SourceArn": "arn:aws:rds:AWS_Region:123456789012:snapshot:rds:example-1-2019-10-31-06-44"
        },
        {
            "Status": "FAILED",
            "TaskEndTime": "2019-10-31T02:12:36.409Z",
            "FailureCause": "The S3 bucket edgcuc-export isn't located in the current AWS Region. Please, review your S3 bucket name and retry the export.",
            "S3Prefix": "",
            "ExportTime": "2019-10-30T06:45:04.526Z",
            "S3Bucket": "examplebucket2",
            "PercentProgress": 0,
            "KmsKeyId": "arn:aws:kms:AWS_Region:123456789012:key/2Zp9Utk/
h3yCo8nvbEXAMPLEKEY",
            "ExportTaskIdentifier": "wednesday-afternoon-test",
            "IamRoleArn": "arn:aws:iam::123456789012:role/export-to-s3",
            "TotalExtractedDataInGB": 0,
            "TaskStartTime": "2019-10-30T22:43:40.034Z",
            "SourceArn": "arn:aws:rds:AWS_Region:123456789012:snapshot:rds:example-1-2019-10-30-06-45"
        }
    ]
}

```

To display information about a specific snapshot export, include the `--export-task-identifier` option with the `describe-export-tasks` command. To filter the output, include the `--Filters` option. For more options, see the [describe-export-tasks](#) command.

RDS API

To display information about DB snapshot exports using the Amazon RDS API, use the [DescribeExportTasks](#) operation.

To track completion of the export workflow or to trigger another workflow, you can subscribe to Amazon Simple Notification Service topics. For more information on Amazon SNS, see [Using Amazon RDS event notification \(p. 675\)](#).

Cancelling a snapshot export task

You can cancel a DB snapshot export task using the AWS Management Console, the AWS CLI, or the RDS API.

Note

Canceling a snapshot export task doesn't remove any data that was exported to Amazon S3. For information about how to delete the data using the console, see [How do I delete objects from an S3 bucket?](#) To delete the data using the CLI, use the `delete-object` command.

Console

To cancel a snapshot export task

1. Sign in to the AWS Management Console and open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. In the navigation pane, choose **Snapshots**.
3. Choose the **Exports in Amazon S3** tab.
4. Choose the snapshot export task that you want to cancel.
5. Choose **Cancel**.
6. Choose **Cancel export task** on the confirmation page.

AWS CLI

To cancel a snapshot export task using the AWS CLI, use the `cancel-export-task` command. The command requires the `--export-task-identifier` option.

Example

```
aws rds cancel-export-task --export-task-identifier my_export
{
    "Status": "CANCELING",
    "S3Prefix": "",
    "ExportTime": "2019-08-12T01:23:53.109Z",
    "S3Bucket": "examplebucket",
    "PercentProgress": 0,
    "KmsKeyId": "arn:aws:kms:AWS_Region:123456789012:key/K7MDENG/bPxRfiCYEXAMPLEKEY",
    "ExportTaskIdentifier": "my_export",
    "IamRoleArn": "arn:aws:iam::123456789012:role/export-to-s3",
    "TotalExtractedDataInGB": 0,
    "TaskStartTime": "2019-11-13T19:46:00.173Z",
    "SourceArn": "arn:aws:rds:AWS_Region:123456789012:snapshot:export-example-1"
}
```

RDS API

To cancel a snapshot export task using the Amazon RDS API, use the `CancelExportTask` operation with the `ExportTaskIdentifier` parameter.

Failure messages for Amazon S3 export tasks

The following table describes the messages that are returned when Amazon S3 export tasks fail.

Failure message	Description
An unknown internal error occurred.	The task has failed because of an unknown error, exception, or failure.
An unknown internal error occurred writing the export task's metadata to the S3 bucket [bucket name].	The task has failed because of an unknown error, exception, or failure.
The RDS export failed to write the export task's metadata because it can't assume the IAM role [role ARN].	The export task assumes your IAM role to validate whether it is allowed to write metadata to your S3 bucket. If the task can't assume your IAM role, it fails.

Failure message	Description
The RDS export failed to write the export task's metadata to the S3 bucket [bucket name] using the IAM role [role ARN] with the KMS key [key ID]. Error code: [error code]	<p>One or more permissions are missing, so the export task can't access the S3 bucket. This failure message is raised when receiving one of the following:</p> <ul style="list-style-type: none"> • <code>AWSecurityTokenServiceException</code> with the error code <code>AccessDenied</code> • <code>AmazonS3Exception</code> with the error code <code>NoSuchBucket</code>, <code>AccessDenied</code>, <code>KMS.InvalidStateException</code>, <code>403 Forbidden</code>, or <code>KMS.DisabledException</code> <p>This means that there are settings misconfigured among the IAM role, S3 bucket, or KMS key.</p>
The IAM role [role ARN] isn't authorized to call [S3 action] on the S3 bucket [bucket name]. Review your permissions and retry the export.	The IAM policy is misconfigured. Permission for the specific S3 action on the S3 bucket is missing. This causes the export task to fail.
KMS key check failed. Check the credentials on your KMS key and try again.	The KMS key credential check failed.
S3 credential check failed. Check the permissions on your S3 bucket and IAM policy.	The S3 credential check failed.
The S3 bucket [bucket name] isn't valid. Either it isn't located in the current AWS Region or it doesn't exist. Review your S3 bucket name and retry the export.	The S3 bucket is invalid.
The S3 bucket [bucket name] isn't located in the current AWS Region. Review your S3 bucket name and retry the export.	The S3 bucket is in the wrong AWS Region.

Troubleshooting PostgreSQL permissions errors

When exporting PostgreSQL databases to Amazon S3, you might see a `PERMISSIONS_DO_NOT_EXIST` error stating that certain tables were skipped. This is usually caused by the superuser, which you specify when creating the DB instance, not having permissions to access those tables.

To fix this error, run the following command:

```
GRANT ALL PRIVILEGES ON ALL TABLES IN SCHEMA schema_name TO superuser_name
```

For more information on superuser privileges, see [Master user account privileges \(p. 1782\)](#).

File naming convention

Exported data for specific tables is stored in the format `base_prefix/files`, where the base prefix is the following:

```
export_identifier/database_name/schema_name.table_name/
```

For example:

```
export-1234567890123-459/rdststdb/rdststdb.DataInsert_7ADB5D19965123A2/
```

There are two conventions for how files are named. The current convention is the following:

```
partition_index/part-00000-random_uuid.format-based_extension
```

For example:

```
1/part-00000-c5a881bb-58ff-4ee6-1111-b41ecff340a3-c000.gz.parquet  
2/part-00000-d7a881cc-88cc-5ab7-2222-c41ecab340a4-c000.gz.parquet  
3/part-00000-f5a991ab-59aa-7fa6-3333-d41eccd340a7-c000.gz.parquet
```

The older convention is the following:

```
part-partition_index-random_uuid.format-based_extension
```

For example:

```
part-00000-c5a881bb-58ff-4ee6-1111-b41ecff340a3-c000.gz.parquet  
part-00001-d7a881cc-88cc-5ab7-2222-c41ecab340a4-c000.gz.parquet  
part-00002-f5a991ab-59aa-7fa6-3333-d41eccd340a7-c000.gz.parquet
```

The file naming convention is subject to change. Therefore, when reading target tables we recommend that you read everything inside the base prefix for the table.

Data conversion when exporting to an Amazon S3 bucket

When you export a DB snapshot to an Amazon S3 bucket, Amazon Aurora converts data to, exports data in, and stores data in the Parquet format. For more information about Parquet, see the [Apache Parquet](#) website.

Parquet stores all data as one of the following primitive types:

- BOOLEAN
- INT32
- INT64
- INT96
- FLOAT
- DOUBLE
- BYTE_ARRAY – A variable-length byte array, also known as binary
- FIXED_LEN_BYTE_ARRAY – A fixed-length byte array used when the values have a constant size

The Parquet data types are few to reduce the complexity of reading and writing the format. Parquet provides logical types for extending primitive types. A *logical type* is implemented as an annotation with the data in a `LogicalType` metadata field. The logical type annotation explains how to interpret the primitive type.

When the `STRING` logical type annotates a `BYTE_ARRAY` type, it indicates that the byte array should be interpreted as a UTF-8 encoded character string. After an export task completes, Amazon Aurora notifies

you if any string conversion occurred. The underlying data exported is always the same as the data from the source. However, due to the encoding difference in UTF-8, some characters might appear different from the source when read in tools such as Athena.

For more information, see [Parquet logical type definitions](#) in the Parquet documentation.

Topics

- [MySQL data type mapping to Parquet \(p. 531\)](#)
- [PostgreSQL data type mapping to Parquet \(p. 533\)](#)

MySQL data type mapping to Parquet

The following table shows the mapping from MySQL data types to Parquet data types when data is converted and exported to Amazon S3.

Source data type	Parquet primitive type	Logical type annotation	Conversion notes
Numeric data types			
BIGINT	INT64		
BIGINT UNSIGNED	FIXED_LEN_BYTE_ARRAY(9)DECIMAL(20,0)		Parquet supports only signed types, so the mapping requires an additional byte (8 plus 1) to store the BIGINT_UNSIGNED type.
BIT	BYTE_ARRAY		
DECIMAL	INT32	DECIMAL(p,s)	If the source value is less than 2^{31} , it's stored as INT32.
	INT64	DECIMAL(p,s)	If the source value is 2^{31} or greater, but less than 2^{63} , it's stored as INT64.
	FIXED_LEN_BYTE_ARRAY(ND)DECIMAL(p,s)		If the source value is 2^{63} or greater, it's stored as FIXED_LEN_BYTE_ARRAY(N).
	BYTE_ARRAY	STRING	Parquet doesn't support Decimal precision greater than 38. The Decimal value is converted to a string in a BYTE_ARRAY type and encoded as UTF8.
DOUBLE	DOUBLE		
FLOAT	DOUBLE		
INT	INT32		

Source data type	Parquet primitive type	Logical type annotation	Conversion notes
INT UNSIGNED	INT64		
MEDIUMINT	INT32		
MEDIUMINT UNSIGNED	INT64		
NUMERIC	INT32	DECIMAL(p,s)	If the source value is less than 2^{31} , it's stored as INT32.
	INT64	DECIMAL(p,s)	If the source value is 2^{31} or greater, but less than 2^{63} , it's stored as INT64.
	FIXED_LEN_ARRAY(N)	DECIMAL(p,s)	If the source value is 2^{63} or greater, it's stored as FIXED_LEN_BYTE_ARRAY(N).
	BYTE_ARRAY	STRING	Parquet doesn't support Numeric precision greater than 38. This Numeric value is converted to a string in a BYTE_ARRAY type and encoded as UTF8.
SMALLINT	INT32		
SMALLINT UNSIGNED	INT32		
TINYINT	INT32		
TINYINT UNSIGNED	INT32		
String data types			
BINARY	BYTE_ARRAY		
BLOB	BYTE_ARRAY		
CHAR	BYTE_ARRAY		
ENUM	BYTE_ARRAY	STRING	
LINESTRING	BYTE_ARRAY		
LONGBLOB	BYTE_ARRAY		
LONGTEXT	BYTE_ARRAY	STRING	
MEDIUMBLOB	BYTE_ARRAY		
MEDIUMTEXT	BYTE_ARRAY	STRING	
MULTILINESTRING	BYTE_ARRAY		
SET	BYTE_ARRAY	STRING	
TEXT	BYTE_ARRAY	STRING	

Source data type	Parquet primitive type	Logical type annotation	Conversion notes
TINYBLOB	BYTE_ARRAY		
TINYTEXT	BYTE_ARRAY	STRING	
VARBINARY	BYTE_ARRAY		
VARCHAR	BYTE_ARRAY	STRING	
Date and time data types			
DATE	BYTE_ARRAY	STRING	A date is converted to a string in a BYTE_ARRAY type and encoded as UTF8.
DATETIME	INT64	TIMESTAMP_MICROS	
TIME	BYTE_ARRAY	STRING	A TIME type is converted to a string in a BYTE_ARRAY and encoded as UTF8.
TIMESTAMP	INT64	TIMESTAMP_MICROS	
YEAR	INT32		
Geometric data types			
GEOMETRY	BYTE_ARRAY		
GEOMETRYCOLLECTION	BYTE_ARRAY		
MULTIPOINT	BYTE_ARRAY		
MULTIPOLYGON	BYTE_ARRAY		
POINT	BYTE_ARRAY		
POLYGON	BYTE_ARRAY		
JSON data type			
JSON	BYTE_ARRAY	STRING	

PostgreSQL data type mapping to Parquet

The following table shows the mapping from PostgreSQL data types to Parquet data types when data is converted and exported to Amazon S3.

PostgreSQL data type	Parquet primitive type	Logical type annotation	Mapping notes
Numeric data types			
BIGINT	INT64		

PostgreSQL data type	Parquet primitive type	Logical type annotation	Mapping notes
BIGSERIAL	INT64		
DECIMAL	BYTE_ARRAY	STRING	A DECIMAL type is converted to a string in a BYTE_ARRAY type and encoded as UTF8. This conversion is to avoid complications due to data precision and data values that are not a number (NaN).
DOUBLE PRECISION	DOUBLE		
INTEGER	INT32		
MONEY	BYTE_ARRAY	STRING	
REAL	FLOAT		
SERIAL	INT32		
SMALLINT	INT32	INT_16	
SMALLSERIAL	INT32	INT_16	
String and related data types			
ARRAY	BYTE_ARRAY	STRING	An array is converted to a string and encoded as BINARY (UTF8). This conversion is to avoid complications due to data precision, data values that are not a number (NaN), and time data values.
BIT	BYTE_ARRAY	STRING	
BIT VARYING	BYTE_ARRAY	STRING	
BYTEA	BINARY		
CHAR	BYTE_ARRAY	STRING	
CHAR(N)	BYTE_ARRAY	STRING	
ENUM	BYTE_ARRAY	STRING	
NAME	BYTE_ARRAY	STRING	
TEXT	BYTE_ARRAY	STRING	
TEXT SEARCH	BYTE_ARRAY	STRING	

PostgreSQL data type	Parquet primitive type	Logical type annotation	Mapping notes
VARCHAR(N)	BYTE_ARRAY	STRING	
XML	BYTE_ARRAY	STRING	
Date and time data types			
DATE	BYTE_ARRAY	STRING	
INTERVAL	BYTE_ARRAY	STRING	
TIME	BYTE_ARRAY	STRING	
TIME WITH TIME ZONE	BYTE_ARRAY	STRING	
TIMESTAMP	BYTE_ARRAY	STRING	
TIMESTAMP WITH TIME ZONE	BYTE_ARRAY	STRING	
Geometric data types			
BOX	BYTE_ARRAY	STRING	
CIRCLE	BYTE_ARRAY	STRING	
LINE	BYTE_ARRAY	STRING	
LINESEGMENT	BYTE_ARRAY	STRING	
PATH	BYTE_ARRAY	STRING	
POINT	BYTE_ARRAY	STRING	
POLYGON	BYTE_ARRAY	STRING	
JSON data types			
JSON	BYTE_ARRAY	STRING	
JSONB	BYTE_ARRAY	STRING	
Other data types			
BOOLEAN	BOOLEAN		
CIDR	BYTE_ARRAY	STRING	Network data type
COMPOSITE	BYTE_ARRAY	STRING	
DOMAIN	BYTE_ARRAY	STRING	
INET	BYTE_ARRAY	STRING	Network data type
MACADDR	BYTE_ARRAY	STRING	
OBJECT IDENTIFIER	N/A		
PG_LSN	BYTE_ARRAY	STRING	
RANGE	BYTE_ARRAY	STRING	

PostgreSQL data type	Parquet primitive type	Logical type annotation	Mapping notes
UUID	BYTE_ARRAY	STRING	

Restoring a DB cluster to a specified time

You can restore a DB cluster to a specific point in time, creating a new DB cluster.

When you restore a DB cluster to a point in time, you can choose the default virtual private cloud (VPC) security group. Or you can apply a custom VPC security group to your DB cluster.

Restored DB clusters are automatically associated with the default DB cluster and DB parameter groups. However, you can apply custom parameter groups by specifying them during a restore.

Amazon RDS uploads transaction logs for DB clusters to Amazon S3 continuously. To see the latest restorable time for a DB cluster, use the AWS CLI [describe-db-clusters](#) command and look at the value returned in the `LatestRestorableTime` field for the DB cluster.

You can restore to any point in time within your backup retention period. To see the earliest restorable time for a DB cluster, use the AWS CLI [describe-db-clusters](#) command and look at the value returned in the `EarliestRestorableTime` field for the DB cluster.

Note

Information in this topic applies to Amazon Aurora. For information on restoring an Amazon RDS DB instance, see [Restoring a DB instance to a specified time](#).

For more information about backing up and restoring an Aurora DB cluster, see [Overview of backing up and restoring an Aurora DB cluster \(p. 491\)](#).

For Aurora MySQL, you can restore a provisioned DB cluster to an Aurora Serverless DB cluster.

For more information, see [Restoring an Aurora Serverless v1 DB cluster \(p. 166\)](#).

You can restore a DB cluster to a point in time using the AWS Management Console, the AWS CLI, or the RDS API.

Console

To restore a DB cluster to a specified time

1. Sign in to the AWS Management Console and open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. In the navigation pane, choose **Databases**.
3. Choose the DB cluster that you want to restore.
4. For **Actions**, choose **Restore to point in time**.

The **Restore to point in time** window appears.

5. Choose **Latest restorable time** to restore to the latest possible time, or choose **Custom** to choose a time.

If you chose **Custom**, enter the date and time to which you want to restore the cluster.

Note

Times are shown in your local time zone, which is indicated by an offset from Coordinated Universal Time (UTC). For example, UTC-5 is Eastern Standard Time/Central Daylight Time.

6. For **DB instance identifier**, enter the name of the target restored DB cluster. The name must be unique.
7. Choose other options as needed, such as DB instance class and storage.
8. Choose **Restore to point in time**.

AWS CLI

To restore a DB cluster to a specified time, use the AWS CLI command [restore-db-cluster-to-point-in-time](#) to create a new DB cluster.

Example

For Linux, macOS, or Unix:

```
aws rds restore-db-cluster-to-point-in-time \
--source-db-cluster-identifier mysourcedbcluster \
--db-cluster-identifier mytargetdbcluster \
--restore-to-time 2017-10-14T23:45:00.000Z
```

For Windows:

```
aws rds restore-db-cluster-to-point-in-time ^
--source-db-cluster-identifier mysourcedbcluster ^
--db-cluster-identifier mytargetdbcluster ^
--restore-to-time 2017-10-14T23:45:00.000Z
```

Important

If you use the console to restore a DB cluster to a specified time, then Amazon RDS automatically creates the primary instance (writer) for your DB cluster. If you use the AWS CLI to restore a DB cluster to a specified time, you must explicitly create the primary instance for your DB cluster. The primary instance is the first instance that is created in a DB cluster.

To create the primary instance for your DB cluster, call the [create-db-instance](#) AWS CLI command. Include the name of the DB cluster as the `--db-cluster-identifier` option value.

RDS API

To restore a DB cluster to a specified time, call the Amazon RDS API

[RestoreDBClusterToPointInTime](#) operation with the following parameters:

- `SourceDBClusterIdentifier`
- `DBClusterIdentifier`
- `RestoreToTime`

Important

If you use the console to restore a DB cluster to a specified time, then Amazon RDS automatically creates the primary instance (writer) for your DB cluster. If you use the RDS API to restore a DB cluster to a specified time, make sure to explicitly create the primary instance for your DB cluster. The primary instance is the first instance that is created in a DB cluster.

To create the primary instance for your DB cluster, call the RDS API operation [CreateDBInstance](#). Include the name of the DB cluster as the `DBClusterIdentifier` parameter value.

Deleting a DB cluster snapshot

You can delete DB cluster snapshots managed by Amazon RDS when you no longer need them.

Note

To delete backups managed by AWS Backup, use the AWS Backup console. For information about AWS Backup, see the [AWS Backup Developer Guide](#).

Deleting a DB cluster snapshot

You can delete a DB cluster snapshot using the console, the AWS CLI, or the RDS API.

To delete a shared or public snapshot, you must sign in to the AWS account that owns the snapshot.

Console

To delete a DB cluster snapshot

1. Sign in to the AWS Management Console and open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. In the navigation pane, choose **Snapshots**.
3. Choose the DB cluster snapshot that you want to delete.
4. For **Actions**, choose **Delete Snapshot**.
5. Choose **Delete** on the confirmation page.

AWS CLI

You can delete a DB cluster snapshot by using the AWS CLI command [delete-db-cluster-snapshot](#).

The following options are used to delete a DB cluster snapshot.

- `--db-cluster-snapshot-identifier` – The identifier for the DB cluster snapshot.

Example

The following code deletes the `mydbclustersnapshot` DB cluster snapshot.

For Linux, macOS, or Unix:

```
aws rds delete-db-cluster-snapshot \
--db-cluster-snapshot-identifier mydbclustersnapshot
```

For Windows:

```
aws rds delete-db-cluster-snapshot ^
--db-cluster-snapshot-identifier mydbclustersnapshot
```

RDS API

You can delete a DB cluster snapshot by using the Amazon RDS API operation [DeleteDBClusterSnapshot](#).

The following parameters are used to delete a DB cluster snapshot.

- **DBClusterSnapshotIdentifier** – The identifier for the DB cluster snapshot.

Monitoring metrics in an Amazon Aurora cluster

Amazon Aurora uses a cluster of replicated database servers. Typically, monitoring an Aurora cluster requires checking the health of multiple DB instances. The instances might have specialized roles, handling mostly write operations, only read operations, or a combination. You also monitor the overall health of the cluster by measuring the *replication lag*. This is the amount of time for changes made by one DB instance to be available to the other instances.

Topics

- [Overview of monitoring metrics in Amazon Aurora \(p. 542\)](#)
- [Viewing cluster status and recommendations \(p. 546\)](#)
- [Viewing metrics in the Amazon RDS console \(p. 563\)](#)
- [Monitoring Amazon Aurora metrics with Amazon CloudWatch \(p. 566\)](#)
- [Monitoring DB load with Performance Insights on Amazon Aurora \(p. 573\)](#)
- [Analyzing performance anomalies with DevOps Guru for RDS \(p. 623\)](#)
- [Monitoring OS metrics with Enhanced Monitoring \(p. 626\)](#)
- [Metrics reference for Amazon Aurora \(p. 633\)](#)

Overview of monitoring metrics in Amazon Aurora

Monitoring is an important part of maintaining the reliability, availability, and performance of Amazon Aurora and your AWS solutions. To more easily debug multi-point failures, we recommend that you collect monitoring data from all parts of your AWS solution.

Topics

- [Monitoring plan \(p. 542\)](#)
- [Performance baseline \(p. 542\)](#)
- [Performance guidelines \(p. 542\)](#)
- [Monitoring tools \(p. 543\)](#)

Monitoring plan

Before you start monitoring Amazon Aurora, create a monitoring plan. This plan should answer the following questions:

- What are your monitoring goals?
- Which resources will you monitor?
- How often will you monitor these resources?
- Which monitoring tools will you use?
- Who will perform the monitoring tasks?
- Whom should be notified when something goes wrong?

Performance baseline

To achieve your monitoring goals, you need to establish a baseline. To do this, measure performance under different load conditions at various times in your Amazon Aurora environment. You can monitor metrics such as the following:

- Network throughput
- Client connections
- I/O for read, write, or metadata operations
- Burst credit balances for your DB instances

We recommend that you store historical performance data for Amazon Aurora. Using the stored data, you can compare current performance against past trends. You can also distinguish normal performance patterns from anomalies, and devise techniques to address issues.

Performance guidelines

In general, acceptable values for performance metrics depend on what your application is doing relative to your baseline. Investigate consistent or trending variances from your baseline. The following metrics are often the source of performance issues:

- **High CPU or RAM consumption** – High values for CPU or RAM consumption might be appropriate, if they're in keeping with your goals for your application (like throughput or concurrency) and are expected.

- **Disk space consumption** – Investigate disk space consumption if space used is consistently at or above 85 percent of the total disk space. See if it is possible to delete data from the instance or archive data to a different system to free up space.
- **Network traffic** – For network traffic, talk with your system administrator to understand what expected throughput is for your domain network and internet connection. Investigate network traffic if throughput is consistently lower than expected.
- **Database connections** – If you see high numbers of user connections and also decreases in instance performance and response time, consider constraining database connections. The best number of user connections for your DB instance varies based on your instance class and the complexity of the operations being performed. To determine the number of database connections, associate your DB instance with a parameter group where the `User_Connections` parameter is set to a value other than 0 (unlimited). You can either use an existing parameter group or create a new one. For more information, see [Working with DB parameter groups and DB cluster parameter groups \(p. 339\)](#).
- **IOPS metrics** – The expected values for IOPS metrics depend on disk specification and server configuration, so use your baseline to know what is typical. Investigate if values are consistently different than your baseline. For best IOPS performance, make sure that your typical working set fits into memory to minimize read and write operations.

When performance falls outside your established baseline, you might need to make changes to optimize your database availability for your workload. For example, you might need to change the instance class of your DB instance. Or you might need to change the number of DB instances and read replicas that are available for clients.

Monitoring tools

Monitoring is an important part of maintaining the reliability, availability, and performance of Amazon Aurora and your other AWS solutions. AWS provides various monitoring tools to watch Amazon Aurora, report when something is wrong, and take automatic actions when appropriate.

Topics

- [Automated monitoring tools \(p. 543\)](#)
- [Manual monitoring tools \(p. 544\)](#)

Automated monitoring tools

We recommend that you automate monitoring tasks as much as possible.

Topics

- [Amazon Aurora cluster status and recommendations \(p. 543\)](#)
- [Amazon CloudWatch metrics for Amazon Aurora \(p. 544\)](#)
- [Amazon RDS Performance Insights and operating-system monitoring \(p. 544\)](#)
- [Integrated services \(p. 544\)](#)

Amazon Aurora cluster status and recommendations

You can use the following automated tools to watch Amazon Aurora and report when something is wrong:

- **Amazon Aurora cluster status** — View details about the current status of your cluster by using the Amazon RDS console, the AWS CLI, or the RDS API.

- **Amazon Aurora recommendations** — Respond to automated recommendations for database resources, such as DB instances, DB clusters, and DB cluster parameter groups. For more information, see [Viewing Amazon Aurora recommendations \(p. 558\)](#).

Amazon CloudWatch metrics for Amazon Aurora

Amazon Aurora integrates with Amazon CloudWatch for additional monitoring capabilities.

- **Amazon CloudWatch** – This service monitors your AWS resources and the applications you run on AWS in real time. You can use the following Amazon CloudWatch features with Amazon Aurora:
 - **Amazon CloudWatch metrics** – Amazon Aurora automatically sends metrics to CloudWatch every minute for each active database. You don't get additional charges for Amazon RDS metrics in CloudWatch. For more information, see [Amazon CloudWatch metrics for Amazon Aurora \(p. 633\)](#)
 - **Amazon CloudWatch alarms** – You can watch a single Amazon Aurora metric over a specific time period. You can then perform one or more actions based on the value of the metric relative to a threshold that you set.

Amazon RDS Performance Insights and operating-system monitoring

You can use the following automated tools to monitor Amazon Aurora performance:

- **Amazon RDS Performance Insights** – Assess the load on your database, and determine when and where to take action. For more information, see [Monitoring DB load with Performance Insights on Amazon Aurora \(p. 573\)](#).
- **Amazon RDS Enhanced Monitoring** – Look at metrics in real time for the operating system. For more information, see [Monitoring OS metrics with Enhanced Monitoring \(p. 626\)](#).

Integrated services

The following AWS services are integrated with Amazon Aurora:

- *Amazon EventBridge* is a serverless event bus service that makes it easy to connect your applications with data from a variety of sources. For more information, see [Monitoring Amazon Aurora events \(p. 671\)](#).
- *Amazon CloudWatch Logs* lets you monitor, store, and access your log files from Amazon Aurora instances, CloudTrail, and other sources. For more information, see [Monitoring Amazon Aurora log files \(p. 695\)](#).
- *AWS CloudTrail* captures API calls and related events made by or on behalf of your AWS account and delivers the log files to an Amazon S3 bucket that you specify. For more information, see [Monitoring Amazon Aurora API calls in AWS CloudTrail \(p. 710\)](#).
- *Database Activity Streams* is an Amazon Aurora feature that provides a near-real-time stream of the activity in your DB cluster. For more information, see [Monitoring Amazon Aurora with Database Activity Streams \(p. 714\)](#).
- *DevOps Guru for RDS* is a capability of Amazon DevOps Guru that applies machine learning to Performance Insights metrics for Amazon Aurora databases. For more information, see [Analyzing performance anomalies with DevOps Guru for RDS \(p. 623\)](#).

Manual monitoring tools

You need to manually monitor those items that the CloudWatch alarms don't cover. The Amazon RDS, CloudWatch, AWS Trusted Advisor and other AWS console dashboards provide an at-a-glance view of the state of your AWS environment. We recommend that you also check the log files on your DB instance.

- From the Amazon RDS console, you can monitor the following items for your resources:
 - The number of connections to a DB instance
 - The amount of read and write operations to a DB instance
 - The amount of storage that a DB instance is currently using
 - The amount of memory and CPU being used for a DB instance
 - The amount of network traffic to and from a DB instance
- From the Trusted Advisor dashboard, you can review the following cost optimization, security, fault tolerance, and performance improvement checks:
 - Amazon RDS Idle DB Instances
 - Amazon RDS Security Group Access Risk
 - Amazon RDS Backups
 - Amazon RDS Multi-AZ
 - Aurora DB Instance Accessibility

For more information on these checks, see [Trusted Advisor best practices \(checks\)](#).

- CloudWatch home page shows:
 - Current alarms and status
 - Graphs of alarms and resources
 - Service health status

In addition, you can use CloudWatch to do the following:

- Create [customized dashboards](#) to monitor the services that you care about.
- Graph metric data to troubleshoot issues and discover trends.
- Search and browse all your AWS resource metrics.
- Create and edit alarms to be notified of problems.

Viewing cluster status and recommendations

Using the Amazon RDS console, you can quickly access the status of your DB cluster and respond to Amazon Aurora recommendations.

Topics

- [Viewing an Amazon Aurora DB cluster \(p. 547\)](#)
- [Viewing DB cluster status \(p. 553\)](#)
- [Viewing DB instance status in an Aurora cluster \(p. 555\)](#)
- [Viewing Amazon Aurora recommendations \(p. 558\)](#)

Viewing an Amazon Aurora DB cluster

You have several options for viewing information about your Amazon Aurora DB clusters and the DB instances in your DB clusters.

- You can view DB clusters and DB instances in the Amazon RDS console by choosing **Databases** from the navigation pane.
- You can get DB cluster and DB instance information using the AWS Command Line Interface (AWS CLI).
- You can get DB cluster and DB instance information using the Amazon RDS API.

Console

In the Amazon RDS console, you can see details about a DB cluster by choosing **Databases** from the console's navigation pane. You can also see details about DB instances that are members of an Amazon Aurora DB cluster on the **Databases** page.

The **Databases** list shows all of the DB clusters for your AWS account. When you choose a DB cluster, you see both information about the DB cluster and also a list of the DB instances that are members of that DB cluster. You can choose the identifier for a DB instance in the list to go directly to the details page for that DB instance in the RDS console.

To view the details page for a DB cluster, choose **Databases** in the navigation pane, and then choose the name of the DB cluster.

You can modify your DB cluster by choosing **Databases** from the console's navigation pane to go to the **Databases** list. To modify a DB cluster, select the DB cluster from the **Databases** list and choose **Modify**.

To modify a DB instance that is a member of a DB cluster, choose **Databases** from the console's navigation pane to go to the **Databases** list.

For example, the following image shows the details page for the DB cluster named `aurora-test`. The DB cluster has four DB instances shown in the **DB identifier** list. The writer DB instance, `dbinstance4`, is the primary DB instance for the DB cluster.

aurora-test

Related

Filter databases

DB identifier	Role	Engine	Region & AZ
aurora-test	Regional	Aurora MySQL	us-east-1
dbinstance4	Writer	Aurora MySQL	us-east-1a
dbinstance1	Reader	Aurora MySQL	us-east-1b
dbinstance2	Reader	Aurora MySQL	us-east-1b
dbinstance3	Reader	Aurora MySQL	us-east-1a

Connectivity & security **Monitoring** **Logs & events** **Configuration** **Maintenance & backups** **Tags**

Endpoints (2)

Filter endpoint

Endpoint name
aurora-test.cluster-ro- .us-east-1.rds.amazonaws.com
aurora-test.cluster- .us-east-1.rds.amazonaws.com

If you click the link for the dbinstance4 DB instance identifier, the Amazon RDS console shows the details page for the dbinstance4 DB instance, as shown in the following image.

dbinstance4

Related

Filter databases

DB identifier	Role	Engine
aurora-test	Regional	Aurora MySQL
dbinstance4	Writer	Aurora MySQL
dbinstance1	Reader	Aurora MySQL
dbinstance2	Reader	Aurora MySQL
dbinstance3	Reader	Aurora MySQL

Connectivity & security **Monitoring** **Logs & events** **Configuration** **Maintenance** **Tags**

Connectivity & security

Endpoint & port

Endpoint	Net
dbinstance4. [REDACTED].us-east-1.rds.amazonaws.com	Avail
Port	VPC
3306	vpc-[REDACTED]
	cloud

AWS CLI

To view DB cluster information by using the AWS CLI, use the [describe-db-clusters](#) command. For example, the following AWS CLI command lists the DB cluster information for all of the DB clusters in the `us-east-1` region for the configured AWS account.

```
aws rds describe-db-clusters --region us-east-1
```

The command returns the following output if your AWS CLI is configured for JSON output.

```
{  
    "DBClusters": [  
        {  
            "Status": "available",  
            "Engine": "aurora",  
            "Endpoint": "sample-cluster1.cluster-123456789012.us-east-1.rds.amazonaws.com",  
            "AllocatedStorage": 1,  
            "DBClusterIdentifier": "sample-cluster1",  
            "MasterUsername": "mymasteruser",  
            "EarliestRestorableTime": "2016-03-30T03:35:42.563Z",  
            "Port": 3306,  
            "PreferredMaintenanceWindow": "2016-04-01T03:35:42.563Z-2016-04-01T03:35:42.563Z",  
            "LastModified": "2016-03-30T03:35:42.563Z",  
            "DBClusterArn": "arn:aws:rds:us-east-1:123456789012:cluster-sample-cluster1",  
            "DBSubnetGroup": "sample-subnetgroup",  
            "Characteristics": "{'Engine': 'aurora'}"  
        }  
    ]  
}
```

```

"DBClusterMembers": [
    {
        "IsClusterWriter": false,
        "DBClusterParameterGroupStatus": "in-sync",
        "DBInstanceIdentifier": "sample-replica"
    },
    {
        "IsClusterWriter": true,
        "DBClusterParameterGroupStatus": "in-sync",
        "DBInstanceIdentifier": "sample-primary"
    }
],
"Port": 3306,
"PreferredBackupWindow": "03:34-04:04",
"VpcSecurityGroups": [
    {
        "Status": "active",
        "VpcSecurityGroupId": "sg-ddb65fec"
    }
],
"DBSubnetGroup": "default",
"StorageEncrypted": false,
"DatabaseName": "sample",
"EngineVersion": "5.6.10a",
"DBClusterParameterGroup": "default.aurora5.6",
"BackupRetentionPeriod": 1,
"AvailabilityZones": [
    "us-east-1b",
    "us-east-1c",
    "us-east-1d"
],
"LatestRestorableTime": "2016-03-31T20:06:08.903Z",
"PreferredMaintenanceWindow": "wed:08:15-wed:08:45"
},
{
    "Status": "available",
    "Engine": "aurora",
    "Endpoint": "aurora-sample.cluster-123456789012.us-east-1.rds.amazonaws.com",
    "AllocatedStorage": 1,
    "DBClusterIdentifier": "aurora-sample-cluster",
    "MasterUsername": "mymasteruser",
    "EarliestRestorableTime": "2016-03-30T10:21:34.826Z",
    "DBClusterMembers": [
        {
            "IsClusterWriter": false,
            "DBClusterParameterGroupStatus": "in-sync",
            "DBInstanceIdentifier": "aurora-replica-sample"
        },
        {
            "IsClusterWriter": true,
            "DBClusterParameterGroupStatus": "in-sync",
            "DBInstanceIdentifier": "aurora-sample"
        }
    ],
    "Port": 3306,
    "PreferredBackupWindow": "10:20-10:50",
    "VpcSecurityGroups": [
        {
            "Status": "active",
            "VpcSecurityGroupId": "sg-55da224b"
        }
    ],
    "DBSubnetGroup": "default",
    "StorageEncrypted": false,
    "DatabaseName": "sample",
    "EngineVersion": "5.6.10a",

```

```
        "DBClusterParameterGroup": "default.aurora5.6",
        "BackupRetentionPeriod": 1,
        "AvailabilityZones": [
            "us-east-1b",
            "us-east-1c",
            "us-east-1d"
        ],
        "LatestRestorableTime": "2016-03-31T20:00:11.491Z",
        "PreferredMaintenanceWindow": "sun:03:53-sun:04:23"
    }
]
}
```

RDS API

To view DB cluster information using the Amazon RDS API, use the [DescribeDBClusters](#) operation. For example, the following Amazon RDS API command lists the DB cluster information for all of the DB clusters in the us-east-1 region.

```
https://rds.us-east-1.amazonaws.com/
?Action=DescribeDBClusters
&MaxRecords=100
&SignatureMethod=HmacSHA256
&SignatureVersion=4
&Version=2014-10-31
&X-Amz-Algorithm=AWS4-HMAC-SHA256
&X-Amz-Credential=AKIADQKE4SARGYLE/20140722/us-east-1/rds/aws4_request
&X-Amz-Date=20140722T200807Z
&X-Amz-SignedHeaders=content-type;host;user-agent;x-amz-content-sha256;x-amz-date
&X-Amz-Signature=2d4f2b9e8abc31122b5546f94c0499bba47de813cb875f9b9c78e8e19c9afe1b
```

The action returns the following output:

```
<DescribeDBClustersResponse xmlns="http://rds.amazonaws.com/doc/2014-10-31/">
<DescribeDBClustersResult>
<DBClusters>
<DBCluster>
<Engine>aurora5.6</Engine>
<Status>available</Status>
<BackupRetentionPeriod>0</BackupRetentionPeriod>
<DBSubnetGroup>my-subgroup</DBSubnetGroup>
<EngineVersion>5.6.10a</EngineVersion>
<Endpoint>sample-cluster2.cluster-cbfvmb0y5fy.us-east-1.rds.amazonaws.com</
Endpoint>
<DBClusterIdentifier>sample-cluster2</DBClusterIdentifier>
<PreferredBackupWindow>04:45-05:15</PreferredBackupWindow>
<PreferredMaintenanceWindow>sat:05:56-sat:06:26</PreferredMaintenanceWindow>
<DBClusterMembers/>
<AllocatedStorage>15</AllocatedStorage>
<MasterUsername>awsuser</MasterUsername>
</DBCluster>
<DBCluster>
<Engine>aurora5.6</Engine>
<Status>available</Status>
<BackupRetentionPeriod>0</BackupRetentionPeriod>
<DBSubnetGroup>my-subgroup</DBSubnetGroup>
<EngineVersion>5.6.10a</EngineVersion>
<Endpoint>sample-cluster3.cluster-cefgqfx9y5fy.us-east-1.rds.amazonaws.com</
Endpoint>
<DBClusterIdentifier>sample-cluster3</DBClusterIdentifier>
<PreferredBackupWindow>07:06-07:36</PreferredBackupWindow>
<PreferredMaintenanceWindow>tue:10:18-tue:10:48</PreferredMaintenanceWindow>
<DBClusterMembers>
```

```
<DBClusterMember>
  <IsClusterWriter>true</IsClusterWriter>
  <DBInstanceIdentifier>sample-cluster3-master</DBInstanceIdentifier>
</DBClusterMember>
<DBClusterMember>
  <IsClusterWriter>false</IsClusterWriter>
  <DBInstanceIdentifier>sample-cluster3-read1</DBInstanceIdentifier>
</DBClusterMember>
</DBClusterMembers>
<AllocatedStorage>15</AllocatedStorage>
<MasterUsername>awsuser</MasterUsername>
</DBCluster>
</DBClusters>
</DescribeDBClustersResult>
<ResponseMetadata>
  <RequestId>d682b02c-1383-11b4-a6bb-172dfac7f170</RequestId>
</ResponseMetadata>
</DescribeDBClustersResponse>
```

Viewing DB cluster status

The status of a DB cluster indicates its health. You can view the status of a DB cluster by using the Amazon RDS console, the AWS CLI command [describe-db-clusters](#), or the API operation [DescribeDBClusters](#).

Note

Aurora also uses another status called *maintenance status*, which is shown in the **Maintenance** column of the Amazon RDS console. This value indicates the status of any maintenance patches that need to be applied to a DB cluster. Maintenance status is independent of DB cluster status. For more information on *maintenance status*, see [Applying updates for a DB cluster \(p. 445\)](#).

Find the possible status values for DB clusters in the following table.

DB cluster status	Billed	Description
available	Billed	The DB cluster is healthy and available. When an Aurora Serverless cluster is available and paused, you're billed for storage only.
backing-up	Billed	The DB cluster is currently being backed up.
backtracking	Billed	The DB cluster is currently being backtracked. This status only applies to Aurora MySQL.
cloning-failed	Not billed	Cloning a DB cluster failed.
creating	Not billed	The DB cluster is being created. The DB cluster is inaccessible while it is being created.
deleting	Not billed	The DB cluster is being deleted.
failing-over	Billed	A failover from the primary instance to an Aurora Replica is being performed.
inaccessible-encryption-credentials	Not billed	The AWS KMS key used to encrypt or decrypt the DB cluster can't be accessed or recovered.
inaccessible-encryption-credentials-recoverable	Billed for storage	The KMS key used to encrypt or decrypt the DB cluster can't be accessed. However, if the KMS key is active, restarting the DB cluster can recover it. For more information, see Enabling encryption for an Amazon Aurora DB cluster (p. 1710) .
maintenance	Billed	Amazon RDS is applying a maintenance update to the DB cluster. This status is used for DB cluster-level maintenance that RDS schedules well in advance.
migrating	Billed	A DB cluster snapshot is being restored to a DB cluster.
migration-failed	Not billed	A migration failed.
modifying	Billed	The DB cluster is being modified because of a customer request to modify the DB cluster.
promoting	Billed	A read replica is being promoted to a standalone DB cluster.

DB cluster status	Billed	Description
renaming	Billed	The DB cluster is being renamed because of a customer request to rename it.
resetting-master-credentials	Billed	The master credentials for the DB cluster are being reset because of a customer request to reset them.
starting	Billed for storage	The DB cluster is starting.
stopped	Billed for storage	The DB cluster is stopped.
stopping	Billed for storage	The DB cluster is being stopped.
update-iam-db-auth	Billed	IAM authorization for the DB cluster is being updated.
upgrading	Billed	The DB cluster engine version is being upgraded.

Viewing DB instance status in an Aurora cluster

The status of a DB instance in an Aurora cluster indicates the health of the DB instance. You can view the status of a DB instance in a cluster by using the Amazon RDS console, the AWS CLI command [describe-db-instances](#), or the API operation [DescribeDBInstances](#).

Note

Amazon RDS also uses another status called *maintenance status*, which is shown in the **Maintenance** column of the Amazon RDS console. This value indicates the status of any maintenance patches that need to be applied to a DB instance. Maintenance status is independent of DB instance status. For more information on *maintenance status*, see [Applying updates for a DB cluster \(p. 445\)](#).

Find the possible status values for DB instances in the following table. This table also shows whether you will be billed for the DB instance and storage, billed only for storage, or not billed. For all DB instance statuses, you are always billed for backup usage.

DB instance status	Billed	Description
available	Billed	The DB instance is healthy and available.
backing-up	Billed	The DB instance is currently being backed up.
backtracking	Billed	The DB instance is currently being backtracked. This status only applies to Aurora MySQL.
configuring-enhanced-monitoring	Billed	Enhanced Monitoring is being enabled or disabled for this DB instance.
configuring-iam-database-auth	Billed	AWS Identity and Access Management (IAM) database authentication is being enabled or disabled for this DB instance.
configuring-log-exports	Billed	Publishing log files to Amazon CloudWatch Logs is being enabled or disabled for this DB instance.
converting-to-vpc	Billed	The DB instance is being converted from a DB instance that is not in an Amazon Virtual Private Cloud (Amazon VPC) to a DB instance that is in an Amazon VPC.
creating	Not billed	The DB instance is being created. The DB instance is inaccessible while it is being created.
deleting	Not billed	The DB instance is being deleted.
failed	Not billed	The DB instance has failed and Amazon RDS can't recover it. Perform a point-in-time restore to the latest restorable time of the DB instance to recover the data.
inaccessible-encryption-credentials	Not billed	The AWS KMS key used to encrypt or decrypt the DB instance can't be accessed or recovered.
inaccessible-encryption-credentials-recoverable	Billed for storage	The KMS key used to encrypt or decrypt the DB instance can't be accessed. However, if the KMS key is active, restarting the DB instance can recover it. For more information, see Enabling encryption for an Amazon Aurora DB cluster (p. 1710) .

DB instance status	Billed	Description
incompatible-network	Not billed	Amazon RDS is attempting to perform a recovery action on a DB instance but can't do so because the VPC is in a state that prevents the action from being completed. This status can occur if, for example, all available IP addresses in a subnet are in use and Amazon RDS can't get an IP address for the DB instance.
incompatible-option-group	Billed	Amazon RDS attempted to apply an option group change but can't do so, and Amazon RDS can't roll back to the previous option group state. For more information, check the Recent Events list for the DB instance. This status can occur if, for example, the option group contains an option such as TDE and the DB instance doesn't contain encrypted information.
incompatible-parameters	Billed	Amazon RDS can't start the DB instance because the parameters specified in the DB instance's DB parameter group aren't compatible with the DB instance. Revert the parameter changes or make them compatible with the DB instance to regain access to your DB instance. For more information about the incompatible parameters, check the Recent Events list for the DB instance.
incompatible-restore	Not billed	Amazon RDS can't do a point-in-time restore. Common causes for this status include using temp tables or using MyISAM tables with MySQL.
insufficient-capacity		Amazon RDS can't create your instance because sufficient capacity isn't currently available. To create your DB instance in the same AZ with the same instance type, delete your DB instance, wait a few hours, and try to create again. Alternatively, create a new instance using a different instance class or AZ.
maintenance	Billed	Amazon RDS is applying a maintenance update to the DB instance. This status is used for instance-level maintenance that RDS schedules well in advance.
modifying	Billed	The DB instance is being modified because of a customer request to modify the DB instance.
moving-to-vpc	Billed	The DB instance is being moved to a new Amazon Virtual Private Cloud (Amazon VPC).
rebooting	Billed	The DB instance is being rebooted because of a customer request or an Amazon RDS process that requires the rebooting of the DB instance.
resetting-master-credentials	Billed	The master credentials for the DB instance are being reset because of a customer request to reset them.
renaming	Billed	The DB instance is being renamed because of a customer request to rename it.
restore-error	Billed	The DB instance encountered an error attempting to restore to a point-in-time or from a snapshot.
starting	Billed for storage	The DB instance is starting.

DB instance status	Billed	Description
stopped	Billed for storage	The DB instance is stopped.
stopping	Billed for storage	The DB instance is being stopped.
storage-full	Billed	The DB instance has reached its storage capacity allocation. This is a critical status, and we recommend that you fix this issue immediately. To do so, scale up your storage by modifying the DB instance. To avoid this situation, set Amazon CloudWatch alarms to warn you when storage space is getting low.
storage-optimization	Billed	Your DB instance is being modified to change the storage size or type. The DB instance is fully operational. However, while the status of your DB instance is storage-optimization , you can't request any changes to the storage of your DB instance. The storage optimization process is usually short, but can sometimes take up to and even beyond 24 hours.
upgrading	Billed	The database engine version is being upgraded.

Viewing Amazon Aurora recommendations

Amazon Aurora provides automated recommendations for database resources, such as DB instances, DB clusters, and DB cluster parameter groups. These recommendations provide best practice guidance by analyzing DB cluster configuration, DB instance configuration, usage, and performance data.

You can find examples of these recommendations in the following table.

Type	Description	Recommendation	Additional information
Nondefault custom memory parameters	Your DB parameter group sets memory parameters that diverge too much from the default values.	Settings that diverge too much from the default values can cause poor performance and errors. We recommend setting custom memory parameters to their default values in the DB parameter group used by the DB instance.	Working with DB parameter groups and DB cluster parameter groups (p. 339)
Change buffering enabled for a MySQL DB instance	Your DB parameter group has change buffering enabled.	Change buffering allows a MySQL DB instance to defer some writes necessary to maintain secondary indexes. This configuration can improve performance slightly, but it can create a large delay in crash recovery. During crash recovery, the secondary index must be brought up to date. So, the benefits of change buffering are outweighed by the potentially very long crash recovery events. We recommend disabling change buffering.	Best practices for configuring parameters for Amazon RDS for MySQL, part 1: Parameters related to performance on the AWS Database Blog
Logging to table	Your DB parameter group sets logging output to <code>TABLE</code> .	Setting logging output to <code>TABLE</code> uses more storage than setting this parameter to <code>FILE</code> . To avoid reaching the storage limit, we recommend setting the logging output parameter to <code>FILE</code> .	Aurora MySQL database log files (p. 700)
DB cluster with one DB instance	Your DB cluster only contains one DB instance.	For improved performance and availability, we recommend adding another DB instance with the same DB instance class in a different Availability Zone.	High availability for Amazon Aurora (p. 68)
DB cluster in one Availability Zone	Your DB cluster has all of its DB instances in the same Availability Zone.	For improved availability, we recommend adding another DB instance with the same DB instance class in a different Availability Zone.	High availability for Amazon Aurora (p. 68)
DB cluster outdated	Your DB cluster is running an older engine version.	We recommend that you keep your DB cluster at the most current minor version because it includes the latest security and functionality fixes. Unlike major version upgrades, minor version upgrades include only changes	Maintaining an Amazon Aurora DB cluster (p. 443)

Type	Description	Recommendation	Additional information
		that are backward-compatible with previous minor versions (of the same major version) of the DB engine. We recommend that you upgrade to a recent engine version	
DB cluster with different parameter groups	Your DB cluster has different DB parameter groups assigned to its DB instances.	Using different parameter groups can cause incompatibilities between the DB instances. To avoid problems and for easier maintenance, we recommend using the same parameter group for all of the DB instances in the DB cluster.	Working with DB parameter groups and DB cluster parameter groups (p. 339)
DB cluster with different DB instance classes	Your DB cluster has DB instances that use different DB instance classes.	Using different DB instance classes for DB instances can cause problems. For example, performance might suffer if a less powerful DB instance class is promoted to replace a more powerful DB instance class. To avoid problems and for easier maintenance, we recommend using the same DB instance class for all of the DB instances in the DB cluster.	Aurora Replicas (p. 70)

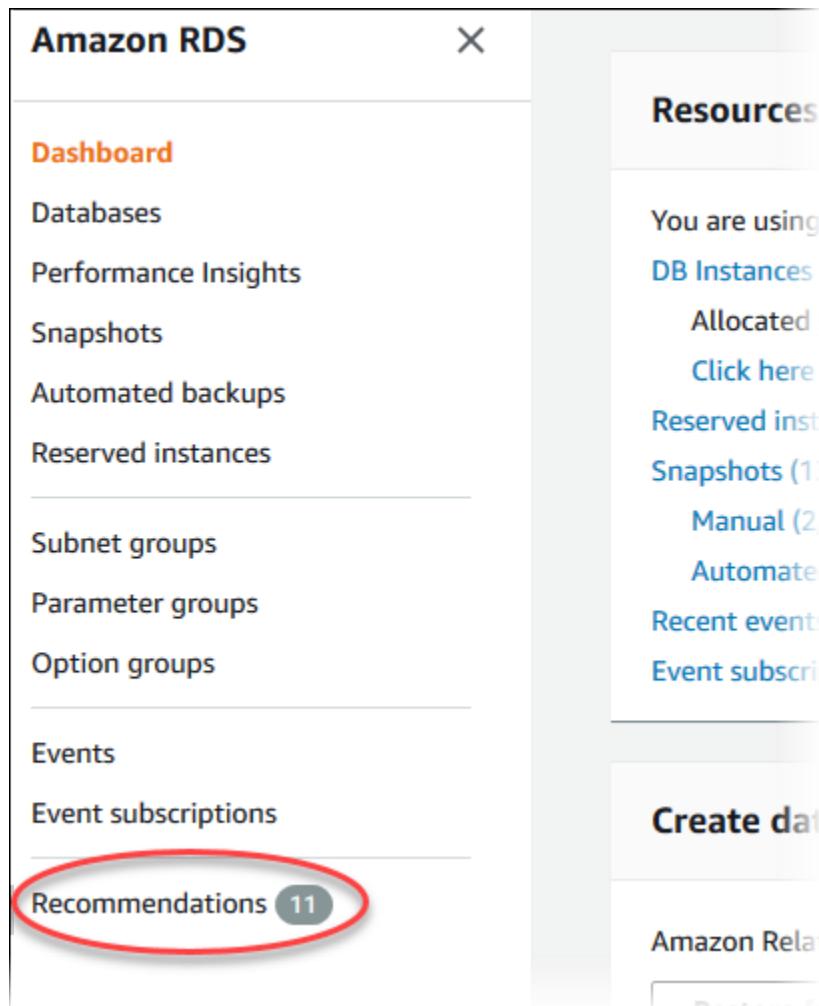
Amazon Aurora generates recommendations for a resource when the resource is created or modified. Amazon Aurora also periodically scans your resources and generates recommendations.

Responding to Amazon Aurora recommendations

You can find recommendations in the AWS Management Console. You can perform the recommended action immediately, schedule it for the next maintenance window, or dismiss it.

To respond to Amazon Aurora recommendations

1. Sign in to the AWS Management Console and open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. In the navigation pane, choose **Recommendations**.



The Recommendations page appears.

This screenshot shows the 'Recommendations' page. At the top, there are four tabs: Active (2), Dismissed (0), Scheduled (0), and Applied (0). The 'Active (2)' tab is selected. Below the tabs, there are two recommendation items, each with a disclosure arrow:

- ▶ Heterogeneous instance classes in DB cluster (1)
DB clusters which have heterogeneous instance types [Info](#)
- ▶ Heterogeneous parameters for DB cluster (1)
DB clusters which have heterogeneous parameter groups assigned to its instances [Info](#)

3. On the **Recommendations** page, choose one of the following:
 - **Active** – Shows the current recommendations that you can apply, dismiss, or schedule.
 - **Dismissed** – Shows the recommendations that have been dismissed. When you choose **Dismissed**, you can apply these dismissed recommendations.
 - **Scheduled** – Shows the recommendations that are scheduled but not yet applied. These recommendations will be applied in the next scheduled maintenance window.
 - **Applied** – Shows the recommendations that are currently applied.

From any list of recommendations, you can open a section to view the recommendations in that section.

Recommendations

Active (2) Dismissed (0) Scheduled (0) Applied (0)

▼ Heterogeneous instance classes in DB cluster (1)
DB clusters which have heterogeneous instance types [Info](#)

DB clusters

Dismiss Schedule Apply now

Resource	Recommendation	Affected resources
cluster-1	Your DB cluster cluster-1 has DB instances that use different DB instance classes. Using different DB instance classes for DB instances can cause problems. To avoid problems and for easier maintenance, we recommend using db.t3.small for all of the DB instances in the DB cluster.	cluster-1-reader

▶ Heterogeneous parameters for DB cluster (1)
DB clusters which have heterogeneous parameter groups assigned to its instances [Info](#)

To configure preferences for displaying recommendations in each section, choose the **Preferences** icon.

Recommendations

Active (2) Dismissed (0) Scheduled (0) Applied (0)

▼ Heterogeneous instance classes in DB cluster (1)
DB clusters which have heterogeneous instance types [Info](#)

DB clusters

Dismiss Schedule Apply now

Resource	Recommendation	Affected resources
cluster-1	Your DB cluster cluster-1 has DB instances that use different DB instance classes. Using different DB instance classes for DB instances can cause problems. To avoid problems and for easier maintenance, we recommend using db.t3.small for all of the DB instances in the DB cluster.	cluster-1-reader

▶ Heterogeneous parameters for DB cluster (1)
DB clusters which have heterogeneous parameter groups assigned to its instances [Info](#)

From the **Preferences** window that appears, you can set display options. These options include the visible columns and the number of recommendations to display on the page.

4. Manage your active recommendations:

- Choose **Active** and open one or more sections to view the recommendations in them.
- Choose one or more recommendations and choose **Apply now** (to apply them immediately), **Schedule** (to apply them in next maintenance window), or **Dismiss**.

If the **Apply now** button appears for a recommendation but is unavailable (grayed out), the DB instance is not available. You can apply recommendations immediately only if the DB instance status is **available**. For example, you can't apply recommendations immediately to the DB

instance if its status is **modifying**. In this case, wait for the DB instance to be available and then apply the recommendation.

If the **Apply now** button doesn't appear for a recommendation, you can't apply the recommendation using the **Recommendations** page. You can modify the DB instance to apply the recommendation manually.

For more information about modifying a DB cluster, see [Modifying an Amazon Aurora DB cluster \(p. 372\)](#).

Note

When you choose **Apply now**, a brief DB instance outage might result.

Viewing metrics in the Amazon RDS console

Amazon RDS integrates with Amazon CloudWatch to display a variety of Aurora DB cluster metrics in the RDS console. Some metrics are apply at the cluster level, whereas others apply at the instance level. For descriptions of the instance-level and cluster-level metrics, see [Metrics reference for Amazon Aurora \(p. 633\)](#).

The **Monitoring** tab for your Aurora DB cluster shows the following categories of metrics:

- **CloudWatch** – Shows the Amazon CloudWatch metrics for Aurora that you can access in the RDS console. You can also access these metrics in the CloudWatch console. Each metric includes a graph that shows the metric monitored over a specific time span. For a list of CloudWatch metrics, see [Amazon CloudWatch metrics for Amazon Aurora \(p. 633\)](#).
- **Enhanced monitoring** – Shows a summary of operating-system metrics when your Aurora DB cluster has turned on Enhanced Monitoring. RDS delivers the metrics from Enhanced Monitoring to your Amazon CloudWatch Logs account. Each OS metric includes a graph showing the metric monitored over a specific time span. For an overview, see [Monitoring OS metrics with Enhanced Monitoring \(p. 626\)](#). For a list of Enhanced Monitoring metrics, see [OS metrics in Enhanced Monitoring \(p. 660\)](#).
- **OS Process list** – Shows details for each process running in your DB cluster.
- **Performance Insights** – Opens the Amazon RDS Performance Insights dashboard for a DB instance in your Aurora DB cluster. Performance Insights isn't supported at the cluster level. For an overview of Performance Insights, see [Monitoring DB load with Performance Insights on Amazon Aurora \(p. 573\)](#). For a list of Performance Insights metrics, see [Amazon CloudWatch metrics for Performance Insights \(p. 652\)](#).

To view metrics for your Aurora DB cluster in the RDS console

1. Sign in to the AWS Management Console and open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. In the navigation pane, choose **Databases**.
3. Choose the name of the Aurora DB cluster that you want to monitor.

The database page appears. The following example shows an Amazon Aurora PostgreSQL database named `apga`.

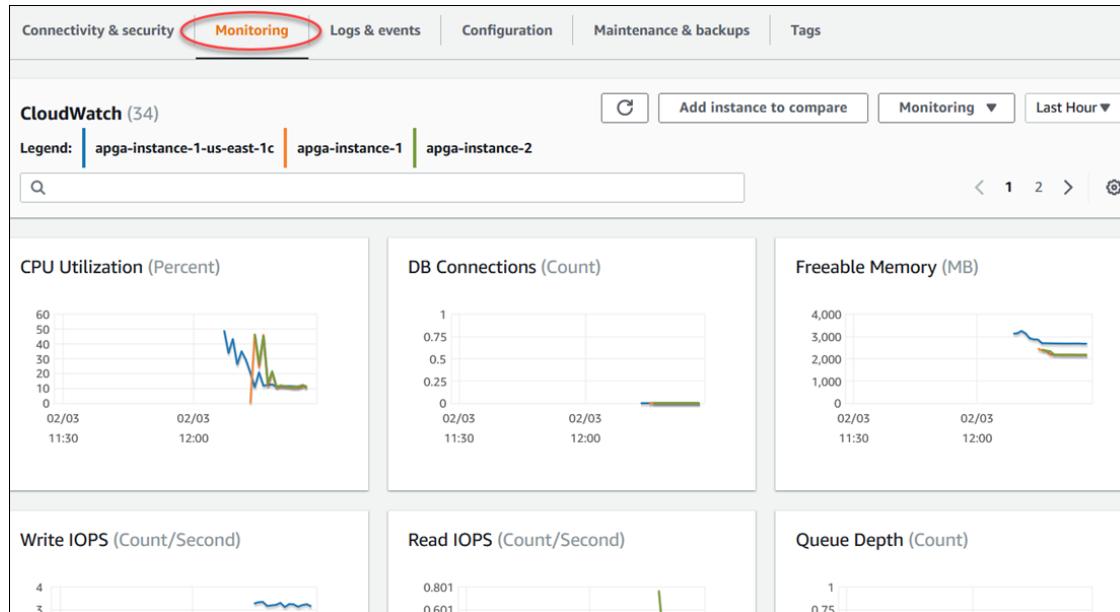
The screenshot shows the 'Databases' section of the Amazon RDS console for the 'apga' cluster. The main table lists the following database instances:

DB identifier	DB cluster identifier	Role	Engine
apga	apga	Regional cluster	Aurora PostgreSQL
apga-instance-1-us-east-1c	apga	Writer instance	Aurora PostgreSQL
apga-instance-1	apga	Reader instance	Aurora PostgreSQL
apga-instance-2	apga	Reader instance	Aurora PostgreSQL

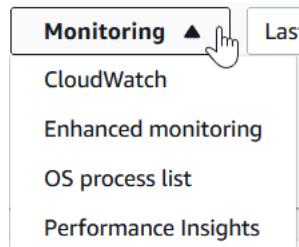
Below the table, there are tabs for **Connectivity & security**, **Monitoring**, **Logs & events**, **Configuration**, **Maintenance & backups**, and **Tags**. The **Monitoring** tab is highlighted in orange.

4. Scroll down and choose **Monitoring**.

The monitoring section appears. By default, CloudWatch metrics are shown. For descriptions of these metrics, see [Amazon CloudWatch metrics for Amazon Aurora \(p. 633\)](#).

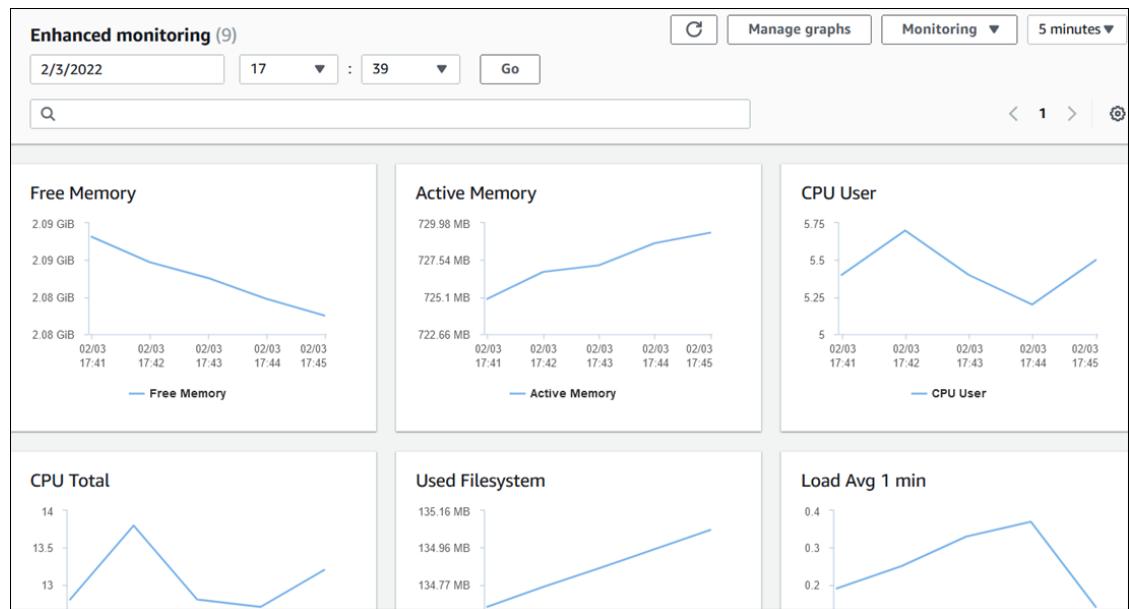


5. Choose **Monitoring** to see the metric categories.



6. Choose the category of metrics that you want to see.

The following example shows Enhanced Monitoring metrics. For descriptions of these metrics, see [OS metrics in Enhanced Monitoring \(p. 660\)](#).



Tip

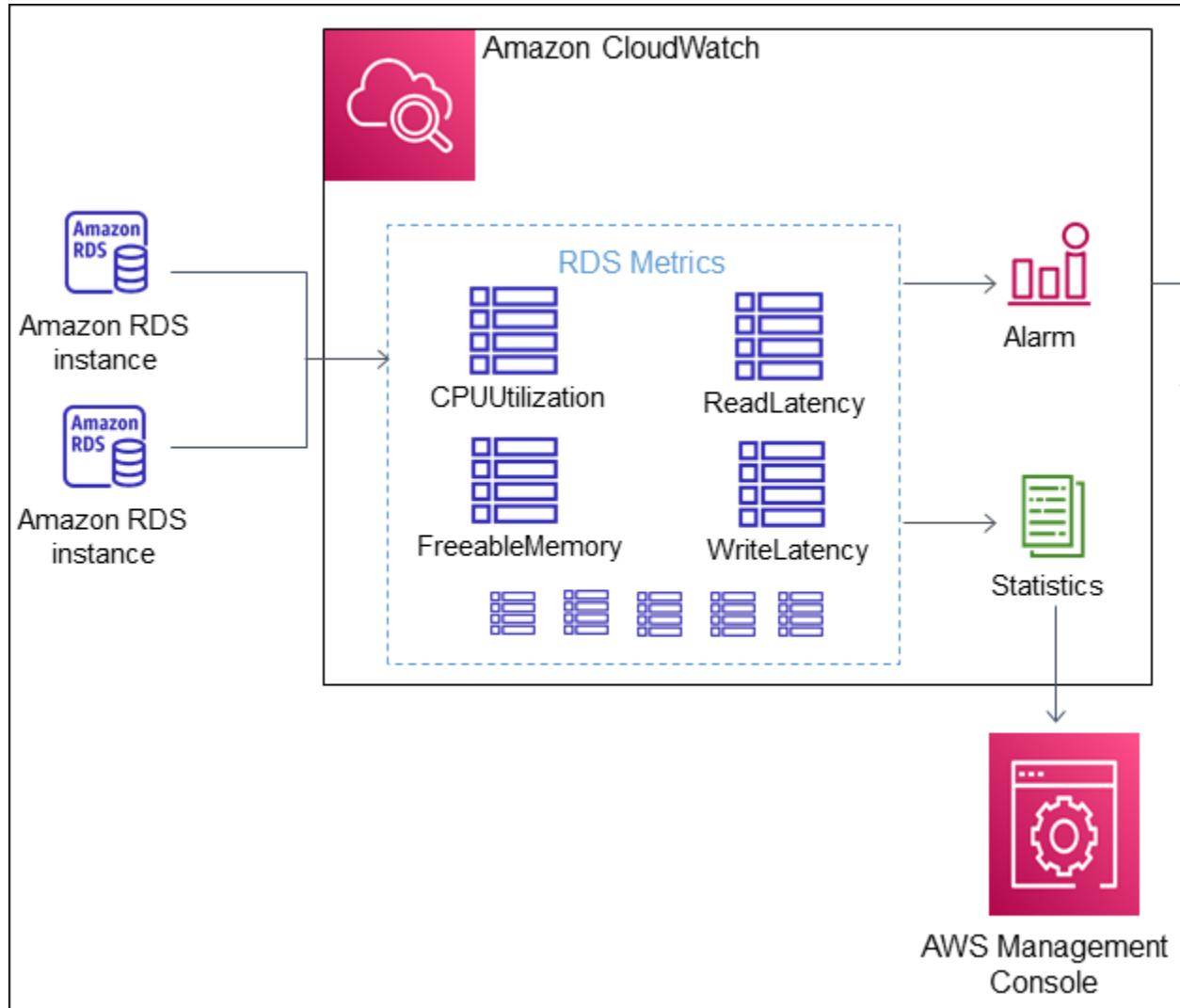
To choose the time range of the metrics represented by the graphs, you can use the time range list.

To bring up a more detailed view, you can choose any graph. You can also apply metric-specific filters to the data.

Monitoring Amazon Aurora metrics with Amazon CloudWatch

Amazon CloudWatch is a metrics repository. The repository collects and processes raw data from Amazon Aurora into readable, near real-time metrics. For a complete list of Amazon Aurora metrics sent to CloudWatch, see [Amazon RDS dimensions and metrics](#) in the *Amazon CloudWatch User Guide*.

By default, Amazon Aurora automatically sends metric data to CloudWatch in 1-minute periods. Data points with a period of 60 seconds (1 minute) are available for 15 days. This means that you can access historical information and see how your web application or service is performing.



For more information about CloudWatch, see [What is Amazon CloudWatch?](#) in the *Amazon CloudWatch User Guide*. For more information about CloudWatch metrics retention, see [Metrics retention](#).

Note

If you are using Amazon RDS Performance Insights, additional metrics are available. For more information, see [Amazon CloudWatch metrics for Performance Insights \(p. 652\)](#).

Topics

- [Viewing DB instance metrics in the CloudWatch console and CLI \(p. 568\)](#)
- [Creating CloudWatch alarms to monitor Amazon Aurora \(p. 571\)](#)

Viewing DB instance metrics in the CloudWatch console and CLI

Following, you can find details about how to view metrics for your DB instance using CloudWatch. For information on monitoring metrics for your DB instance's operating system in real time using CloudWatch Logs, see [Monitoring OS metrics with Enhanced Monitoring \(p. 626\)](#).

When you use Amazon Aurora resources, Amazon Aurora sends metrics and dimensions to Amazon CloudWatch every minute. You can use the following procedures to view the metrics for Amazon Aurora in the CloudWatch console and CLI.

Console

To view metrics using the Amazon CloudWatch console

Metrics are grouped first by the service namespace, and then by the various dimension combinations within each namespace.

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. If necessary, change the AWS Region. From the navigation bar, choose the AWS Region where your AWS resources are. For more information, see [Regions and endpoints](#).
3. In the navigation pane, choose **Metrics** and then **All metrics**.

The screenshot shows the CloudWatch Metrics console interface. At the top, there are tabs: 'Browse' (highlighted in orange), 'Query', 'Graphed metrics', 'Options', and 'Source'. To the right of these are buttons for 'Add math ▾' and 'Graph with SQL'. Below the tabs, the title 'Metrics (1301)' is followed by a 'Info' link. A dropdown menu shows 'N. Virginia ▾' and a search bar with placeholder text 'Search for any metric, dimension or resource id'. The main area displays a grid of service namespaces and their metric counts:

EBS	9	EC2	17	Events
Lambda	26	Logs	35	RDS
S3	8	SSM Run Command	3	Usage

A red circle highlights the 'RDS' namespace in the third row. To the right of the grid, there are additional buttons for 'Graph with SQL' and 'Graph'.

4. Scroll down and choose the **RDS** metric namespace.

The page displays the Amazon Aurora dimensions. For descriptions of these dimensions, see [Amazon CloudWatch dimensions for Aurora \(p. 649\)](#).

The screenshot shows the CloudWatch Metrics console with the 'Metrics' tab selected. The top navigation bar includes 'Browse', 'Query', 'Graphed metrics', 'Options', 'Source', 'Add math ▾', and 'Graph with SQL'. A search bar at the top right says 'Search for any metric, dimension or resource id'. Below the navigation is a breadcrumb trail: 'N. Virginia ▾ > All > RDS'. The main area displays several metric categories with counts: 'DBClusterIdentifier, Role' (153), 'DbClusterIdentifier, EngineName' (6), 'DBClusterIdentifier' (114), 'Per-Database Metrics' (332), 'By Database Class' (191), and 'By Database Engine' (191). A large 'Across All Databases' box shows a count of 114.

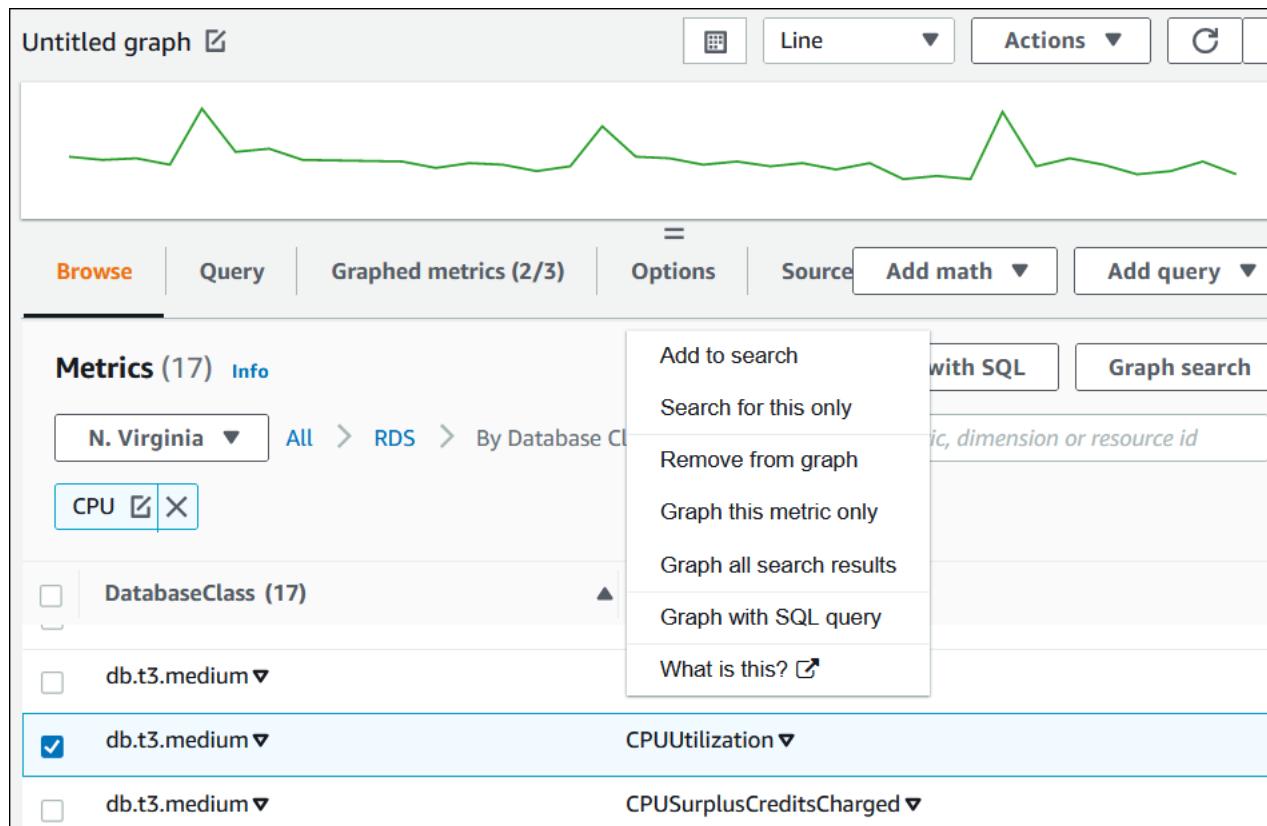
- Choose a metric dimension, for example **By Database Class**.

The screenshot shows the CloudWatch Metrics console with the 'Metrics' tab selected. The top navigation bar includes 'Browse', 'Query', 'Graphed metrics (1)', 'Options', 'Source', 'Add math ▾', and 'Add query ▾'. A search bar at the top right says 'Search for any metric, dimension or resource id'. Below the navigation is a breadcrumb trail: 'N. Virginia ▾ > All > RDS > By Database Class'. The main area lists metrics under 'DatabaseClass (191)'. Each item has a checkbox on the left and a dropdown arrow on the right. The columns are labeled 'Metric name' and 'Dimension'. The listed metrics are: AbortedClients, ActiveTransactions, and Aurora_pq_request_attempted, all associated with the db.r6g.large dimension.

- Do any of the following actions:

- To sort the metrics, use the column heading.
- To graph a metric, select the check box next to the metric.
- To filter by resource, choose the resource ID, and then choose **Add to search**.
- To filter by metric, choose the metric name, and then choose **Add to search**.

The following example filters on the **db.t3.medium** class and graphs the **CPUUtilization** metric.



AWS CLI

To obtain metric information by using the AWS CLI, use the CloudWatch command [list-metrics](#). In the following example, you list all metrics in the AWS/RDS namespace.

```
aws cloudwatch list-metrics --namespace AWS/RDS
```

To obtain metric statistics, use the command [get-metric-statistics](#). The following command gets CPUUtilization statistics for instance `my-instance` over the specific 24-hour period, with a 5-minute granularity.

Example

For Linux, macOS, or Unix:

```
aws cloudwatch get-metric-statistics --namespace AWS/RDS \
--metric-name CPUUtilization \
--start-time 2021-12-15T00:00:00Z \
--end-time 2021-12-16T00:00:00Z \
--period 360 \
--statistics Minimum \
--dimensions Name=DBInstanceIdentifier,Value=my-instance
```

For Windows:

```
aws cloudwatch get-metric-statistics --namespace AWS/RDS ^
--metric-name CPUUtilization ^
```

```
--start-time 2021-12-15T00:00:00Z ^
--end-time 2021-12-16T00:00:00Z ^
--period 360 ^
--statistics Minimum ^
--dimensions Name=DBInstanceIdentifier,Value=my-instance
```

Sample output appears as follows:

```
{
    "Datapoints": [
        {
            "Timestamp": "2021-12-15T18:00:00Z",
            "Minimum": 8.7,
            "Unit": "Percent"
        },
        {
            "Timestamp": "2021-12-15T23:54:00Z",
            "Minimum": 8.12486458559024,
            "Unit": "Percent"
        },
        {
            "Timestamp": "2021-12-15T17:24:00Z",
            "Minimum": 8.841666666666667,
            "Unit": "Percent"
        },
        ...
        {
            "Timestamp": "2021-12-15T22:48:00Z",
            "Minimum": 8.366248354248954,
            "Unit": "Percent"
        }
    ],
    "Label": "CPUUtilization"
}
```

For more information, see [Getting statistics for a metric](#) in the *Amazon CloudWatch User Guide*.

Creating CloudWatch alarms to monitor Amazon Aurora

You can create a CloudWatch alarm that sends an Amazon SNS message when the alarm changes state. An alarm watches a single metric over a time period that you specify. The alarm can also perform one or more actions based on the value of the metric relative to a given threshold over a number of time periods. The action is a notification sent to an Amazon SNS topic or Amazon EC2 Auto Scaling policy.

Alarms invoke actions for sustained state changes only. CloudWatch alarms don't invoke actions simply because they are in a particular state. The state must have changed and have been maintained for a specified number of time periods. The following procedures show how to create alarms for Amazon RDS.

Note

For Aurora, use WRITER or READER role metrics to set up alarms instead of relying on metrics for specific DB instances. Aurora DB instance roles can change roles over time. You can find these role-based metrics in the CloudWatch console.

Aurora Auto Scaling automatically sets alarms based on READER role metrics. For more information about Aurora Auto Scaling, see [Using Amazon Aurora Auto Scaling with Aurora replicas \(p. 427\)](#).

To set alarms using the CloudWatch console

1. Sign in to the AWS Management Console and open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.

2. Choose **Alarms** and then choose **Create Alarm**. Doing this launches the Create Alarm Wizard.
3. Choose **RDS Metrics** and scroll through the Amazon RDS metrics to find the metric that you want to place an alarm on. To display just Amazon RDS metrics, search for the identifier of your resource. Choose the metric to create an alarm on and then choose **Next**.
4. Enter **Name**, **Description**, and **Whenever** values for the metric.
5. If you want CloudWatch to send you an email when the alarm state is reached, for **Whenever this alarm**, choose **State is ALARM**. For **Send notification to**, choose an existing SNS topic. If you choose **Create topic**, you can set the name and email addresses for a new email subscription list. This list is saved and appears in the field for future alarms.

Note

If you use **Create topic** to create a new Amazon SNS topic, the email addresses must be verified before they receive notifications. Emails are only sent when the alarm enters an alarm state. If this alarm state change happens before the email addresses are verified, the addresses don't receive a notification.

6. Preview the alarm that you're about to create in the **Alarm Preview** area, and then choose **Create Alarm**.

To set an alarm using the AWS CLI

- Call [put-metric-alarm](#). For more information, see [AWS CLI Command Reference](#).

To set an alarm using the CloudWatch API

- Call [PutMetricAlarm](#). For more information, see [Amazon CloudWatch API Reference](#)

Monitoring DB load with Performance Insights on Amazon Aurora

Performance Insights expands on existing Amazon Aurora monitoring features to illustrate your cluster performance and help you analyze any issues that affect it. With the Performance Insights dashboard, you can visualize the database load and filter the load by waits, SQL statements, hosts, or users.

Topics

- [Overview of Performance Insights \(p. 573\)](#)
- [Enabling and disabling Performance Insights \(p. 577\)](#)
- [Enabling the Performance Schema for Performance Insights on Aurora MySQL \(p. 580\)](#)
- [Configuring access policies for Performance Insights \(p. 582\)](#)
- [Analyzing metrics with the Performance Insights dashboard \(p. 585\)](#)
- [Retrieving metrics with the Performance Insights API \(p. 607\)](#)
- [Logging Performance Insights calls using AWS CloudTrail \(p. 621\)](#)

Overview of Performance Insights

By default, Performance Insights is enabled in the console create wizard for Amazon RDS engines. If you have more than one database on a DB instance, Performance Insights aggregates performance data.

You can find an overview of Performance Insights in the following video.

[Using Performance Insights to Analyze Performance of Amazon Aurora PostgreSQL](#)

Topics

- [Database load \(p. 573\)](#)
- [Maximum CPU \(p. 576\)](#)
- [Amazon Aurora DB engine support for Performance Insights \(p. 576\)](#)
- [AWS Region support for Performance Insights \(p. 577\)](#)
- [Cost of Performance Insights \(p. 577\)](#)

Database load

Database load (DB load) measures the level of activity in your database. The key metric in Performance Insights is `DBLoad`, which is collected every second.

Topics

- [Active sessions \(p. 573\)](#)
- [Average active sessions \(p. 574\)](#)
- [Average active executions \(p. 574\)](#)
- [Dimensions \(p. 575\)](#)

Active sessions

A *database session* represents an application's dialogue with a relational database. An *active session* is a connection that has submitted work to the DB engine and is waiting for a response.

A session is active when it's either running on CPU or waiting for a resource to become available so that it can proceed. For example, an active session might wait for a page to be read into memory, and then consume CPU while it reads data from the page.

Average active sessions

The *average active sessions (AAS)* is the unit for the `DBLoad` metric in Performance Insights. To get the average active sessions, Performance Insights samples the number of sessions concurrently running a query. The AAS is the total number of sessions divided by the total number of samples for a specific time period. The following table shows 5 consecutive samples of a running query.

Sample	Number of sessions running query	AAS	Calculation
1	2	2	2 sessions / 1 sample
2	0	1	2 sessions / 2 samples
3	4	2	6 sessions / 3 samples
4	0	1.5	6 sessions / 4 samples
5	4	2	10 sessions / 5 samples

In the preceding example, the DB load for the time interval was 2 AAS. This measurement means that, on average, 2 sessions were active at a time during the time period when the 5 samples were taken.

An analogy for DB load is activity in a warehouse. Suppose that the warehouse employs 100 workers. If 1 order comes in, 1 worker fulfills the order while the other workers are idle. If 100 orders come in, all 100 workers fulfill orders simultaneously. If you periodically sample how many workers are active over a given time period, you can calculate the average number of active workers. The calculation shows that, on average, N workers are busy fulfilling orders at any given time. If the average was 50 workers yesterday and 75 workers today, the activity level in the warehouse increased. In the same way, DB load increases as session activity increases.

Average active executions

The *average active executions (AAE)* per second is related to AAS. To calculate the AAE, Performance Insights divides the total execution time of a query by the time interval. The following table shows the AAE calculation for the same query in the preceding table.

Elapsed time (sec)	Total execution time (sec)	AAE	Calculation
60	120	2	120 execution seconds/60 elapsed seconds
120	120	1	120 execution seconds/120 elapsed seconds
180	380	2.11	380 execution seconds/180 elapsed seconds
240	380	1.58	380 execution seconds/240 elapsed seconds

Elapsed time (sec)	Total execution time (sec)	AAE	Calculation
300	600	2	600 execution seconds/300 elapsed seconds

In most cases, the AAS and AAE for a query are approximately the same. However, because the inputs to the calculations are different data sources, the calculations often vary slightly.

Dimensions

The `db.load` metric is different from the other time-series metrics because you can break it into subcomponents called *dimensions*. You can think of dimensions as "slice by" categories for the different characteristics of the DBLoad metric.

When you are diagnosing performance issues, the following dimensions are often the most useful:

Topics

- [Wait events \(p. 575\)](#)
- [Top SQL \(p. 576\)](#)

For a complete list of dimensions for the Aurora engines, see [DB load sliced by dimensions \(p. 588\)](#).

Wait events

A *wait event* causes a SQL statement to wait for a specific event to happen before it can continue running. Wait events are an important dimension, or category, for DB load because they indicate where work is impeded.

Every active session is either running on the CPU or waiting. For example, sessions consume CPU when they search memory for a buffer, perform a calculation, or run procedural code. When sessions aren't consuming CPU, they might be waiting for a memory buffer to become free, a data file to be read, or a log to be written to. The more time that a session waits for resources, the less time it runs on the CPU.

When you tune a database, you often try to find out the resources that sessions are waiting for. For example, two or three wait events might account for 90 percent of DB load. This measure means that, on average, active sessions are spending most of their time waiting for a small number of resources. If you can find out the cause of these waits, you can attempt a solution.

Consider the analogy of a warehouse worker. An order comes in for a book. The worker might be delayed in fulfilling the order. For example, a different worker might be currently restocking the shelves, a trolley might not be available. Or the system used to enter the order status might be slow. The longer the worker waits, the longer it takes to fulfill the order. Waiting is a natural part of the warehouse workflow, but if wait time becomes excessive, productivity decreases. In the same way, repeated or lengthy session waits can degrade database performance. For more information, see [Tuning with wait events for Aurora PostgreSQL](#) and [Tuning with wait events for Aurora MySQL](#) in the *Amazon Aurora User Guide*.

Wait events vary by DB engine:

- For a list of the common wait events for Aurora MySQL, see [Aurora MySQL wait events \(p. 1063\)](#). To learn how to tune using these wait events, see [Tuning Aurora MySQL with wait events and thread states \(p. 837\)](#).
- For information about all MySQL wait events, see [Wait Event Summary Tables](#) in the MySQL documentation.

- For a list of common wait events for Aurora PostgreSQL, see [Amazon Aurora PostgreSQL wait events \(p. 1570\)](#). To learn how to tune using these wait events, see [Tuning with wait events for Aurora PostgreSQL \(p. 1376\)](#).
- For information about all PostgreSQL wait events, see [PostgreSQL Wait Events](#) in the PostgreSQL documentation.

Top SQL

Where wait events show bottlenecks, top SQL shows which queries are contributing the most to DB load. For example, many queries might be currently running on the database, but a single query might consume 99 percent of the DB load. In this case, the high load might indicate a problem with the query.

By default, the Performance Insights console displays top SQL queries that are contributing to the database load. The console also shows relevant statistics for each statement. To diagnose performance problems for a specific statement, you can examine its execution plan.

Maximum CPU

In the dashboard, the **Database load** chart collects, aggregates, and displays session information. To see whether active sessions are exceeding the maximum CPU, look at their relationship to the **Max vCPU** line. The **Max vCPU** value is determined by the number of vCPU (virtual CPU) cores for your DB instance. For Aurora Serverless v2, **Max vCPU** represents the estimated number of vCPUs.

If the DB load is often above the **Max vCPU** line, and the primary wait state is CPU, the CPU is overloaded. In this case, you might want to throttle connections to the instance, tune any SQL queries with a high CPU load, or consider a larger instance class. High and consistent instances of any wait state indicate that there might be bottlenecks or resource contention issues to resolve. This can be true even if the DB load doesn't cross the **Max vCPU** line.

Amazon Aurora DB engine support for Performance Insights

Following, you can find the Amazon Aurora DB engines that support Performance Insights.

Amazon Aurora DB engine	Supported engine versions when parallel query isn't turned on	Supported engine versions when parallel query is turned on	Instance class restrictions
Amazon Aurora MySQL-Compatible Edition	<p>Performance Insights is supported for the following engine versions:</p> <ul style="list-style-type: none"> 3.0 and higher 3 versions (compatible with MySQL 8.0) 2.04.2 and higher 2 versions (compatible with MySQL 5.7) 1.17.3 and higher 1 versions (compatible with MySQL 5.6) 	<p>Performance Insights with parallel query enabled is supported for the following engine versions:</p> <ul style="list-style-type: none"> 2.09.0 and higher 2 versions (compatible with MySQL 5.7) 1.23.0 and higher 1 versions (compatible with MySQL 5.6) 	<p>Performance Insights has the following engine class restrictions:</p> <ul style="list-style-type: none"> db.t2 – Not supported db.t3 – Not supported db.t4g – Supported only for 2.10.1 and higher 2 versions (compatible with MySQL 5.7)
Amazon Aurora PostgreSQL-	Performance Insights is supported for all engine versions.	N/A	N/A

Amazon Aurora DB engine	Supported engine versions when parallel query isn't turned on	Supported engine versions when parallel query is turned on	Instance class restrictions
Compatible Edition			

Note

Aurora Serverless doesn't support Performance Insights.

AWS Region support for Performance Insights

Performance Insights for Amazon Aurora is supported for all AWS Regions except the following:

- AWS GovCloud (US-East)
- AWS GovCloud (US-West)
- Asia Pacific (Jakarta)

Cost of Performance Insights

For cost information, see [Performance Insights Pricing](#).

Enabling and disabling Performance Insights

To use Performance Insights, enable it on your DB instance. If needed, you can disable it later. Enabling and disabling Performance Insights doesn't cause downtime, a reboot, or a failover.

Note

Performance Schema is an optional performance tool used by Aurora MySQL. If you turn Performance Schema on or off, you need to reboot. If you turn Performance Insights on or off, however, you don't need to reboot. For more information, see [Enabling the Performance Schema for Performance Insights on Aurora MySQL \(p. 580\)](#).

If you use Performance Insights together with Aurora Global Database, enable Performance Insights individually for the DB instances in each AWS Region. For details, see [Monitoring an Amazon Aurora global database with Amazon RDS Performance Insights \(p. 277\)](#).

The Performance Insights agent consumes limited CPU and memory on the DB host. When the DB load is high, the agent limits the performance impact by collecting data less frequently.

Console

In the console, you can enable or disable Performance Insights when you create or modify a new DB instance.

[Enabling or disabling Performance Insights when creating an instance](#)

When you create a new DB instance, enable Performance Insights by choosing **Enable Performance Insights** in the **Performance Insights** section. Or choose **Disable Performance Insights**.

To create a DB instance, follow the instructions for your DB engine in [Creating an Amazon Aurora DB cluster \(p. 125\)](#).

The following screenshot shows the **Performance Insights** section.

Description	Account	KMS key ID
Default master key that protects my RDS database volumes when no other key is defined	This account ([REDACTED])	[REDACTED]

If you choose **Enable Performance Insights**, you have the following options:

- **Retention** – The amount of time to retain Performance Insights data. Choose either 7 days (the default) or 2 years.
- **AWS KMS key** – Specify your AWS KMS key. Performance Insights encrypts all potentially sensitive data using your KMS key. Data is encrypted in flight and at rest. For more information, see [Configuring an AWS KMS policy for Performance Insights \(p. 584\)](#).

Enabling or disabling Performance Insights when modifying an instance

In the console, you can modify a DB instance to enable or disable Performance Insights using the console.

To enable or disable Performance Insights for a DB instance using the console

1. Sign in to the AWS Management Console and open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. Choose **Databases**.
3. Choose a DB instance, and choose **Modify**.
4. In the **Performance Insights** section, choose either **Enable Performance Insights** or **Disable Performance Insights**.

If you choose **Enable Performance Insights**, you have the following options:

- **Retention** – The amount of time to retain Performance Insights data. Choose either 7 days (the default) or 2 years. If you chose Long Term Retention (2 years) when you enable Performance Insights, All displays 2 years of data. If you chose Default (7 days) instead, All displays only the past week.
 - **AWS KMS key** – Specify your KMS key. Performance Insights encrypts all potentially sensitive data using your KMS key. Data is encrypted in flight and at rest. For more information, see [Encrypting Amazon Aurora resources \(p. 1709\)](#).
5. Choose **Continue**.

6. For **Scheduling of Modifications**, choose Apply immediately. If you choose Apply during the next scheduled maintenance window, your instance ignores this setting and enables Performance Insights immediately.
7. Choose **Modify instance**.

AWS CLI

When you use the [create-db-instance](#) AWS CLI command, enable Performance Insights by specifying `--enable-performance-insights`. Or disable Performance Insights by specifying `--no-enable-performance-insights`.

You can also specify these values using the following AWS CLI commands:

- [create-db-instance-read-replica](#)
- [modify-db-instance](#)
- [restore-db-instance-from-s3](#)

The following procedure describes how to enable or disable Performance Insights for a DB instance using the AWS CLI.

To enable or disable Performance Insights for a DB instance using the AWS CLI

- Call the [modify-db-instance](#) AWS CLI command and supply the following values:
 - `--db-instance-identifier` – The name of the DB instance.
 - `--enable-performance-insights` to enable or `--no-enable-performance-insights` to disable

The following example enables Performance Insights for `sample-db-instance`.

For Linux, macOS, or Unix:

```
aws rds modify-db-instance \
    --db-instance-identifier sample-db-instance \
    --enable-performance-insights
```

For Windows:

```
aws rds modify-db-instance ^
    --db-instance-identifier sample-db-instance ^
    --enable-performance-insights
```

When you enable Performance Insights, you can optionally specify the amount of time, in days, to retain Performance Insights data with the `--performance-insights-retention-period` option. Valid values are 7 (the default) or 731 (2 years).

The following example enables Performance Insights for `sample-db-instance` and specifies that Performance Insights data is retained for two years.

For Linux, macOS, or Unix:

```
aws rds modify-db-instance \
    --db-instance-identifier sample-db-instance \
    --enable-performance-insights \
```

```
--performance-insights-retention-period 731
```

For Windows:

```
aws rds modify-db-instance ^
--db-instance-identifier sample-db-instance ^
--enable-performance-insights ^
--performance-insights-retention-period 731
```

RDS API

When you create a new DB instance using the [CreateDBInstance](#) operation Amazon RDS API operation, enable Performance Insights by setting `EnablePerformanceInsights` to `True`. To disable Performance Insights, set `EnablePerformanceInsights` to `False`.

You can also specify the `EnablePerformanceInsights` value using the following API operations:

- [ModifyDBInstance](#)
- [CreateDBInstanceReadReplica](#)
- [RestoreDBInstanceFromS3](#)

When you enable Performance Insights, you can optionally specify the amount of time, in days, to retain Performance Insights data with the `PerformanceInsightsRetentionPeriod` parameter. Valid values are 7 (the default) or 731 (2 years).

Enabling the Performance Schema for Performance Insights on Aurora MySQL

The Performance Schema is an optional feature for monitoring Aurora MySQL runtime performance at a low level. You can use Performance insights with or without the Performance Schema. The Performance Schema is designed to have minimal impact on database performance.

Topics

- [Overview of the Performance Schema \(p. 580\)](#)
- [Options for enabling Performance Schema \(p. 581\)](#)
- [Configuring the Performance Schema for automatic management \(p. 581\)](#)

Overview of the Performance Schema

The Performance Schema monitors server events. In this context, an event is a server action that consumes time. Performance Schema events are distinct from binlog events and scheduler events.

The `PERFORMANCE_SCHEMA` storage engine collects event data using instrumentation in the database source code. The engine stores collected events in tables in the `performance_schema` database. You can query `performance_schema` just as you can query any other tables. For more information, see [MySQL Performance Schema](#) in [MySQL Reference Manual](#).

When the Performance Schema is enabled for Aurora MySQL, Performance Insights uses it to provide more detailed information. For example, Performance Insights displays DB load categorized by detailed wait events. You can use wait events to identify bottlenecks. Without the Performance Schema, Performance Insights reports user states such as inserting and sending, which don't help you identify bottlenecks.

Options for enabling Performance Schema

You have the following options for enabling the Performance Schema:

- Allow Performance Insights to manage required Performance Schema parameters automatically.

When you create an Aurora MySQL DB instance with Performance Insights enabled, the Performance Schema is also enabled. In this case, Performance Insights automatically manages your Performance Schema parameters.

For automatic management, the `performance_schema` must be set to 0 and the **Source** must be set to a value other than 0. By default, **Source** is `engine-default`. If you change the `performance_schema` value manually, and then later want to revert to automatic management, see [Configuring the Performance Schema for automatic management \(p. 581\)](#).

Important

When Performance Insights enables the Performance Schema, it doesn't change the parameter group values. However, the values are changed on the instances that are running. The only way to see the changed values is to run the `SHOW GLOBAL VARIABLES` command.

- Set the required Performance Schema parameters yourself.

For Performance Insights to list wait events, set all Performance Schema parameters as shown in the following table.

Parameter Name	Parameter Value
<code>performance_schema</code>	1 (the Source column has the value <code>engine-default</code>)
<code>performance-schema-consumer-events-waits-current</code>	ON
<code>performance-schema-instrument</code>	<code>wait/=%=ON</code>
<code>performance_schema_consumer_global_instrumentation</code>	ON
<code>performance_schema_consumer_thread_instrumentation</code>	ON

Note

If you enable or disable the Performance Schema, you must reboot the database. If you enable or disable Performance Insights, you don't need to reboot the database.

For more information, see [Performance Schema Command Options](#) and [Performance Schema Option and Variable Reference](#) in the MySQL documentation.

Configuring the Performance Schema for automatic management

The following table shows the difference in settings when Performance Insights is and isn't managing the Performance Schema.

Performance Insights isn't managing the Performance Schema	Performance Insights is managing the Performance Schema
<code>performance_schema</code> is 0 or 1	<code>performance_schema</code> is 0

Performance Insights isn't managing the Performance Schema	Performance Insights is managing the Performance Schema
The Source column is set to user	The Source column is set to system

To let Performance Insights manage the Performance Schema automatically

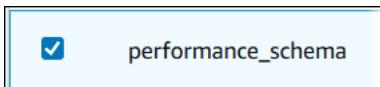
1. Sign in to the AWS Management Console and open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.

2. Choose **Parameter groups**.

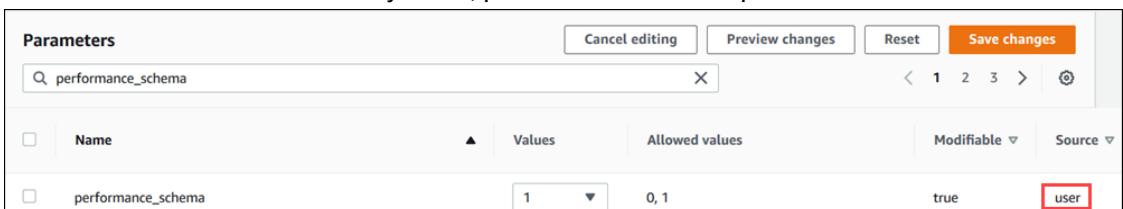
3. Select the name of the parameter group for your DB instance.

4. Enter **performance_schema** in the search bar.

5. Select the **performance_schema** parameter.



6. Check whether **Source** is **system** and **Values** is **0**. If so, Performance Insights is managing the Performance Schema automatically. If not, proceed to the next step.



7. Choose **Edit parameters**.

8. In **Values**, choose **0**.

9. Select **Reset**. When you reset, Aurora MySQL sets **Source** to **system** and **Values** to **0**.



The **Reset parameters in DB parameter group** page appears.

10. Select **Reset parameters**.

11. Restart the DB instance.

Important

Whenever you enable or disable the Performance Schema, you must restart the DB instance.

For more information about modifying instance parameters, see [Modifying parameters in a DB parameter group \(p. 347\)](#). For more information about the dashboard, see [Analyzing metrics with the Performance Insights dashboard \(p. 585\)](#). For more information about the MySQL performance schema, see [MySQL 8.0 Reference Manual](#).

Configuring access policies for Performance Insights

To access Performance Insights, you must have the appropriate permissions from AWS Identity and Access Management (IAM). You have the following options for granting access:

- Attach the `AmazonRDSFullAccess` managed policy to an IAM user or role.
- Create a custom IAM policy and attach it to an IAM user or role.

Also, if you specified a customer managed key when you turned on Performance Insights, make sure that users in your account have the `kms:Decrypt` and `kms:GenerateDataKey` permissions on the KMS key.

Attaching the `AmazonRDSFullAccess` policy to an IAM principal

`AmazonRDSFullAccess` is an AWS-managed policy that grants access to all of the Amazon RDS API operations. This policy does the following:

- Grants access to related services used by the Amazon RDS console. For example, this policy grants access to event notifications using Amazon SNS.
- Grants permissions needed for using Performance Insights.

If you attach `AmazonRDSFullAccess` to an IAM user or role, the recipient can use Performance Insights with other console features.

Creating a custom IAM policy for Performance Insights

For users who don't have full access with the `AmazonRDSFullAccess` policy, you can grant access to Performance Insights by creating or modifying a user-managed IAM policy. When you attach the policy to an IAM user or role, the recipient can use Performance Insights.

To create a custom policy

1. Open the IAM console at <https://console.aws.amazon.com/iam/>.
2. In the navigation pane, choose **Policies**.
3. Choose **Create policy**.
4. On the **Create Policy** page, choose the JSON tab.
5. Copy and paste the following text, replacing `us-east-1` with the name of your AWS Region and `111122223333` with your customer account number.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": "pi:*",  
            "Resource": "arn:aws:pi:us-east-1:111122223333:metrics/rds/*"  
        },  
        {  
            "Effect": "Allow",  
            "Action": "rds:DescribeDBInstances",  
            "Resource": "*"  
        }  
    ]  
}
```

6. Choose **Review policy**.
7. Provide a name for the policy and optionally a description, and then choose **Create policy**.

You can now attach the policy to an IAM user or role. The following procedure assumes that you already have an IAM user available for this purpose.

To attach the policy to an IAM user

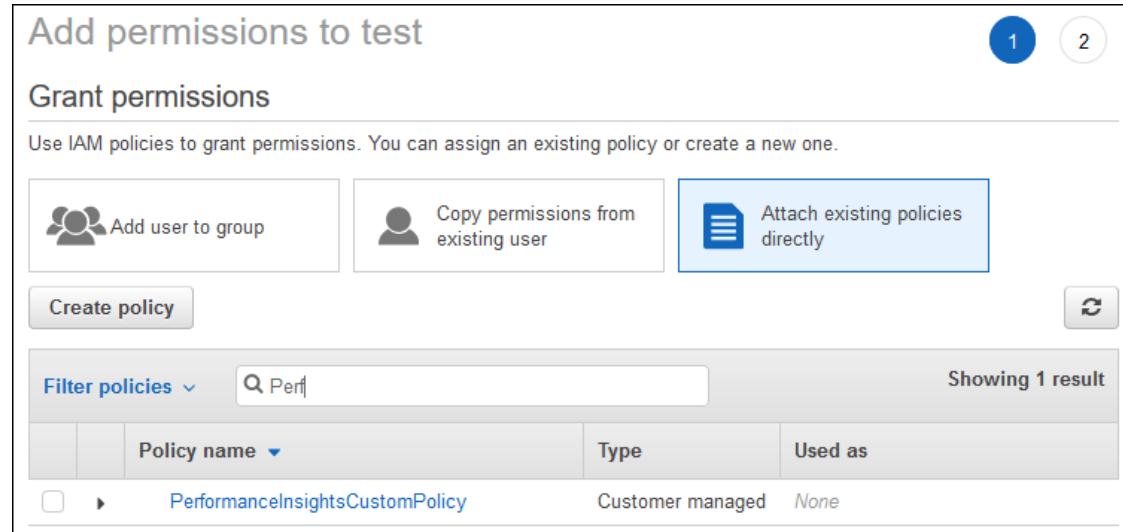
1. Open the IAM console at <https://console.aws.amazon.com/iam/>.

2. In the navigation pane, choose **Users**.
3. Choose an existing user from the list.

Important

To use Performance Insights, make sure that you have access to Amazon RDS in addition to the custom policy. For example, the `AmazonRDSReadOnlyAccess` predefined policy provides read-only access to Amazon RDS. For more information, see [Managing access using policies \(p. 1726\)](#).

4. On the **Summary** page, choose **Add permissions**.
5. Choose **Attach existing policies directly**. For **Search**, type the first few characters of your policy name, as shown following.



6. Choose your policy, and then choose **Next: Review**.
7. Choose **Add permissions**.

Configuring an AWS KMS policy for Performance Insights

Performance Insights uses an AWS KMS key to encrypt sensitive data. When you enable Performance Insights through the API or the console, you have the following options:

- Choose the default AWS managed key.

Amazon RDS uses the AWS managed key for your new DB instance. Amazon RDS creates an AWS managed key for your AWS account. Your AWS account has a different AWS managed key for Amazon RDS for each AWS Region.

- Choose a customer managed key.

If you specify a customer managed key, users in your account that call the Performance Insights API need the `kms:Decrypt` and `kms:GenerateDataKey` permissions on the KMS key. You can configure these permissions through IAM policies. However, we recommend that you manage these permissions through your KMS key policy. For more information, see [Using key policies in AWS KMS](#).

Example

The following sample key policy shows how to add statements to your KMS key policy. These statements allow access to Performance Insights. Depending on how you use the KMS key, you might want to change some restrictions. Before adding statements to your policy, remove all comments.

```
{  
    "Version" : "2012-10-17",  
    "Id" : "your-policy",  
    "Statement" : [ {  
        //This represents a statement that currently exists in your policy.  
    }  
    ....,  
    //Starting here, add new statement to your policy for Performance Insights.  
    //We recommend that you add one new statement for every RDS instance  
    {  
        "Sid" : "Allow viewing RDS Performance Insights",  
        "Effect": "Allow",  
        "Principal": {  
            "AWS": [  
                //One or more principals allowed to access Performance Insights  
                "arn:aws:iam::444455556666:role/Role1"  
            ]  
        },  
        "Action": [  
            "kms:Decrypt",  
            "kms:GenerateDataKey"  
        ],  
        "Resource": "*",  
        "Condition" : {  
            "StringEquals" : {  
                //Restrict access to only RDS APIs (including Performance Insights).  
                //Replace region with your AWS Region.  
                //For example, specify us-west-2.  
                "kms:ViaService" : "rds.region.amazonaws.com"  
            },  
            "ForAnyValue:StringEquals": {  
                //Restrict access to only data encrypted by Performance Insights.  
                "kms:EncryptionContext:aws:pi:service": "rds",  
                "kms:EncryptionContext:service": "pi",  
  
                //Restrict access to a specific RDS instance.  
                //The value is a DbiResourceId.  
                "kms:EncryptionContext:aws:rds:db-id": "db-AAAAABBBBCCCCDDDDDEEEEE"  
            }  
        }  
    }  
}
```

Analyzing metrics with the Performance Insights dashboard

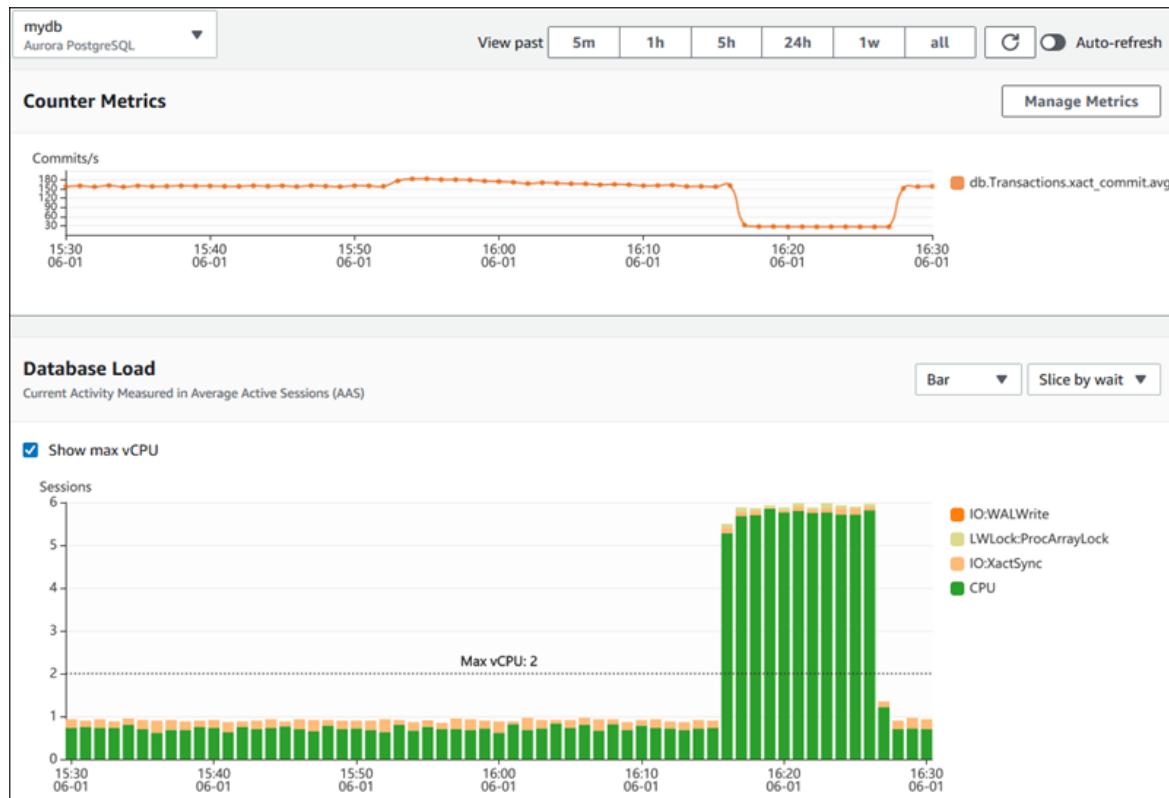
The Performance Insights dashboard contains database performance information to help you analyze and troubleshoot performance issues. On the main dashboard page, you can view information about the database load. You can "slice" DB load by dimensions such as wait events or SQL.

Performance Insights dashboard

- [Overview of the Performance Insights dashboard \(p. 586\)](#)
- [Opening the Performance Insights dashboard \(p. 591\)](#)
- [Analyzing DB load by wait events \(p. 592\)](#)
- [Analyzing running queries using the Performance Insights dashboard \(p. 593\)](#)
- [Accessing the text of SQL statements \(p. 604\)](#)
- [Zooming In on the DB Load chart \(p. 606\)](#)

Overview of the Performance Insights dashboard

The dashboard is the easiest way to interact with Performance Insights. The following example shows the dashboard for a MySQL DB instance. By default, the Performance Insights dashboard shows data for the last hour.



The dashboard is divided into the following parts:

1. **Counter Metrics** – Shows data for specific performance counter metrics.
2. **DB Load Chart** – Shows how the DB load compares to DB instance capacity as represented by the **Max vCPU** line.
3. **Top items** – Shows the top dimensions contributing to DB load.

Topics

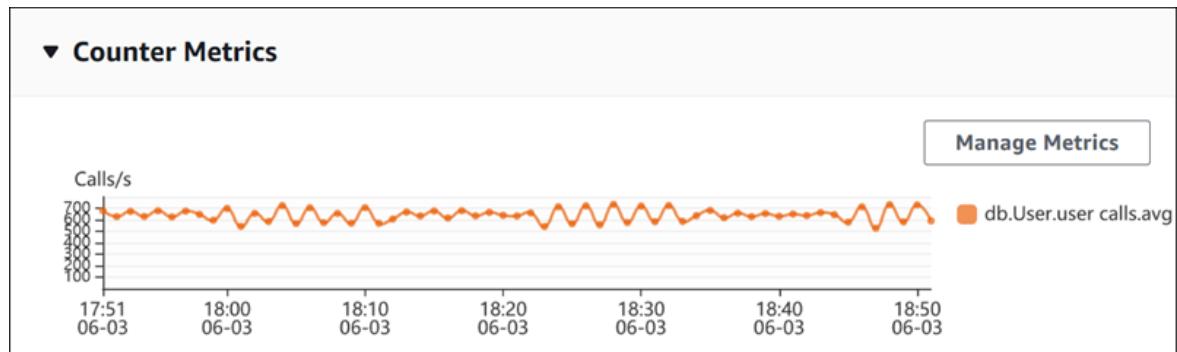
- [Counter metrics chart \(p. 586\)](#)
- [Database load chart \(p. 588\)](#)
- [Top dimensions table \(p. 590\)](#)

Counter metrics chart

With counter metrics, you can customize the Performance Insights dashboard to include up to 10 additional graphs. These graphs show a selection of dozens of operating system and database performance metrics. You can correlate this information with DB load to help identify and analyze performance problems.

The **Counter metrics** chart displays data for performance counters. The default metrics depend on the DB engine:

- Aurora MySQL – db.SQL.Innodb_rows_read.avg
- Aurora PostgreSQL – db.Transactions.xact_commit.avg



To change the performance counters, choose **Manage Metrics**. You can select multiple **OS metrics** or **Database metrics**, as shown in the following screenshot. To see details for any metric, hover over the metric name.

Select metrics shown on the graph

Check the metrics that you want to see on the Performance Insights dashboard.

Find metrics

OS metrics (0) **Database metrics (1)** **Clear all selections**

User

- CPU used by this session
- SQL*Net roundtrips to/from client
- bytes received via SQL*Net from client
- user commits
- logons cumulative
- user calls
- bytes sent via SQL*Net to client
- user rollbacks

Redo

- redo size

Cache

- physical read bytes
- db block gets
- DBWR checkpoints
- physical reads
- consistent gets from cache
- db block gets from cache
- consistent gets

SQL

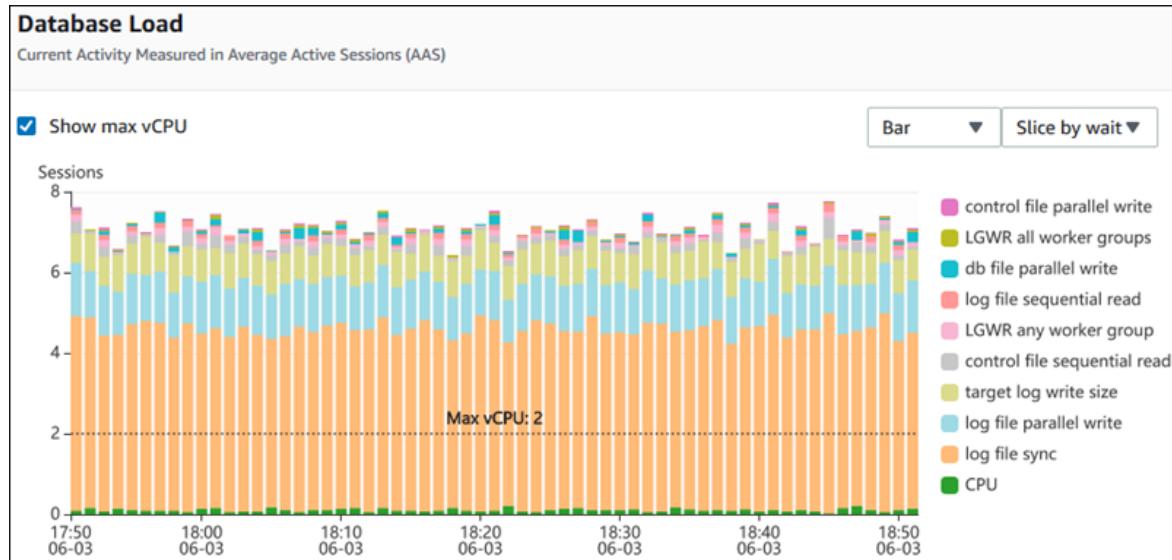
- parse count (total)
- parse count (hard)
- table scan rows gotten
- sorts (memory)
- sorts (disk)
- sorts (rows)

Cancel **Update graph**

For descriptions of the counter metrics that you can add for each DB engine, see [Performance Insights counter metrics \(p. 653\)](#).

Database load chart

The **Database load** chart shows how the database activity compares to DB instance capacity as represented by the **Max vCPU** line. By default, the stacked line chart represents DB load as average active sessions per unit of time. The DB load is sliced (grouped) by wait states.

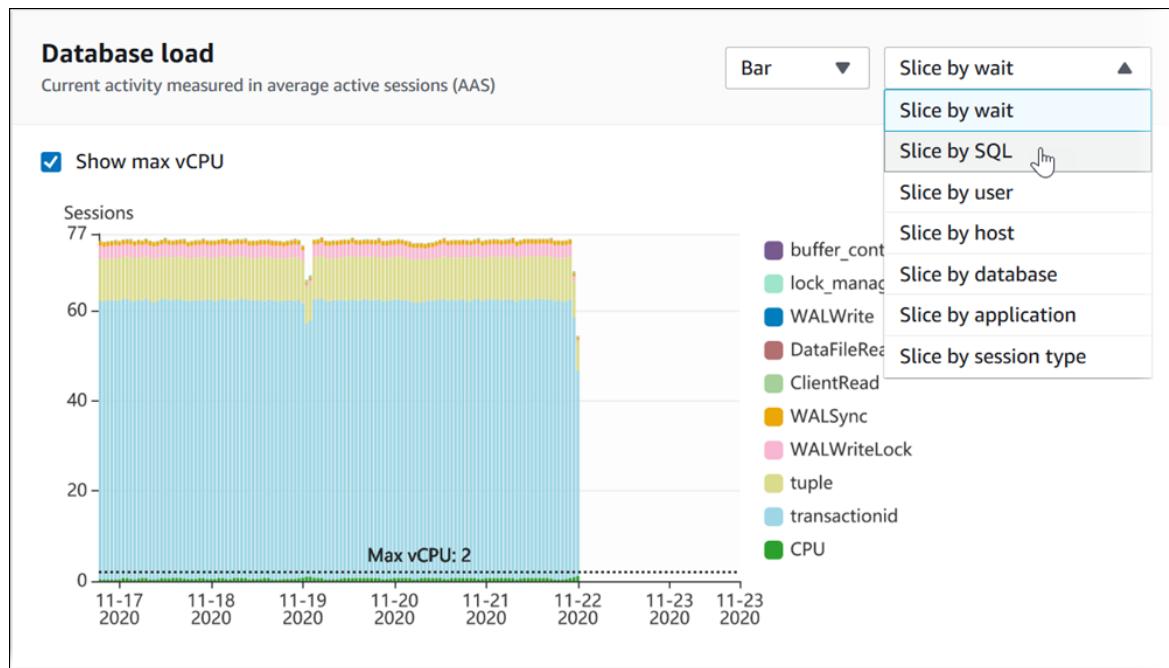


DB load sliced by dimensions

You can choose to display load as active sessions grouped by any supported dimensions. The following table shows which dimensions are supported for the different engines.

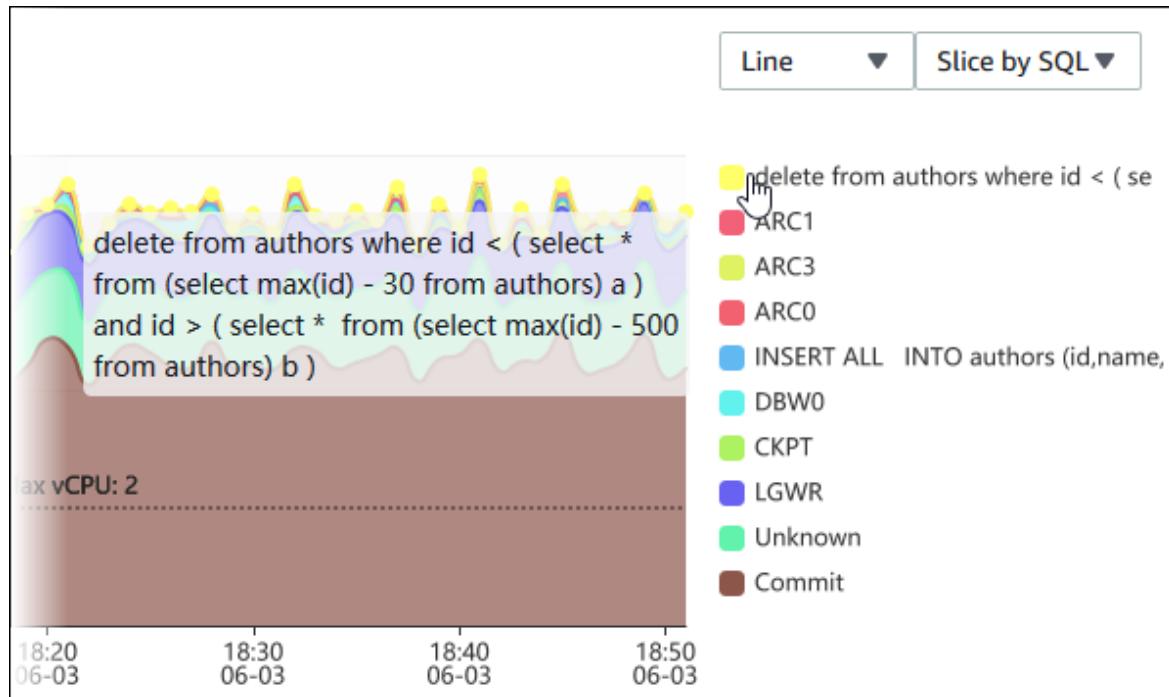
Dimension	Aurora PostgreSQL	Aurora MySQL
Host	Yes	Yes
SQL	Yes	Yes
User	Yes	Yes
Waits	Yes	Yes
Application	Yes	No
Database	Yes	Yes
Session type	Yes	No

The following image shows the dimensions for a PostgreSQL DB instance.

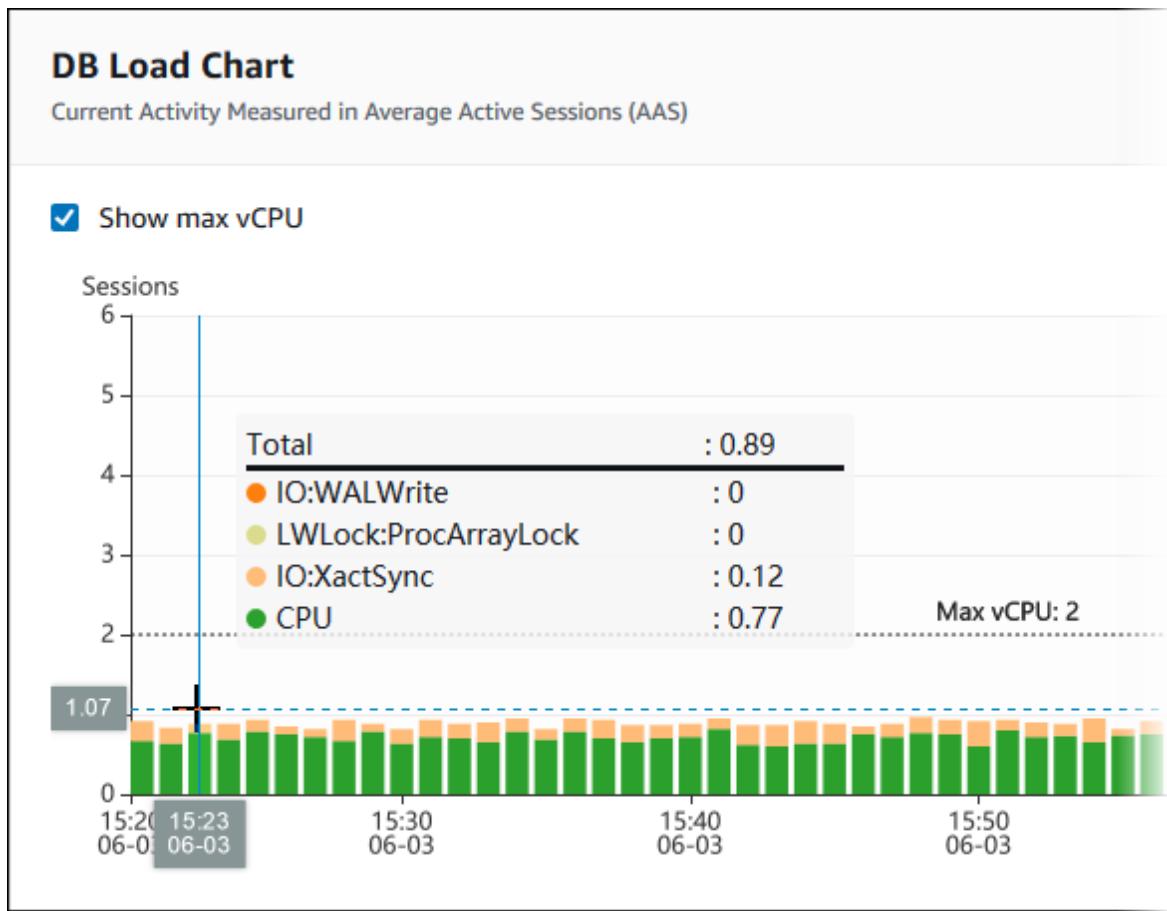


DB load details for a dimension item

To see details about a DB load item within a dimension, hover over the item name. The following image shows details for a SQL statement.



To see details for any item for the selected time period in the legend, hover over that item.



Top dimensions table

The Top dimensions table slices DB load by different dimensions. A dimension is a category or "slice by" for different characteristics of DB load. If the dimension is SQL, **Top SQL** shows the SQL statements that contribute the most to DB load.

Top waits	Top SQL	Top hosts	Top users	Top connections	Top databases	Top applications	Top session types
Top SQL (0) Learn more							
<input type="text"/> Find SQL statements							
Load by waits (AAS)				SQL statements			

Choose any of the following dimension tabs.

Tab	Description	Supported engines
Top SQL	The SQL statements that are currently running	All
Top waits	The event for which the database backend is waiting	All

Tab	Description	Supported engines
Top hosts	The host name of the connected client	All
Top users	The user logged in to the database	All
The name of the database to which the client is connected		
Top applications	The name of the application that is connected to the database	Aurora PostgreSQL only
Top session types	The type of the current session	Aurora PostgreSQL only

To learn how to analyze queries by using the **Top SQL** tab, see [Overview of the Top SQL tab \(p. 593\)](#).

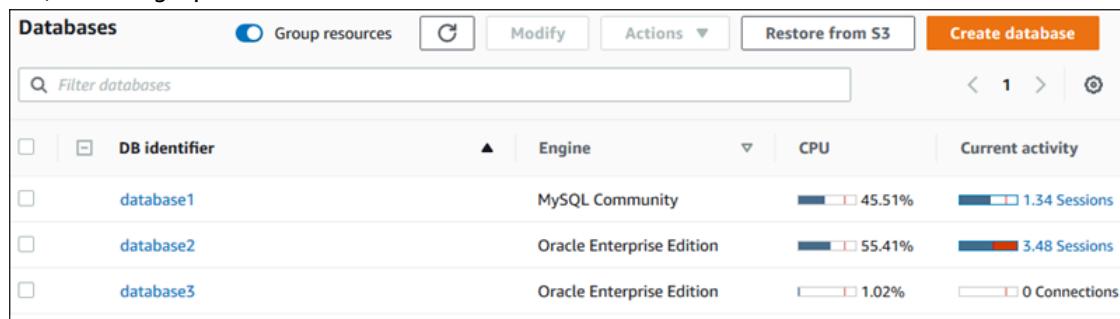
Opening the Performance Insights dashboard

To see the Performance Insights dashboard, use the following procedure.

To view the Performance Insights dashboard in the AWS Management Console

1. Open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. In the navigation pane, choose **Performance Insights**.
3. Choose a DB instance. The Performance Insights dashboard is shown for that DB instance.

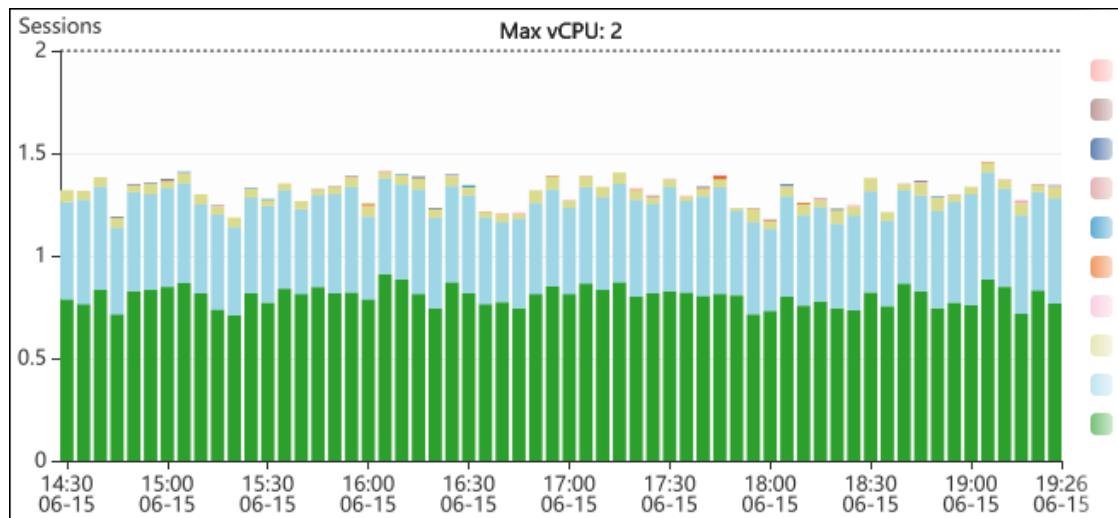
For DB instances with Performance Insights enabled, you can also reach the dashboard by choosing the **Sessions** item in the list of DB instances. Under **Current activity**, the **Sessions** item shows the database load in average active sessions over the last five minutes. The bar graphically shows the load. When the bar is empty, the DB instance is idle. As the load increases, the bar fills with blue. When the load passes the number of virtual CPUs (vCPUs) on the DB instance class, the bar turns red, indicating a potential bottleneck.



4. (Optional) Choose a different time interval by selecting a button in the upper right. For example, to change the interval to 5 hours, select **5h**.



In the following screenshot, the DB load interval is 5 hours.



5. (Optional) To refresh your data automatically, enable **Auto refresh**.



The Performance Insight dashboard automatically refreshes with new data. The refresh rate depends on the amount of data displayed:

- 5 minutes refreshes every 5 seconds.
- 1 hour refreshes every minute.
- 5 hours refreshes every minute.
- 24 hours refreshes every 5 minutes.
- 1 week refreshes every hour.

Analyzing DB load by wait events

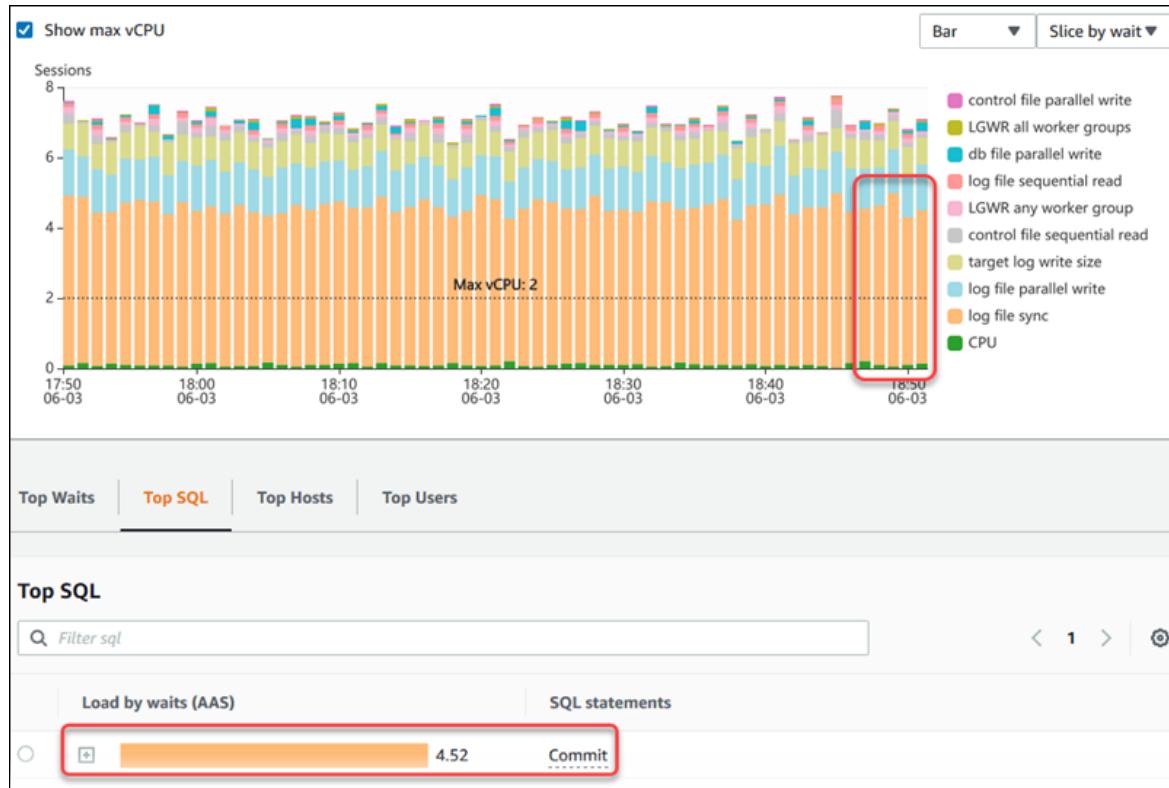
If the **Database load** chart shows a bottleneck, you can find out where the load is coming from. To do so, look at the top load items table below the **Database load** chart. Choose a particular item, like a SQL query or a user, to drill down into that item and see details about it.

DB load grouped by waits and top SQL queries is the default Performance Insights dashboard view. This combination typically provides the most insight into performance issues. DB load grouped by waits shows if there are any resource or concurrency bottlenecks in the database. In this case, the **SQL** tab of the top load items table shows which queries are driving that load.

Your typical workflow for diagnosing performance issues is as follows:

1. Review the **Database load** chart and see if there are any incidents of database load exceeding the **Max CPU** line.
2. If there is, look at the **Database load** chart and identify which wait state or states are primarily responsible.
3. Identify the digest queries causing the load by seeing which of the queries the **SQL** tab on the top load items table are contributing most to those wait states. You can identify these by the **DB Load by Wait** column.
4. Choose one of these digest queries in the **SQL** tab to expand it and see the child queries that it is composed of.

For example, in the dashboard following, **log file sync** waits account for most of the DB load. The **LGWR all worker groups** wait is also high. The **Top SQL** chart shows what is causing the **log file sync** waits: frequent COMMIT statements. In this case, committing less frequently will reduce DB load.



Analyzing running queries using the Performance Insights dashboard

In the Amazon RDS Performance Insights dashboard, you can find information about running queries in the **Top SQL** tab in the **Top dimensions** table. You can use this information to tune your queries.

Note

RDS for SQL Server doesn't show SQL-level statistics.

Topics

- [Overview of the Top SQL tab \(p. 593\)](#)
- [Analyzing running queries in Aurora MySQL \(p. 598\)](#)
- [Analyzing running queries in Aurora PostgreSQL \(p. 601\)](#)

Overview of the Top SQL tab

By default, the **Top SQL** tab shows the SQL queries that are contributing the most to DB load. To help tune your queries, you can analyze information such as the query text, statistics, and Support SQL ID. You can also choose the statistics that you want to appear in the **Top SQL** tab.

Topics

- [SQL statistics \(p. 594\)](#)
- [Load by waits \(AAS\) \(p. 594\)](#)
- [SQL information \(p. 595\)](#)

- [Preferences \(p. 596\)](#)

SQL statistics

SQL statistics are performance-related metrics about SQL queries. For example, Performance Insights might show executions per second or rows processed per second. Performance Insights collects statistics for only the most common queries. Typically, these match the top queries by load shown in the Performance Insights dashboard.

A *SQL digest* is a composite of multiple actual queries that are structurally similar but might have different literal values. The digest replaces hardcoded values with a question mark. For example, a digest might be `SELECT * FROM emp WHERE lname= ?`. This digest might include the following child queries:

```
SELECT * FROM emp WHERE lname = 'Miller'
SELECT * FROM emp WHERE lname = 'Olagappan'
SELECT * FROM emp WHERE lname = 'Wu'
```

Every line in the **Top SQL** table shows relevant statistics for the SQL statement or digest, as shown in the following example.

Top SQL		SQL statements	calls/sec	rows/sec
		select minute_rollups(?)	0.06	0.06
		select count(*) from authors where id < (select max(id) - 31 from authors) and...	33.68	101.04
		WITH cte AS (SELECT id FROM authors LIMIT ?) UPDATE ...	33.68	33.68
		delete from authors where id < (select * from (select max(id) - ? from authors...)	33.68	303.13
		INSERT INTO authors (id,name,email) VALUES (nextval(?),?,(nextval(?),?))	33.68	303.13
		select count(*) from authors where id < (select max(id) - 31 from authors) and...	0.00	0.00

To see the literal SQL statements in a digest, select the query, and then choose the plus symbol (+). In the following screenshot, the selected query is a digest.

Load by waits (AAS)	SQL statements
0.88	select minute_rollups(?)
0.50	select minute_rollups(1000000)
0.53	select count(*) from authors where id < (select max(id) - 31 from authors) and...

Note

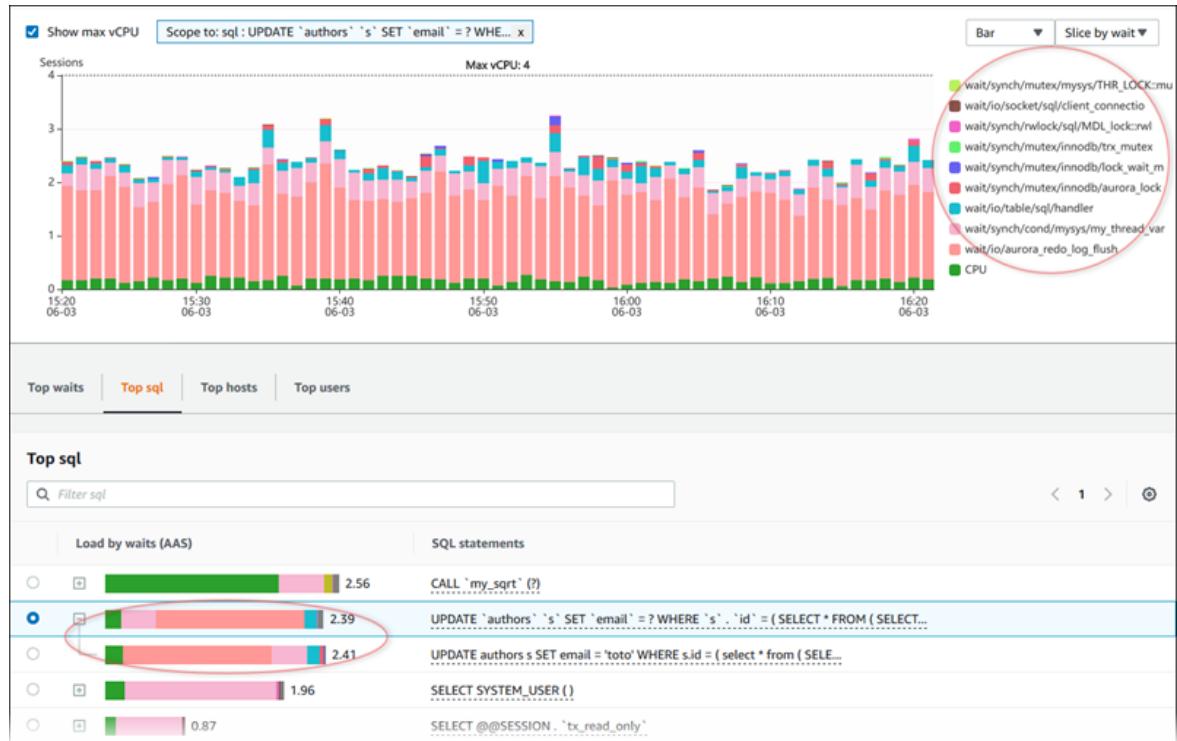
A SQL digest groups similar SQL statements, but does not redact sensitive information.

Load by waits (AAS)

In **Top SQL**, the **Load by waits (AAS)** column illustrates the percentage of the database load associated with each top load item. This column reflects the load for that item by whatever grouping is currently selected in the **DB Load Chart**.

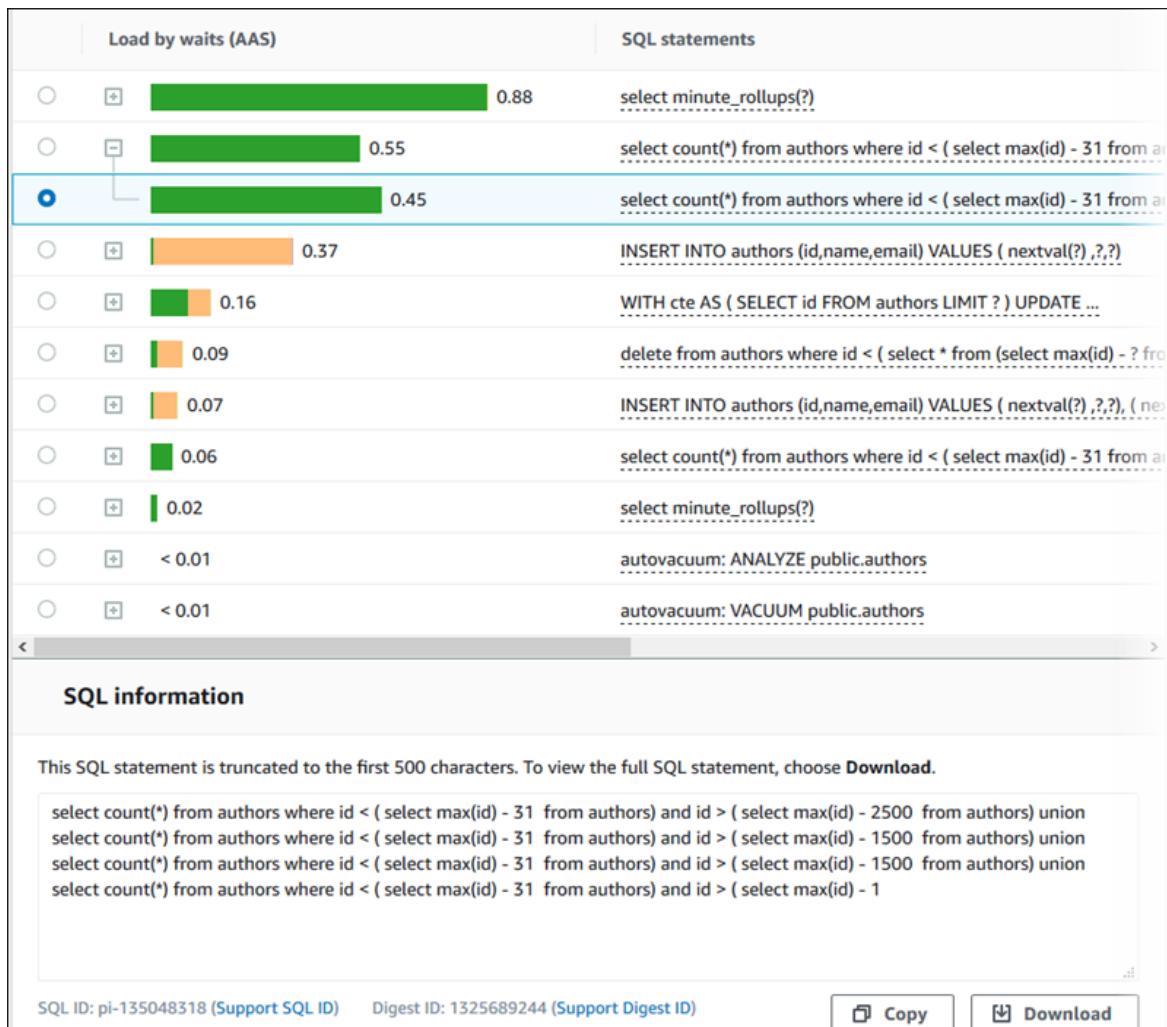
For example, you might group the **DB load** chart by wait states. You examine SQL queries in the top load items table. In this case, the **DB Load by Waits** bar is sized, segmented, and color-coded to show how

much of a given wait state that query is contributing to. It also shows which wait states are affecting the selected query.



SQL information

In the **Top SQL** table, you can open a statement to view its information. The information appears in the bottom pane.

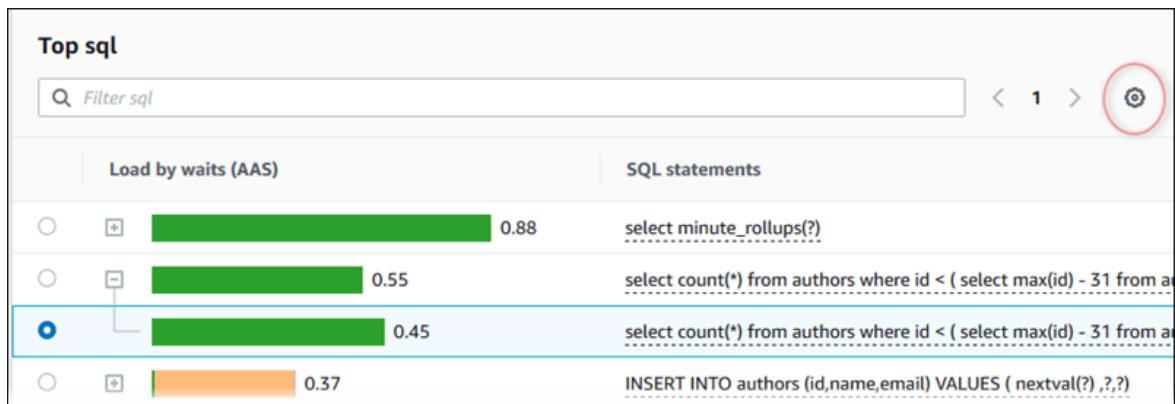


The following types of identifiers (IDs) that are associated with SQL statements:

- **Support SQL ID** – A hash value of the SQL ID. This value is only for referencing a SQL ID when you are working with AWS Support. AWS Support doesn't have access to your actual SQL IDs and SQL text.
- **Support Digest ID** – A hash value of the digest ID. This value is only for referencing a digest ID when you are working with AWS Support. AWS Support doesn't have access to your actual digest IDs and SQL text.

Preferences

You can control the statistics displayed in the **Top SQL** tab by choosing the **Preferences** icon.



When you choose the **Preferences** icon, the **Preferences** window opens.

The Preferences window allows you to select which metrics are displayed in the Top SQL tab. Most checkboxes are currently selected (indicated by a blue circle).

Metric	Status
Load by waits (AAS)	Selected
SQL statements	Selected
calls/sec (calls_per_sec)	Selected
rows/sec (rows_per_sec)	Selected
AAE (total_time_per_sec)	Selected
blk hits/sec (shared_blk_hits_per_sec)	Selected
blk reads/sec (shared_blk_reads_per_sec)	Selected
blk dirty/sec (shared_blk_dirty_per_sec)	Selected
blk writes/sec (shared_blk_written_per_sec)	Selected
local blk hits/sec (local_blk_hits_per_sec)	Selected
local blk reads/sec (local_blk_reads_per_sec)	Selected
local blk dirty/sec (local_blk_dirty_per_sec)	Selected

To enable the statistics that you want to appear in the **Top SQL** tab, use your mouse to scroll to the bottom of the window, and then choose **Continue**.

Analyzing running queries in Aurora MySQL

Aurora MySQL collect SQL statistics only at the digest level. No statistics are shown at the statement level.

Topics

- [Digest table in Aurora MySQL \(p. 598\)](#)
- [Per-second statistics for Aurora MySQL \(p. 598\)](#)
- [Per-call statistics for Aurora MySQL \(p. 599\)](#)
- [Viewing SQL statistics for Aurora MySQL \(p. 599\)](#)

Digest table in Aurora MySQL

Performance Insights collects SQL digest statistics from the `events_statements_summary_by_digest` table. The `events_statements_summary_by_digest` table is managed by your database.

The digest table doesn't have an eviction policy. When the table is full, the AWS Management Console shows the following message:

Performance Insights is unable to collect SQL Digest statistics on new queries because the table `events_statements_summary_by_digest` is full.
Please truncate `events_statements_summary_by_digest` table to clear the issue. Check the User Guide for more details.

In this situation, Aurora MySQL doesn't track SQL queries. To address this issue, Performance Insights automatically truncates the digest table when both of the following conditions are met:

- The table is full.
- Performance Insights manages the Performance Schema automatically. For automatic management, the `performance_schema` parameter must be set to 0 and the **Source** must not be set to `user`.

If Performance Insights isn't managing the Performance Schema automatically, see [Enabling the Performance Schema for Performance Insights on Aurora MySQL \(p. 580\)](#).

In the AWS CLI, check the source of a parameter value by running the `describe-db-parameters` command.

Per-second statistics for Aurora MySQL

The following SQL statistics are available for Aurora MySQL DB clusters.

Metric	Unit
<code>db.sql_tokenized.stats.count_star_per_sec</code>	Calls per second
<code>db.sql_tokenized.stats.sum_timer_wait_per_sec</code>	Average active executions per second (AAE)
<code>db.sql_tokenized.stats.sum_select_full_join_per_sec</code>	Select full join per second
<code>db.sql_tokenized.stats.sum_select_range_check_per_sec</code>	Select range check per second
<code>db.sql_tokenized.stats.sum_select_scan_per_sec</code>	Select scan per second
<code>db.sql_tokenized.stats.sum_sort_merge_passes_per_sec</code>	Sort merge passes per second
<code>db.sql_tokenized.stats.sum_sort_scan_per_sec</code>	Sort scans per second

Metric	Unit
db.sql_tokenized.stats.sum_sort_range_per_sec	Sort ranges per second
db.sql_tokenized.stats.sum_sort_rows_per_sec	Sort rows per second
db.sql_tokenized.stats.sum_rows_affected_per_sec	Rows affected per second
db.sql_tokenized.stats.sum_rows_examined_per_sec	Rows examined per second
db.sql_tokenized.stats.sum_rows_sent_per_sec	Rows sent per second
db.sql_tokenized.stats.sum_created_tmp_disk_tables	Created temporary disk tables per second
db.sql_tokenized.stats.sum_created_tmp_tables_per_sec	Created temporary tables per second
db.sql_tokenized.stats.sum_lock_time_per_sec	Lock time per second (in ms)

Per-call statistics for Aurora MySQL

The following metrics provide per call statistics for a SQL statement.

Metric	Unit
db.sql_tokenized.stats.sum_timer_wait_per_call	Average latency per call (in ms)
db.sql_tokenized.stats.sum_select_full_join_per_call	Select full joins per call
db.sql_tokenized.stats.sum_select_range_check_per_call	Select range check per call
db.sql_tokenized.stats.sum_select_scan_per_call	Select scans per call
db.sql_tokenized.stats.sum_sort_merge_passes_per_call	Sort merge passes per call
db.sql_tokenized.stats.sum_sort_scan_per_call	Sort scans per call
db.sql_tokenized.stats.sum_sort_range_per_call	Sort ranges per call
db.sql_tokenized.stats.sum_sort_rows_per_call	Sort rows per call
db.sql_tokenized.stats.sum_rows_affected_per_call	Rows affected per call
db.sql_tokenized.stats.sum_rows_examined_per_call	Rows examined per call
db.sql_tokenized.stats.sum_rows_sent_per_call	Rows sent per call
db.sql_tokenized.stats.sum_created_tmp_disk_tables	Created temporary disk tables per call
db.sql_tokenized.stats.sum_created_tmp_tables_per_call	Created temporary tables per call
db.sql_tokenized.stats.sum_lock_time_per_call	Lock time per call (in ms)

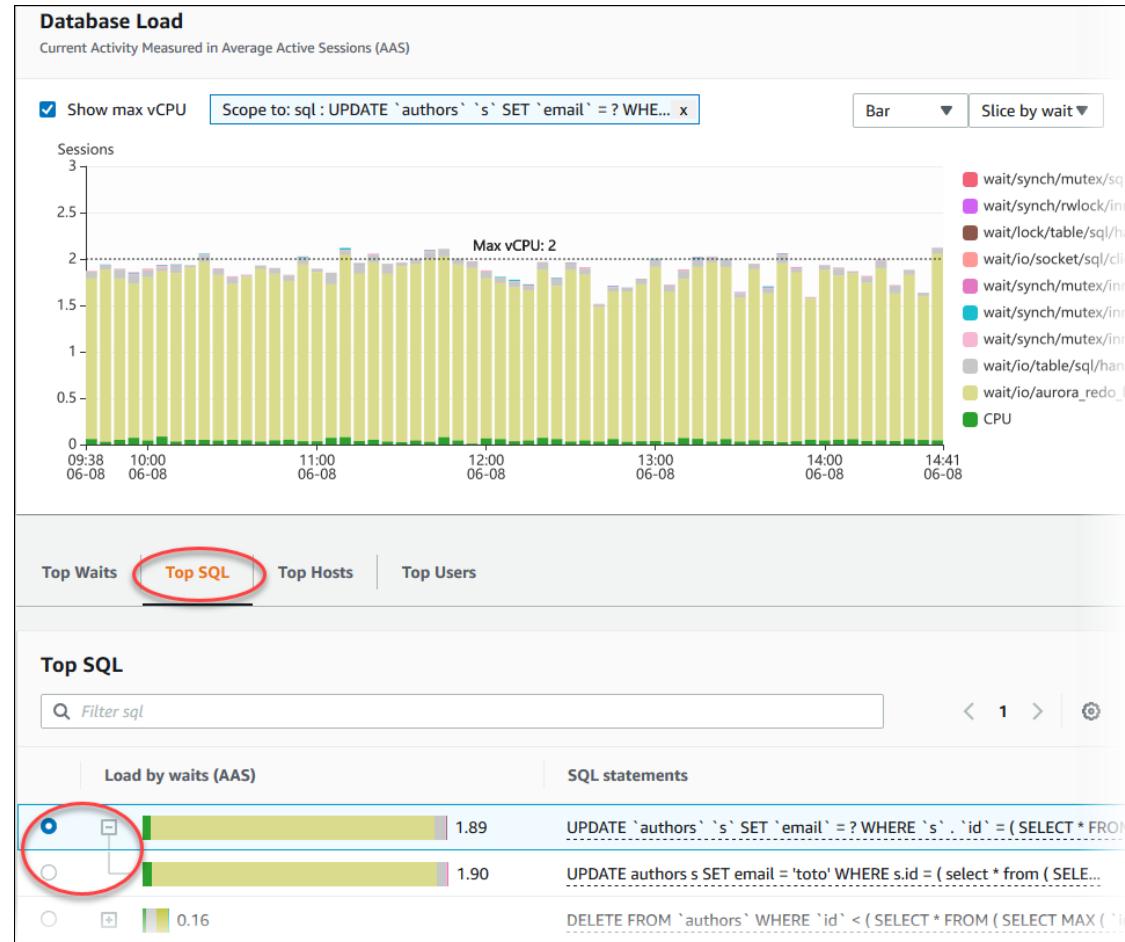
Viewing SQL statistics for Aurora MySQL

The statistics are available in the **Top SQL** tab of the **Database load** chart.

To view the SQL statistics

1. Open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.

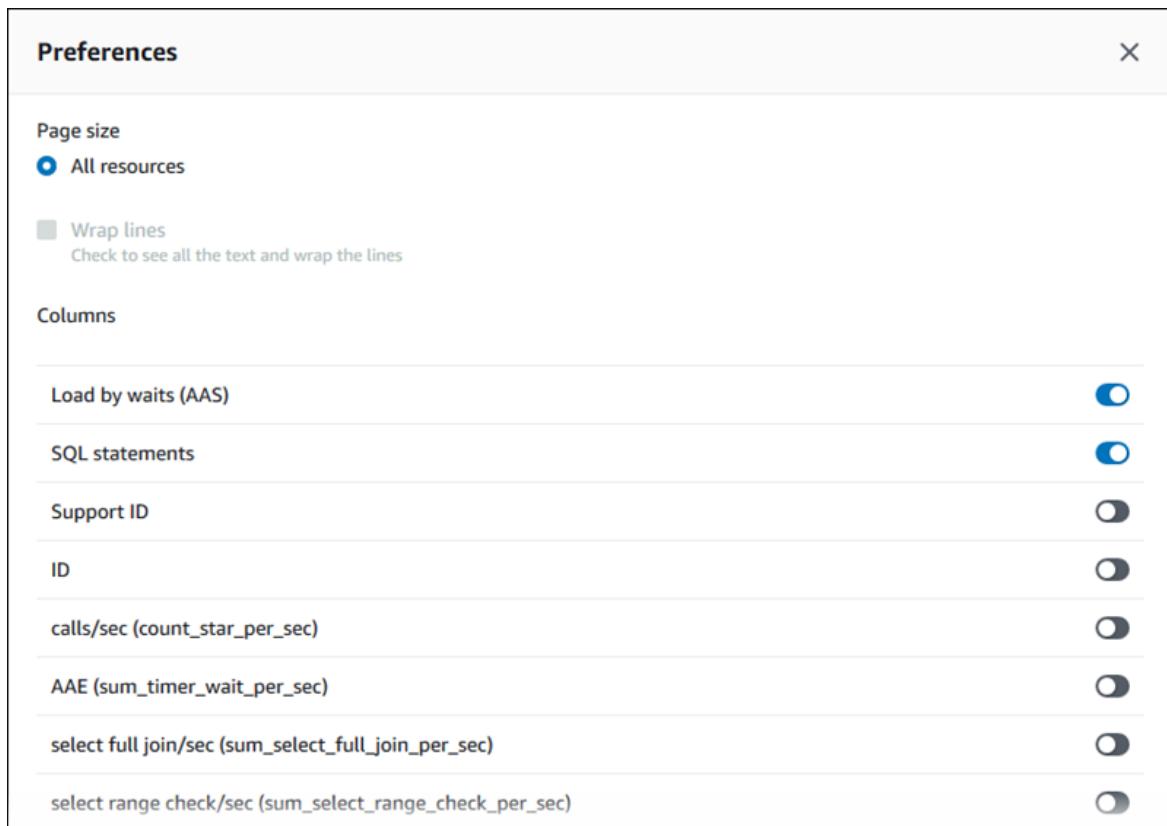
2. Go to the Performance Insights dashboard.
3. Choose the **SQL** tab and expand a query.



4.

Choose which statistics to display by choosing the gear icon in the upper-right corner of the chart.

The following screenshot shows the preferences for Aurora MySQL DB instances.



Analyzing running queries in Aurora PostgreSQL

Aurora PostgreSQL collects SQL statistics only at the digest level. No statistics are shown at the statement level.

Topics

- [Digest statistics for Aurora PostgreSQL \(p. 601\)](#)
- [Per-second digest statistics for Aurora PostgreSQL \(p. 602\)](#)
- [Per-call digest statistics for Aurora PostgreSQL \(p. 602\)](#)
- [Viewing SQL statistics for Aurora PostgreSQL \(p. 603\)](#)

Digest statistics for Aurora PostgreSQL

To view SQL digest statistics, the `pg_stat_statements` library must be loaded. For Aurora PostgreSQL DB clusters that are compatible with PostgreSQL 10, this library is loaded by default. For Aurora PostgreSQL DB clusters that are compatible with PostgreSQL 9.6, you enable this library manually. To enable it manually, add `pg_stat_statements` to `shared_preload_libraries` in the DB parameter group associated with the DB instance. Then reboot your DB instance. For more information, see [Working with DB parameter groups and DB cluster parameter groups \(p. 339\)](#).

Note

Performance Insights can only collect statistics for queries in `pg_stat_activity` that aren't truncated. By default, PostgreSQL databases truncate queries longer than 1,024 bytes. To increase the query size, change the `track_activity_query_size` parameter in the DB parameter group associated with your DB instance. When you change this parameter, a DB instance reboot is required.

Per-second digest statistics for Aurora PostgreSQL

The following SQL digest statistics are available for Aurora PostgreSQL DB instances.

Metric	Unit
db.sql_tokenized.stats.calls_per_sec	Calls per second
db.sql_tokenized.stats.rows_per_sec	Rows per second
db.sql_tokenized.stats.total_time_per_sec	Average active executions per second (AAE)
db.sql_tokenized.stats.shared_blk_hit_per_sec	Block hits per second
db.sql_tokenized.stats.shared_blk_read_per_sec	Block reads per second
db.sql_tokenized.stats.shared_blk_dirtied_per_sec	Blocks dirtied per second
db.sql_tokenized.stats.shared_blk_written_per_sec	Block writes per second
db.sql_tokenized.stats.local_blk_hit_per_sec	Local block hits per second
db.sql_tokenized.stats.local_blk_read_per_sec	Local block reads per second
db.sql_tokenized.stats.local_blk_dirtied_per_sec	Local block dirty per second
db.sql_tokenized.stats.local_blk_written_per_sec	Local block writes per second
db.sql_tokenized.stats.temp_blk_written_per_sec	Temporary writes per second
db.sql_tokenized.stats.temp_blk_read_per_sec	Temporary reads per second
db.sql_tokenized.stats.blk_read_time_per_sec	Average concurrent reads per second
db.sql_tokenized.stats.blk_write_time_per_sec	Average concurrent writes per second

Per-call digest statistics for Aurora PostgreSQL

The following metrics provide per call statistics for a SQL statement.

Metric	Unit
db.sql_tokenized.stats.rows_per_call	Rows per call
db.sql_tokenized.stats.avg_latency_per_call	Average latency per call (in ms)
db.sql_tokenized.stats.shared_blk_hit_per_call	Block hits per call
db.sql_tokenized.stats.shared_blk_read_per_call	Block reads per call
db.sql_tokenized.stats.shared_blk_written_per_call	Block writes per call
db.sql_tokenized.stats.shared_blk_dirtied_per_call	Blocks dirtied per call
db.sql_tokenized.stats.local_blk_hit_per_call	Local block hits per call
db.sql_tokenized.stats.local_blk_read_per_call	Local block reads per call
db.sql_tokenized.stats.local_blk_dirtied_per_call	Local block dirty per call
db.sql_tokenized.stats.local_blk_written_per_call	Local block writes per call

Metric	Unit
db.sql_tokenized.stats.temp_blk_written_per_call	Temporary block writes per call
db.sql_tokenized.stats.temp_blk_read_per_call	Temporary block reads per call
db.sql_tokenized.stats.blk_read_time_per_call	Read time per call (in ms)
db.sql_tokenized.stats.blk_write_time_per_call	Write time per call (in ms)

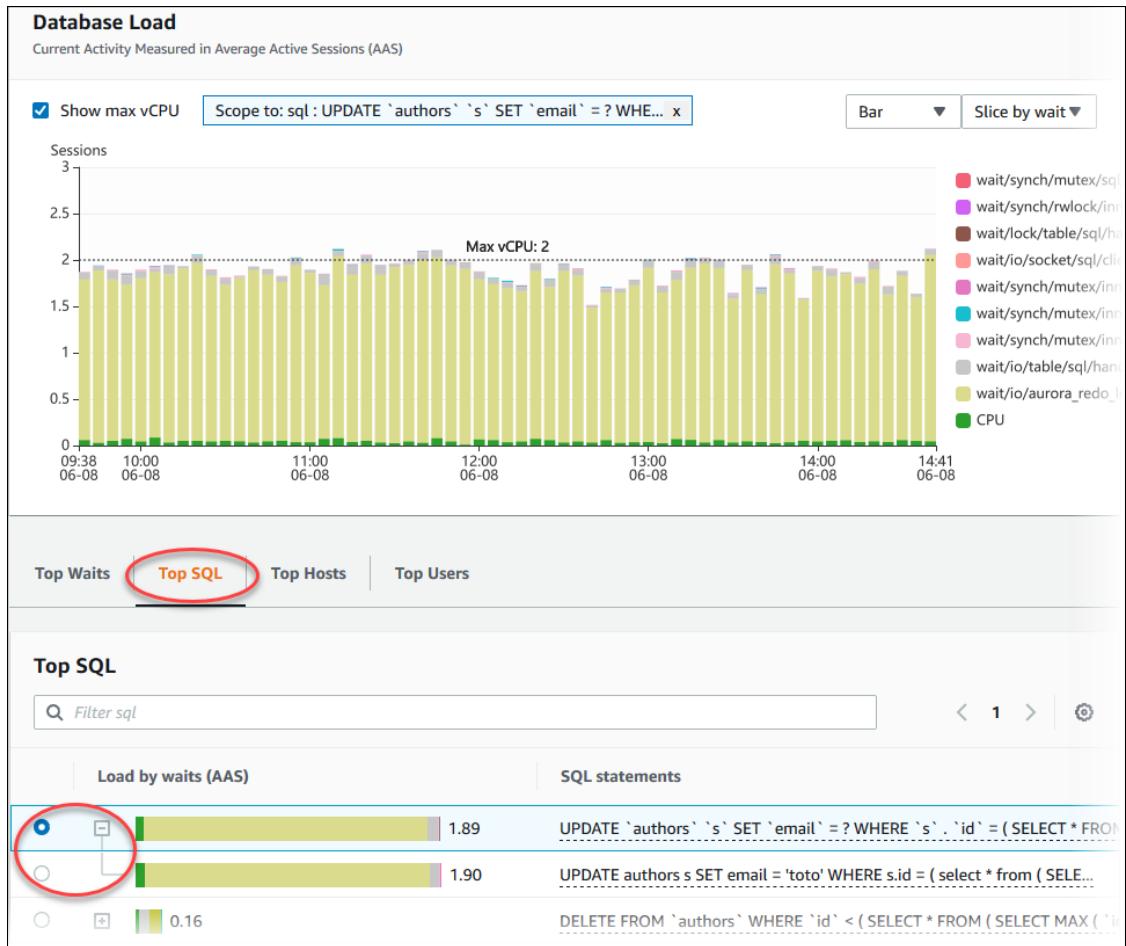
For more information about these metrics, see [pg_stat_statements](#) in the PostgreSQL documentation.

Viewing SQL statistics for Aurora PostgreSQL

The statistics are available in the **Top SQL** tab of the **Database load** chart.

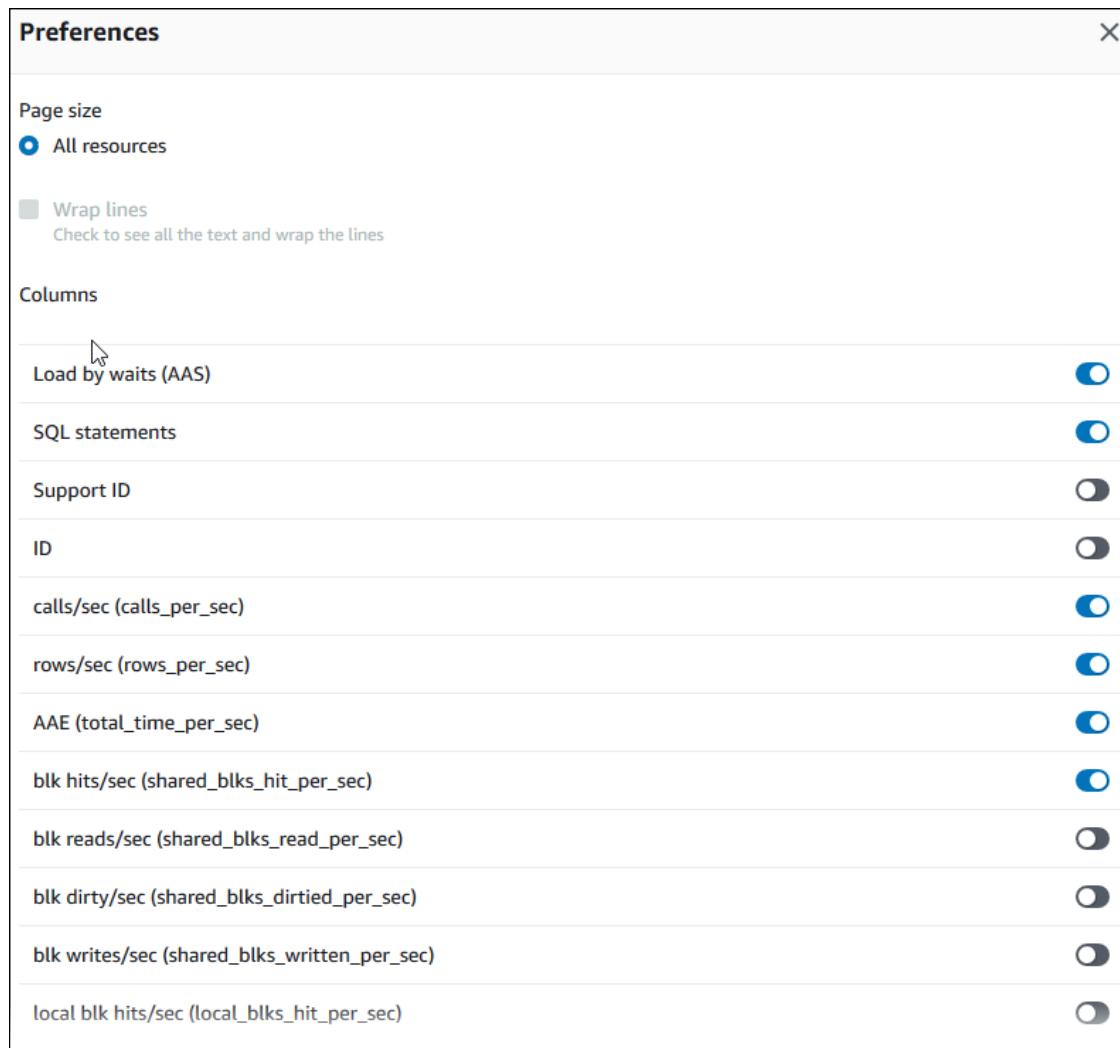
To view the SQL statistics

1. Open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. Go to the Performance Insights dashboard.
3. Choose the **SQL** tab.



4. Choose which statistics to display by choosing the gear icon in the upper-right corner of the chart.

The following screenshot shows the preferences for Aurora PostgreSQL.



Accessing the text of SQL statements

By default, each row in the **Top SQL** table shows 500 bytes of SQL text for each SQL statement. When a SQL statement exceeds 500 bytes, you can view more text by opening the statement in the Performance Insights dashboard. In this case, the maximum length for the displayed query is 4 KB. This limit is introduced by the console and is subject to the limits set by the database engine. If you view a child SQL statement, you can also choose **Download**.

Topics

- [Text size limits for Aurora MySQL \(p. 604\)](#)
- [Setting the SQL text limit for Aurora PostgreSQL DB instances \(p. 605\)](#)
- [Viewing and downloading SQL text in the Performance Insights dashboard \(p. 605\)](#)

Text size limits for Aurora MySQL

When you download a SQL statement, the database engine determines the maximum length of the text. You can download text up to the following per-engine limits:

- Aurora MySQL 5.7 – 4,096 bytes
- Aurora MySQL 5.6 – 1,024 bytes

The Performance Insights console displays up to the maximum that the engine returns. For example, if Aurora MySQL returns at most 1 KB to Performance Insights, it can only collect and show 1 KB, even if the original query is larger. Thus, when you view or download the query, Performance Insights returns the same number of bytes.

If you use the AWS CLI or API, Performance Insights doesn't have the 4 KB limit enforced by the console. `DescribeDimensionKeys` and `GetResourceMetrics` return at most 500 bytes. `GetDimensionKeyDetails` returns the full query, but the size is subject to the engine limit.

Setting the SQL text limit for Aurora PostgreSQL DB instances

Aurora PostgreSQL handles text differently. You can set the text size limit with the DB instance parameter `track_activity_query_size`. This parameter has the following characteristics:

Default text size

On Aurora PostgreSQL version 9.6, the default setting for the `track_activity_query_size` parameter is 1,024 bytes. On Aurora PostgreSQL version 10 or higher, the default is 4,096 bytes.

Maximum text size

The limit for `track_activity_query_size` is 102,400 bytes for Aurora PostgreSQL version 12 and lower. The maximum is 1 MB for version 13 and higher.

If the engine returns 1 MB to Performance Insights, the console displays only the first 4 KB. If you download the query, you get the full 1 MB. In this case, viewing and downloading return different numbers of bytes. For more information about the `track_activity_query_size` DB instance parameter, see [Run-time Statistics](#) in the PostgreSQL documentation.

To increase the SQL text size, increase the `track_activity_query_size` limit. To modify the parameter, change the parameter setting in the parameter group that is associated with the Aurora PostgreSQL DB instance.

To change the setting when the instance uses the default parameter group

1. Create a new DB instance parameter group for the appropriate DB engine and DB engine version.
2. Set the parameter in the new parameter group.
3. Associate the new parameter group with the DB instance.

For information about setting a DB instance parameter, see [Modifying parameters in a DB parameter group \(p. 347\)](#).

Viewing and downloading SQL text in the Performance Insights dashboard

In the Performance Insights dashboard, you can view or download SQL text.

To view more SQL text in the Performance Insights dashboard

1. Open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. In the navigation pane, choose **Performance Insights**.
3. Choose a DB instance. The Performance Insights dashboard is displayed for that DB instance.

SQL statements with text larger than 500 bytes look similar to the following image.

The screenshot shows the 'Top SQL' section of the Performance Insights dashboard. It includes a search bar labeled 'Filter sql', a legend for 'Load by waits (AAS)', and three rows of SQL statements with their execution times:

SQL Statement	Execution Time
delete from authors where id < (select * from (select max(id) - ? from authors)	1.07
select count(*) from authors where id < (select max(id) - ? from authors)	0.99
select count(*) from authors where id < (select max(id) - 30 from autho	0.99

Below this, a 'SQL information' panel displays the full SQL text for the third statement, along with 'SQL ID' and 'Digest ID' links.

- Examine the SQL information section to view more of the SQL text.

The Performance Insights dashboard can display up to 4,096 bytes for each SQL statement.

- (Optional) Choose **Copy** to copy the displayed SQL statement, or choose **Download** to download the SQL statement to view the SQL text up to the DB engine limit.

Note

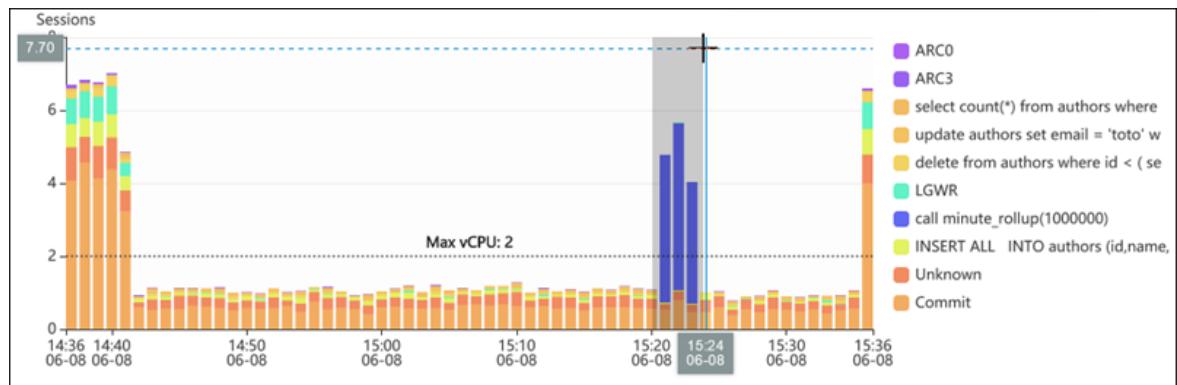
To copy or download the SQL statement, disable pop-up blockers.

Zooming In on the DB Load chart

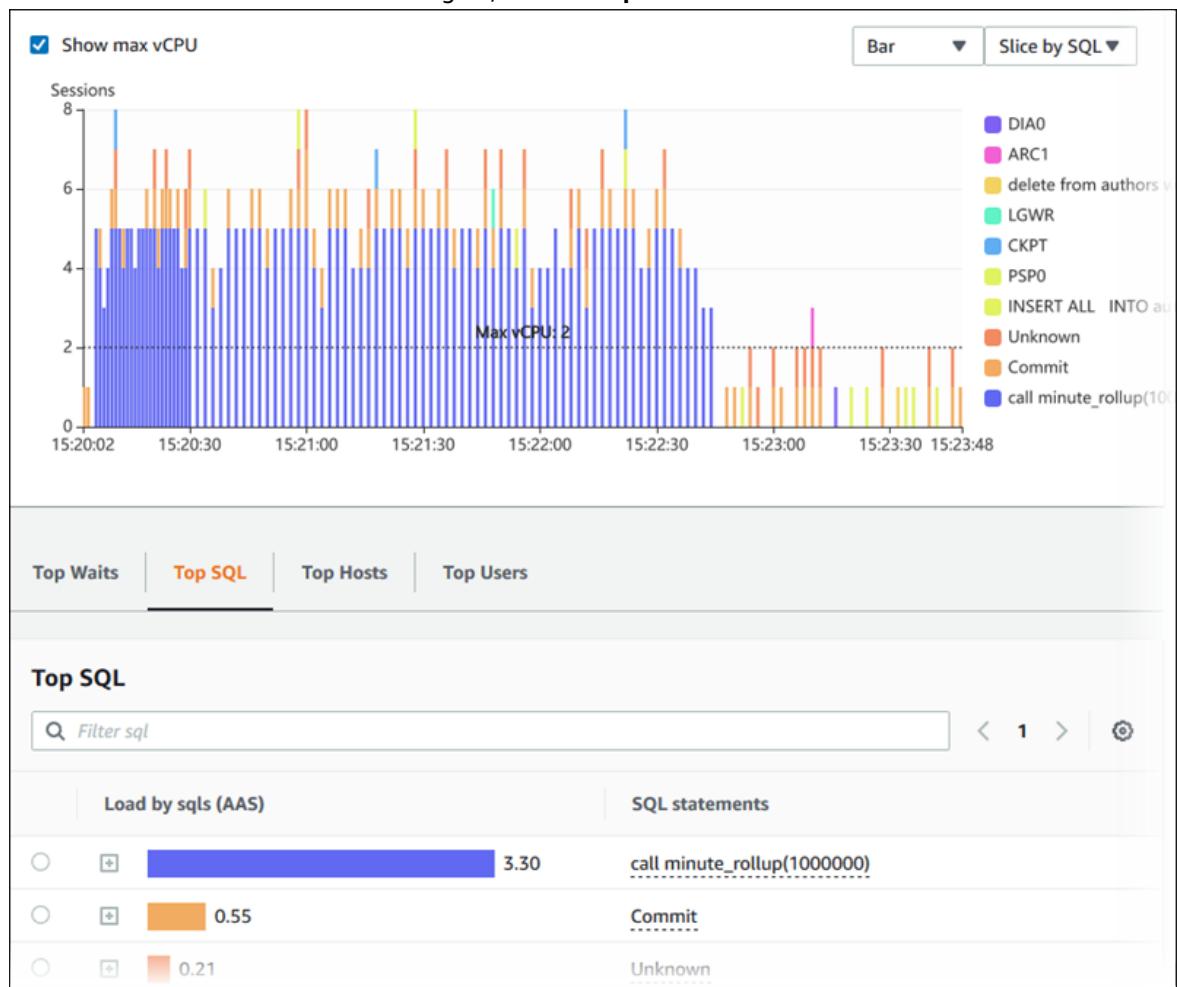
You can use other features of the Performance Insights user interface to help analyze performance data.

Click-and-Drag Zoom In

In the Performance Insights interface, you can choose a small portion of the load chart and zoom in on the detail.



To zoom in on a portion of the load chart, choose the start time and drag to the end of the time period you want. When you do this, the selected area is highlighted. When you release the mouse, the load chart zooms in on the selected AWS Region, and the **Top items** table is recalculated.



Retrieving metrics with the Performance Insights API

When Performance Insights is enabled, the API provides visibility into instance performance. Amazon CloudWatch Logs provides the authoritative source for vended monitoring metrics for AWS services.

Performance Insights offers a domain-specific view of database load measured as average active sessions (AAS). This metric appears to API consumers as a two-dimensional time-series dataset. The time dimension of the data provides DB load data for each time point in the queried time range. Each time point decomposes overall load in relation to the requested dimensions, such as `SQL`, `Wait-event`, `User`, or `Host`, measured at that time point.

Amazon RDS Performance Insights monitors your Amazon Aurora cluster so that you can analyze and troubleshoot database performance. One way to view Performance Insights data is in the AWS Management Console. Performance Insights also provides a public API so that you can query your own data. You can use the API to do the following:

- Offload data into a database
- Add Performance Insights data to existing monitoring dashboards
- Build monitoring tools

To use the Performance Insights API, enable Performance Insights on one of your Amazon RDS DB instances. For information about enabling Performance Insights, see [Enabling and disabling Performance Insights \(p. 577\)](#). For more information about the Performance Insights API, see the [Amazon RDS Performance Insights API Reference](#).

The Performance Insights API provides the following operations.

Performance Insights action	AWS CLI command	Description
DescribeDimensionKeys	<code>aws pi describe-dimension-keys</code>	Retrieves the top N dimension keys for a metric for a specific time period.
GetDimensionKeyDetails	<code>aws pi get-dimension-key-details</code>	Retrieves the attributes of the specified dimension group for a DB instance or data source. For example, if you specify a SQL ID, and if the dimension details are available, <code>GetDimensionKeyDetails</code> retrieves the full text of the dimension <code>db.sql.statement</code> associated with this ID. This operation is useful because <code>GetResourceMetrics</code> and <code>DescribeDimensionKeys</code> don't support retrieval of large SQL statement text.
GetResourceMetadata	<code>aws pi get-resource-metadata</code>	Retrieve the metadata for different features. For example, the metadata might indicate that a feature is turned on or off on a specific DB instance.
GetResourceMetrics	<code>aws pi get-resource-metrics</code>	Retrieves Performance Insights metrics for a set of data sources over a time period. You can provide specific dimension groups and dimensions, and provide aggregation and filtering criteria for each group.

Performance Insights action	AWS CLI command	Description
ListAvailableResourceDimensions	aws pi list-available-resource-dimensions	Retrieve the dimensions that can be queried for each specified metric type on a specified instance.
ListAvailableResourceMetrics	aws pi list-available-resource-metrics	Retrieve all available metrics of the specified metric types that can be queried for a specified DB instance.

Topics

- [AWS CLI for Performance Insights \(p. 609\)](#)
- [Retrieving time-series metrics \(p. 609\)](#)
- [AWS CLI examples for Performance Insights \(p. 610\)](#)

AWS CLI for Performance Insights

You can view Performance Insights data using the AWS CLI. You can view help for the AWS CLI commands for Performance Insights by entering the following on the command line.

```
aws pi help
```

If you don't have the AWS CLI installed, see [Installing the AWS Command Line Interface](#) in the *AWS CLI User Guide* for information about installing it.

Retrieving time-series metrics

The GetResourceMetrics operation retrieves one or more time-series metrics from the Performance Insights data. GetResourceMetrics requires a metric and time period, and returns a response with a list of data points.

For example, the AWS Management Console uses GetResourceMetrics to populate the **Counter Metrics** chart and the **Database Load** chart, as seen in the following image.



All metrics returned by `GetResourceMetrics` are standard time-series metrics, with the exception of `db.load`. This metric is displayed in the **Database Load** chart. The `db.load` metric is different from the other time-series metrics because you can break it into subcomponents called *dimensions*. In the previous image, `db.load` is broken down and grouped by the waits states that make up the `db.load`.

Note

`GetResourceMetrics` can also return the `db.sampleload` metric, but the `db.load` metric is appropriate in most cases.

For information about the counter metrics returned by `GetResourceMetrics`, see [Performance Insights counter metrics \(p. 653\)](#).

The following calculations are supported for the metrics:

- Average – The average value for the metric over a period of time. Append `.avg` to the metric name.
- Minimum – The minimum value for the metric over a period of time. Append `.min` to the metric name.
- Maximum – The maximum value for the metric over a period of time. Append `.max` to the metric name.
- Sum – The sum of the metric values over a period of time. Append `.sum` to the metric name.
- Sample count – The number of times the metric was collected over a period of time. Append `.sample_count` to the metric name.

For example, assume that a metric is collected for 300 seconds (5 minutes), and that the metric is collected one time each minute. The values for each minute are 1, 2, 3, 4, and 5. In this case, the following calculations are returned:

- Average – 3
- Minimum – 1
- Maximum – 5
- Sum – 15
- Sample count – 5

For information about using the `get-resource-metrics` AWS CLI command, see [get-resource-metrics](#).

For the `--metric-queries` option, specify one or more queries that you want to get results for. Each query consists of a mandatory `Metric` and optional `GroupBy` and `Filter` parameters. The following is an example of a `--metric-queries` option specification.

```
{  
    "Metric": "string",  
    "GroupBy": {  
        "Group": "string",  
        "Dimensions": ["string", ...],  
        "Limit": integer  
    },  
    "Filter": {"string": "string"  
    ...}
```

AWS CLI examples for Performance Insights

The following examples show how to use the AWS CLI for Performance Insights.

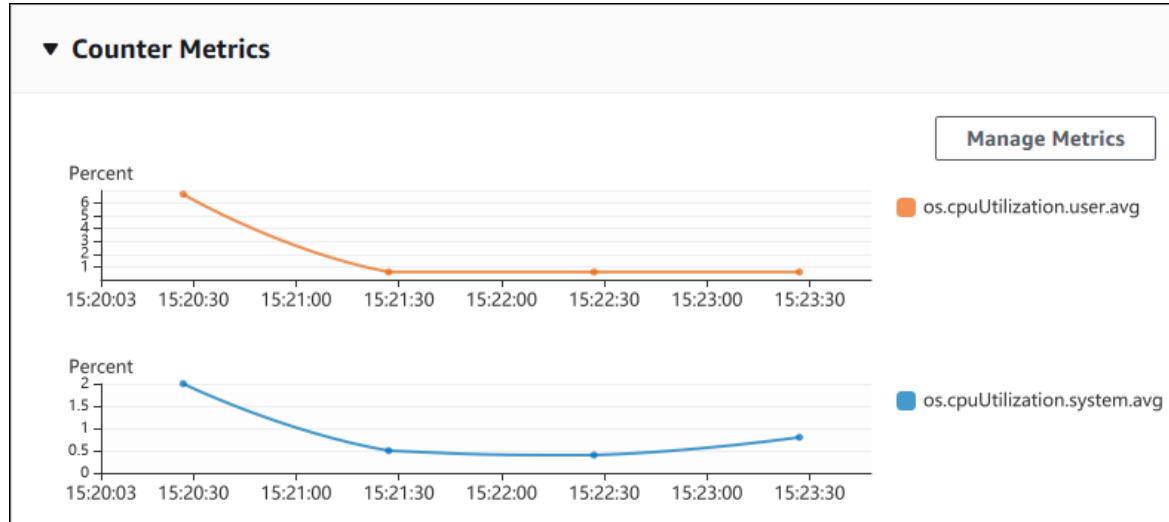
Topics

- [Retrieving counter metrics \(p. 611\)](#)

- [Retrieving the DB load average for top wait events \(p. 613\)](#)
- [Retrieving the DB load average for top SQL \(p. 615\)](#)
- [Retrieving the DB load average filtered by SQL \(p. 617\)](#)
- [Retrieving the full text of a SQL statement \(p. 620\)](#)

Retrieving counter metrics

The following screenshot shows two counter metrics charts in the AWS Management Console.



The following example shows how to gather the same data that the AWS Management Console uses to generate the two counter metric charts.

For Linux, macOS, or Unix:

```
aws pi get-resource-metrics \
--service-type RDS \
--identifier db-ID \
--start-time 2018-10-30T00:00:00Z \
--end-time 2018-10-30T01:00:00Z \
--period-in-seconds 60 \
--metric-queries '[{"Metric": "os.cpuUtilization.user.avg" },
 {"Metric": "os.cpuUtilization.idle.avg"}]'
```

For Windows:

```
aws pi get-resource-metrics ^
--service-type RDS ^
--identifier db-ID ^
--start-time 2018-10-30T00:00:00Z ^
--end-time 2018-10-30T01:00:00Z ^
--period-in-seconds 60 ^
--metric-queries '[{"Metric": "os.cpuUtilization.user.avg" },
 {"Metric": "os.cpuUtilization.idle.avg"}]'
```

You can also make a command easier to read by specifying a file for the `--metrics-query` option. The following example uses a file called `query.json` for the option. The file has the following contents.

```
[
```

```
{  
    "Metric": "os.cpuUtilization.user.avg"  
},  
{  
    "Metric": "os.cpuUtilization.idle.avg"  
}  
]
```

Run the following command to use the file.

For Linux, macOS, or Unix:

```
aws pi get-resource-metrics \  
    --service-type RDS \  
    --identifier db-ID \  
    --start-time 2018-10-30T00:00:00Z \  
    --end-time 2018-10-30T01:00:00Z \  
    --period-in-seconds 60 \  
    --metric-queries file://query.json
```

For Windows:

```
aws pi get-resource-metrics ^  
    --service-type RDS ^  
    --identifier db-ID ^  
    --start-time 2018-10-30T00:00:00Z ^  
    --end-time 2018-10-30T01:00:00Z ^  
    --period-in-seconds 60 ^  
    --metric-queries file://query.json
```

The preceding example specifies the following values for the options:

- **--service-type** – RDS for Amazon RDS
- **--identifier** – The resource ID for the DB instance
- **--start-time** and **--end-time** – The ISO 8601 DateTime values for the period to query, with multiple supported formats

It queries for a one-hour time range:

- **--period-in-seconds** – 60 for a per-minute query
- **--metric-queries** – An array of two queries, each just for one metric.

The metric name uses dots to classify the metric in a useful category, with the final element being a function. In the example, the function is `avg` for each query. As with Amazon CloudWatch, the supported functions are `min`, `max`, `total`, and `avg`.

The response looks similar to the following.

```
{  
    "Identifier": "db-XXX",  
    "AlignedStartTime": 1540857600.0,  
    "AlignedEndTime": 1540861200.0,  
    "MetricList": [  
        { //A list of key/datapoints  
            "Key": {  
                "Metric": "os.cpuUtilization.user.avg" //Metric1  
            },  
            "DataPoints": [  
                {  
                    "Time": 1540857600.0,  
                    "Value": 0.5  
                },  
                {  
                    "Time": 1540857620.0,  
                    "Value": 0.5  
                },  
                {  
                    "Time": 1540857640.0,  
                    "Value": 0.5  
                },  
                {  
                    "Time": 1540857660.0,  
                    "Value": 0.5  
                },  
                {  
                    "Time": 1540857680.0,  
                    "Value": 0.5  
                },  
                {  
                    "Time": 1540857700.0,  
                    "Value": 0.5  
                },  
                {  
                    "Time": 1540857720.0,  
                    "Value": 0.5  
                },  
                {  
                    "Time": 1540857740.0,  
                    "Value": 0.5  
                },  
                {  
                    "Time": 1540857760.0,  
                    "Value": 0.5  
                },  
                {  
                    "Time": 1540857780.0,  
                    "Value": 0.5  
                },  
                {  
                    "Time": 1540857800.0,  
                    "Value": 0.5  
                },  
                {  
                    "Time": 1540857820.0,  
                    "Value": 0.5  
                },  
                {  
                    "Time": 1540857840.0,  
                    "Value": 0.5  
                },  
                {  
                    "Time": 1540857860.0,  
                    "Value": 0.5  
                },  
                {  
                    "Time": 1540857880.0,  
                    "Value": 0.5  
                },  
                {  
                    "Time": 1540857900.0,  
                    "Value": 0.5  
                },  
                {  
                    "Time": 1540857920.0,  
                    "Value": 0.5  
                },  
                {  
                    "Time": 1540857940.0,  
                    "Value": 0.5  
                },  
                {  
                    "Time": 1540857960.0,  
                    "Value": 0.5  
                },  
                {  
                    "Time": 1540857980.0,  
                    "Value": 0.5  
                },  
                {  
                    "Time": 1540858000.0,  
                    "Value": 0.5  
                }  
            ]  
        }  
    ]  
}
```

```

//Each list of datapoints has the same timestamps and same number of items
{
    "Timestamp": 1540857660.0, //Minute1
    "Value": 4.0
},
{
    "Timestamp": 1540857720.0, //Minute2
    "Value": 4.0
},
{
    "Timestamp": 1540857780.0, //Minute 3
    "Value": 10.0
}
//... 60 datapoints for the os.cpuUtilization.user.avg metric
]
},
{
    "Key": {
        "Metric": "os.cpuUtilization.idle.avg" //Metric2
    },
    "DataPoints": [
        {
            "Timestamp": 1540857660.0, //Minute1
            "Value": 12.0
        },
        {
            "Timestamp": 1540857720.0, //Minute2
            "Value": 13.5
        }
        //... 60 datapoints for the os.cpuUtilization.idle.avg metric
    ]
}
] //end of MetricList
} //end of response

```

The response has an `Identifier`, `AlignedStartTime`, and `AlignedEndTime`. Because the `--period-in-seconds` value was 60, the start and end times have been aligned to the minute. If the `--period-in-seconds` was 3600, the start and end times would have been aligned to the hour.

The `MetricList` in the response has a number of entries, each with a `Key` and a `DataPoints` entry. Each `DataProvider` has a `Timestamp` and a `Value`. Each `DataPoints` list has 60 data points because the queries are for per-minute data over an hour, with `Timestamp1/Minute1`, `Timestamp2/Minute2`, and so on, up to `Timestamp60/Minute60`.

Because the query is for two different counter metrics, there are two elements in the response `MetricList`.

Retrieving the DB load average for top wait events

The following example is the same query that the AWS Management Console uses to generate a stacked area line graph. This example retrieves the `db.load.avg` for the last hour with load divided according to the top seven wait events. The command is the same as the command in [Retrieving counter metrics \(p. 611\)](#). However, the `query.json` file has the following contents.

```

[
    {
        "Metric": "db.load.avg",
        "GroupBy": { "Group": "db.wait_event", "Limit": 7 }
    }
]

```

Run the following command.

For Linux, macOS, or Unix:

```
aws pi get-resource-metrics \
--service-type RDS \
--identifier db-ID \
--start-time 2018-10-30T00:00:00Z \
--end-time 2018-10-30T01:00:00Z \
--period-in-seconds 60 \
--metric-queries file://query.json
```

For Windows:

```
aws pi get-resource-metrics ^
--service-type RDS ^
--identifier db-ID ^
--start-time 2018-10-30T00:00:00Z ^
--end-time 2018-10-30T01:00:00Z ^
--period-in-seconds 60 ^
--metric-queries file://query.json
```

The example specifies the metric of db.load.avg and a GroupBy of the top seven wait events. For details about valid values for this example, see [DimensionGroup](#) in the *Performance Insights API Reference*.

The response looks similar to the following.

```
{
    "Identifier": "db-XXX",
    "AlignedStartTime": 1540857600.0,
    "AlignedEndTime": 1540861200.0,
    "MetricList": [
        { //A list of key/datapoints
            "Key": {
                //A Metric with no dimensions. This is the total db.load.avg
                "Metric": "db.load.avg"
            },
            "DataPoints": [
                //Each list of datapoints has the same timestamps and same number of items
                {
                    "Timestamp": 1540857660.0, //Minute1
                    "Value": 0.5166666666666667
                },
                {
                    "Timestamp": 1540857720.0, //Minute2
                    "Value": 0.3833333333333336
                },
                {
                    "Timestamp": 1540857780.0, //Minute 3
                    "Value": 0.2666666666666666
                }
                //... 60 datapoints for the total db.load.avg key
            ]
        },
        {
            "Key": {
                //Another key. This is db.load.avg broken down by CPU
                "Metric": "db.load.avg",
                "Dimensions": {
                    "db.wait_event.name": "CPU",
                    "db.wait_event.type": "CPU"
                }
            },
        }
    ]
}
```

```

    "DataPoints": [
        {
            "Timestamp": 1540857660.0, //Minute1
            "Value": 0.35
        },
        {
            "Timestamp": 1540857720.0, //Minute2
            "Value": 0.15
        },
        //... 60 datapoints for the CPU key
    ]
},
//... In total we have 8 key/datapoints entries, 1) total, 2-8) Top Wait Events
] //end of MetricList
} //end of response
}

```

In this response, there are eight entries in the `MetricList`. There is one entry for the total `db.load.avg`, and seven entries each for the `db.load.avg` divided according to one of the top seven wait events. Unlike in the first example, because there was a grouping dimension, there must be one key for each grouping of the metric. There can't be only one key for each metric, as in the basic counter metric use case.

Retrieving the DB load average for top SQL

The following example groups `db.wait_events` by the top 10 SQL statements. There are two different groups for SQL statements:

- `db.sql` – The full SQL statement, such as `select * from customers where customer_id = 123`
- `db.sql_tokenized` – The tokenized SQL statement, such as `select * from customers where customer_id = ?`

When analyzing database performance, it can be useful to consider SQL statements that only differ by their parameters as one logic item. So, you can use `db.sql_tokenized` when querying. However, especially when you're interested in explain plans, sometimes it's more useful to examine full SQL statements with parameters, and query grouping by `db.sql`. There is a parent-child relationship between tokenized and full SQL, with multiple full SQL (children) grouped under the same tokenized SQL (parent).

The command in this example is the similar to the command in [Retrieving the DB load average for top wait events \(p. 613\)](#). However, the `query.json` file has the following contents.

```
[
{
    "Metric": "db.load.avg",
    "GroupBy": { "Group": "db.sql_tokenized", "Limit": 10 }
}
]
```

The following example uses `db.sql_tokenized`.

For Linux, macOS, or Unix:

```
aws pi get-resource-metrics \
--service-type RDS \
--identifier db-ID \
--start-time 2018-10-29T00:00:00Z \
```

```
--end-time 2018-10-30T00:00:00Z \
--period-in-seconds 3600 \
--metric-queries file://query.json
```

For Windows:

```
aws pi get-resource-metrics ^
--service-type RDS ^
--identifier db-ID ^
--start-time 2018-10-29T00:00:00Z ^
--end-time 2018-10-30T00:00:00Z ^
--period-in-seconds 3600 ^
--metric-queries file://query.json
```

This example queries over 24 hours, with a one hour period-in-seconds.

The example specifies the metric of db.load.avg and a GroupBy of the top seven wait events. For details about valid values for this example, see [DimensionGroup](#) in the *Performance Insights API Reference*.

The response looks similar to the following.

```
{
    "AlignedStartTime": 1540771200.0,
    "AlignedEndTime": 1540857600.0,
    "Identifier": "db-XXX",

    "MetricList": [ //11 entries in the MetricList
        {
            "Key": { //First key is total
                "Metric": "db.load.avg"
            }
            "DataPoints": [ //Each DataPoints list has 24 per-hour Timestamps and a value
                {
                    "Value": 1.6964980544747081,
                    "Timestamp": 1540774800.0
                },
                //... 24 datapoints
            ]
        },
        {
            "Key": { //Next key is the top tokenized SQL
                "Dimensions": {
                    "db.sql_tokenized.statement": "INSERT INTO authors (id,name,email) VALUES\n( nextval(?) ,?,?)",
                    "db.sql_tokenized.db_id": "pi-2372568224",
                    "db.sql_tokenized.id": "AKIAIOSFODNN7EXAMPLE"
                },
                "Metric": "db.load.avg"
            },
            "DataPoints": [ //... 24 datapoints
            ]
        },
        // In total 11 entries, 10 Keys of top tokenized SQL, 1 total key
    ] //End of MetricList
} //End of response
```

This response has 11 entries in the MetricList (1 total, 10 top tokenized SQL), with each entry having 24 per-hour DataPoints.

For tokenized SQL, there are three entries in each dimensions list:

- `db.sql_tokenized.statement` – The tokenized SQL statement.
- `db.sql_tokenized.db_id` – Either the native database ID used to refer to the SQL, or a synthetic ID that Performance Insights generates for you if the native database ID isn't available. This example returns the `pi-2372568224` synthetic ID.
- `db.sql_tokenized.id` – The ID of the query inside Performance Insights.

In the AWS Management Console, this ID is called the Support ID. It's named this because the ID is data that AWS Support can examine to help you troubleshoot an issue with your database. AWS takes the security and privacy of your data extremely seriously, and almost all data is stored encrypted with your AWS KMS customer master key (CMK). Therefore, nobody inside AWS can look at this data. In the example preceding, both the `tokenized.statement` and the `tokenized.db_id` are stored encrypted. If you have an issue with your database, AWS Support can help you by referencing the Support ID.

When querying, it might be convenient to specify a Group in `GroupBy`. However, for finer-grained control over the data that's returned, specify the list of dimensions. For example, if all that is needed is the `db.sql_tokenized.statement`, then a `Dimensions` attribute can be added to the `query.json` file.

```
[  
  {  
    "Metric": "db.load.avg",  
    "GroupBy": {  
      "Group": "db.sql_tokenized",  
      "Dimensions": ["db.sql_tokenized.statement"],  
      "Limit": 10  
    }  
  }  
]
```

Retrieving the DB load average filtered by SQL



The preceding image shows that a particular query is selected, and the top average active sessions stacked area line graph is scoped to that query. Although the query is still for the top seven overall wait events, the value of the response is filtered. The filter causes it to take into account only sessions that are a match for the particular filter.

The corresponding API query in this example is similar to the command in [Retrieving the DB load average for top SQL \(p. 615\)](#). However, the query.json file has the following contents.

```
[  
 {  
     "Metric": "db.load.avg",  
     "GroupBy": { "Group": "db.wait_event", "Limit": 5 },  
     "Filter": { "db.sql_tokenized.id": "AKIAIOSFODNN7EXAMPLE" }  
 }  
]
```

For Linux, macOS, or Unix:

```
aws pi get-resource-metrics \  
  --service-type RDS \  
  --identifier db-ID \  
  --start-time 2018-10-30T00:00:00Z \  
  --end-time 2018-10-30T01:00:00Z \  
  --period-in-seconds 60 \  
  --metric-queries file://query.json
```

For Windows:

```
aws pi get-resource-metrics ^  
  --service-type RDS ^  
  --identifier db-ID ^  
  --start-time 2018-10-30T00:00:00Z ^  
  --end-time 2018-10-30T01:00:00Z ^  
  --period-in-seconds 60 ^  
  --metric-queries file://query.json
```

The response looks similar to the following.

```
{  
    "Identifier": "db-XXX",  
    "AlignedStartTime": 1556215200.0,  
    "MetricList": [  
        {  
            "Key": {  
                "Metric": "db.load.avg"  
            },  
            "DataPoints": [  
                {  
                    "Timestamp": 1556218800.0,  
                    "Value": 1.4878117913832196  
                },  
                {  
                    "Timestamp": 1556222400.0,  
                    "Value": 1.192823803967328  
                }  
            ]  
        },  
        {  
            "Key": {  
                "Metric": "db.load.avg",  
                "Dimensions": {  
                    "db.wait_event.type": "io",  
                    "db.wait_event.name": "wait/io/aurora_redo_log_flush"  
                }  
            },  
            "DataPoints": [
```

```
{
    "Timestamp": 1556218800.0,
    "Value": 1.1360544217687074
},
{
    "Timestamp": 1556222400.0,
    "Value": 1.058051341890315
}
]
},
{
    "Key": {
        "Metric": "db.load.avg",
        "Dimensions": {
            "db.wait_event.type": "io",
            "db.wait_event.name": "wait/io/table/sql/handler"
        }
    },
    "DataPoints": [
        {
            "Timestamp": 1556218800.0,
            "Value": 0.16241496598639457
        },
        {
            "Timestamp": 1556222400.0,
            "Value": 0.05163360560093349
        }
    ]
},
{
    "Key": {
        "Metric": "db.load.avg",
        "Dimensions": {
            "db.wait_event.type": "synch",
            "db.wait_event.name": "wait/synch/mutex/innodb/aurora_lock_thread_slot_futex"
        }
    },
    "DataPoints": [
        {
            "Timestamp": 1556218800.0,
            "Value": 0.11479591836734694
        },
        {
            "Timestamp": 1556222400.0,
            "Value": 0.013127187864644107
        }
    ]
},
{
    "Key": {
        "Metric": "db.load.avg",
        "Dimensions": {
            "db.wait_event.type": "CPU",
            "db.wait_event.name": "CPU"
        }
    },
    "DataPoints": [
        {
            "Timestamp": 1556218800.0,
            "Value": 0.05215419501133787
        },
        {
            "Timestamp": 1556222400.0,
            "Value": 0.05805134189031505
        }
    ]
}
```

```

        ],
    },
    {
        "Key": {
            "Metric": "db.load.avg",
            "Dimensions": {
                "db.wait_event.type": "synch",
                "db.wait_event.name": "wait/synch/mutex/innodb/lock_wait_mutex"
            }
        },
        "DataPoints": [
            {
                "Timestamp": 1556218800.0,
                "Value": 0.017573696145124718
            },
            {
                "Timestamp": 1556222400.0,
                "Value": 0.002333722287047841
            }
        ]
    }
],
"AlignedEndTime": 1556222400.0
} //end of response

```

In this response, all values are filtered according to the contribution of tokenized SQL AKIAIOSFODNN7EXAMPLE specified in the query.json file. The keys also might follow a different order than a query without a filter, because it's the top five wait events that affected the filtered SQL.

Retrieving the full text of a SQL statement

The following example retrieves the full text of a SQL statement for DB instance db-10BCD2EFGHIJ3KL4M5NO6PQRS5. The --group is db.sql, and the --group-identifier is db.sql.id. In this example, *my-sql-id* represents a SQL ID retrieved by invoking pi get-resource-metrics or pi describe-dimension-keys.

Run the following command.

For Linux, macOS, or Unix:

```
aws pi get-dimension-key-details \
--service-type RDS \
--identifier db-10BCD2EFGHIJ3KL4M5NO6PQRS5 \
--group db.sql \
--group-identifier my-sql-id \
--requested-dimensions statement
```

For Windows:

```
aws pi get-dimension-key-details ^
--service-type RDS ^
--identifier db-10BCD2EFGHIJ3KL4M5NO6PQRS5 ^
--group db.sql ^
--group-identifier my-sql-id ^
--requested-dimensions statement
```

In this example, the dimensions details are available. Thus, Performance Insights retrieves the full text of the SQL statement, without truncating it.

```
{
```

```
"Dimensions": [
  {
    "Value": "SELECT e.last_name, d.department_name FROM employees e, departments d
WHERE e.department_id=d.department_id",
    "Dimension": "db.sql.statement",
    "Status": "AVAILABLE"
  },
  ...
]
```

Logging Performance Insights calls using AWS CloudTrail

Performance Insights runs with AWS CloudTrail, a service that provides a record of actions taken by a user, role, or an AWS service in Performance Insights. CloudTrail captures all API calls for Performance Insights as events. This capture includes calls from the Amazon RDS console and from code calls to the Performance Insights API operations.

If you create a trail, you can enable continuous delivery of CloudTrail events to an Amazon S3 bucket, including events for Performance Insights. If you don't configure a trail, you can still view the most recent events in the CloudTrail console in **Event history**. Using the data collected by CloudTrail, you can determine certain information. This information includes the request that was made to Performance Insights, the IP address the request was made from, who made the request, and when it was made. It also includes additional details.

To learn more about CloudTrail, see the [AWS CloudTrail User Guide](#).

Working with Performance Insights information in CloudTrail

CloudTrail is enabled on your AWS account when you create the account. When activity occurs in Performance Insights, that activity is recorded in a CloudTrail event along with other AWS service events in the CloudTrail console in **Event history**. You can view, search, and download recent events in your AWS account. For more information, see [Viewing Events with CloudTrail Event History](#) in *AWS CloudTrail User Guide*.

For an ongoing record of events in your AWS account, including events for Performance Insights, create a trail. A *trail* enables CloudTrail to deliver log files to an Amazon S3 bucket. By default, when you create a trail in the console, the trail applies to all AWS Regions. The trail logs events from all AWS Regions in the AWS partition and delivers the log files to the Amazon S3 bucket that you specify. Additionally, you can configure other AWS services to further analyze and act upon the event data collected in CloudTrail logs. For more information, see the following topics in *AWS CloudTrail User Guide*:

- [Overview for Creating a Trail](#)
- [CloudTrail Supported Services and Integrations](#)
- [Configuring Amazon SNS Notifications for CloudTrail](#)
- [Receiving CloudTrail Log Files from Multiple Regions](#) and [Receiving CloudTrail Log Files from Multiple Accounts](#)

All Performance Insights operations are logged by CloudTrail and are documented in the [Performance Insights API Reference](#). For example, calls to the `DescribeDimensionKeys` and `GetResourceMetrics` operations generate entries in the CloudTrail log files.

Every event or log entry contains information about who generated the request. The identity information helps you determine the following:

- Whether the request was made with root or IAM user credentials.
- Whether the request was made with temporary security credentials for a role or federated user.
- Whether the request was made by another AWS service.

For more information, see the [CloudTrail userIdentity Element](#).

Performance Insights log file entries

A *trail* is a configuration that enables delivery of events as log files to an Amazon S3 bucket that you specify. CloudTrail log files contain one or more log entries. An *event* represents a single request from any source. Each event includes information about the requested operation, the date and time of the operation, request parameters, and so on. CloudTrail log files aren't an ordered stack trace of the public API calls, so they don't appear in any specific order.

The following example shows a CloudTrail log entry that demonstrates the `GetResourceMetrics` operation.

```
{  
    "eventVersion": "1.05",  
    "userIdentity": {  
        "type": "IAMUser",  
        "principalId": "AKIAIOSFODNN7EXAMPLE",  
        "arn": "arn:aws:iam::123456789012:user/johndoe",  
        "accountId": "123456789012",  
        "accessKeyId": "AKIAI44QH8DHBEXAMPLE",  
        "userName": "johndoe"  
    },  
    "eventTime": "2019-12-18T19:28:46Z",  
    "eventSource": "pi.amazonaws.com",  
    "eventName": "GetResourceMetrics",  
    "awsRegion": "us-east-1",  
    "sourceIPAddress": "72.21.198.67",  
    "userAgent": "aws-cli/1.16.240 Python/3.7.4 Darwin/18.7.0 botocore/1.12.230",  
    "requestParameters": {  
        "identifier": "db-YTDU5J5V66X7CXSCVDFD2V3SZM",  
        "metricQueries": [  
            {  
                "metric": "os.cpuUtilization.user.avg"  
            },  
            {  
                "metric": "os.cpuUtilization.idle.avg"  
            }  
        ],  
        "startTime": "Dec 18, 2019 5:28:46 PM",  
        "periodInSeconds": 60,  
        "endTime": "Dec 18, 2019 7:28:46 PM",  
        "serviceType": "RDS"  
    },  
    "responseElements": null,  
    "requestID": "9ffbe15c-96b5-4fe6-bed9-9fccff1a0525",  
    "eventID": "08908de0-2431-4e2e-ba7b-f5424f908433",  
    "eventType": "AwsApiCall",  
    "recipientAccountId": "123456789012"  
}
```

Analyzing performance anomalies with DevOps Guru for RDS

Amazon DevOps Guru is a fully managed operations service that helps developers and operators improve the performance and availability of their applications. DevOps Guru offloads the tasks associated with identifying operational issues so that you can quickly implement recommendations to improve your application. To learn more, see [What is Amazon DevOps Guru?](#) in the *Amazon DevOps Guru User Guide*.

DevOps Guru detects, analyzes, and makes recommendations for operational issues for all Amazon RDS DB engines. DevOps Guru for RDS extends this capability by applying machine learning to Performance Insights metrics for Amazon Aurora databases. These monitoring features allow DevOps Guru for RDS to detect and diagnose performance bottlenecks and recommend specific corrective actions. To learn more, see [Overview of DevOps Guru for RDS](#) in the *Amazon DevOps Guru User Guide*.

Topics

- [Benefits of DevOps Guru for RDS \(p. 623\)](#)
- [How DevOps Guru for RDS works \(p. 624\)](#)
- [Setting up DevOps Guru for RDS \(p. 624\)](#)

Benefits of DevOps Guru for RDS

If you're responsible for an Amazon Aurora database, you might not know that an event or regression that is affecting that database is occurring. When you learn about the issue, you might not know why it's occurring or what to do about it. Rather than turning to a database administrator (DBA) for help or relying on third-party tools, you can follow recommendations from DevOps Guru for RDS.

You gain the following advantages from the detailed analysis of DevOps Guru for RDS:

Fast diagnosis

DevOps Guru for RDS continuously monitors and analyzes database telemetry. Performance Insights, Enhanced Monitoring, and Amazon CloudWatch collect telemetry data for your database cluster. DevOps Guru for RDS uses statistical and machine learning techniques to mine this data and detect anomalies. To learn more about telemetry data, see [Monitoring DB load with Performance Insights on Amazon Aurora](#) and [Monitoring the OS by using Enhanced Monitoring](#) in the *Amazon Aurora User Guide*.

Fast resolution

Each anomaly identifies the performance issue and suggests avenues of investigation or corrective actions. For example, DevOps Guru for RDS might recommend that you investigate specific wait events. Or it might recommend that you tune your application pool settings to limit the number of database connections. Based on these recommendations, you can resolve performance issues more quickly than by troubleshooting manually.

Deep knowledge of Amazon engineers

To detect performance issues and help you resolve bottlenecks, DevOps Guru for RDS relies on machine learning (ML). Amazon database engineers contributed to the development of the DevOps Guru for RDS findings, which encapsulate many years of managing hundreds of thousands of databases. By drawing on this collective knowledge, DevOps Guru for RDS can teach you best practices.

How DevOps Guru for RDS works

DevOps Guru for RDS collects data about your Aurora databases from Amazon RDS Performance Insights. The most important metric is `DBLoad`. DevOps Guru for RDS consumes the Performance Insights metrics, analyzes them with machine learning, and publishes insights to the dashboard.

Insights

An *insight* is a collection of related anomalies that were detected by DevOps Guru. If DevOps Guru for RDS finds performance issues in your Amazon Aurora DB instances, it publishes an insight in the DevOps Guru dashboard. To learn more about insights, see [Working with insights in DevOps Guru](#) in the [Amazon DevOps Guru User Guide](#).

Anomalies

In DevOps Guru for RDS, an *anomaly* is a pattern that deviates from what is considered normal performance for your Amazon Aurora database.

Causal anomalies

A *causal anomaly* is a top-level anomaly within an insight. **Database load (DB load)** is the causal anomaly for DevOps Guru for RDS.

An anomaly measures performance impact by assigning a severity level of **High**, **Medium**, or **Low**. To learn more, see [Key concepts for DevOps Guru for RDS](#) in the [Amazon DevOps Guru User Guide](#).

If DevOps Guru detects an anomaly on your DB instance, you're alerted in the **Databases** page of the RDS console. To go to the anomaly page from the RDS console, choose the link in the alert message. The RDS console also alerts you in the page for your Amazon Aurora cluster.

Contextual anomalies

A *contextual anomaly* is a finding within **Database load (DB load)**. Each contextual anomaly describes a specific Amazon Aurora performance issue that requires investigation. For example, DevOps Guru for RDS might recommend that you consider increasing CPU capacity or investigate wait events that are contributing to DB load. [Amazon Aurora versions \(p. 5\)](#)

Important

We recommend that you test any changes on a test instance before modifying a production instance. In this way, you understand the impact of the change.

To learn more, see [Analyzing anomalies in Amazon Aurora clusters](#) in the [Amazon DevOps Guru User Guide](#).

Setting up DevOps Guru for RDS

To allow DevOps Guru for RDS to publish insights for an Amazon Aurora database, complete the following tasks.

Topics

- [Turning on Performance Insights for your Amazon Aurora DB instances \(p. 625\)](#)
- [Configuring access policies for DevOps Guru for RDS \(p. 625\)](#)
- [Adding Amazon Aurora resources to your DevOps Guru coverage \(p. 625\)](#)

Turning on Performance Insights for your Amazon Aurora DB instances

DevOps Guru for RDS relies on Performance Insights for its data. Without Performance Insights, DevOps Guru publishes anomalies, but doesn't include the detailed analysis and recommendations.

When you create or modify a DB instance, you can turn on Performance Insights. For more information, see [Enabling and disabling Performance Insights \(p. 577\)](#).

Configuring access policies for DevOps Guru for RDS

For your IAM user or role to access DevOps Guru for RDS, it must have either of the following policies:

- The AWS managed policy `AmazonRDSFullAccess`
- A customer managed policy that allows the following actions:
 - `pi:GetResourceMetrics`
 - `pi:DescribeDimensionKeys`
 - `pi:GetDimensionKeyDetails`

For more information, see [Configuring access policies for Performance Insights \(p. 582\)](#).

Adding Amazon Aurora resources to your DevOps Guru coverage

To set up DevOps Guru for the first time, perform the following steps:

1. Sign up to AWS if you aren't already signed up.
2. Determine coverage for your resources.

To allow DevOps Guru for RDS to generate anomalies for your Amazon Aurora DB instances, specify the instances that you want to be covered. By default, DevOps Guru analyzes all supported AWS resources in your AWS Region and account. You can also specify individual resources by using AWS CloudFormation stacks or applying tags. To learn more, see [Adding Amazon Aurora resources to your DevOps Guru coverage](#) in the *Amazon DevOps Guru User Guide*.

3. Identify Amazon SNS topics.

Use one or two Amazon SNS topics to generate notifications about important DevOps Guru for RDS events. An example is when an insight is created for an Amazon Aurora DB instance. In this way, you know about issues that DevOps Guru for RDS finds as soon as possible. To learn more, see [Identify your Amazon SNS notifications topic](#) in the *Amazon DevOps Guru User Guide*.

For more information, see [Setting up Amazon DevOps Guru](#) in the *Amazon DevOps Guru User Guide*.

Monitoring OS metrics with Enhanced Monitoring

With Enhanced Monitoring, you can monitor the operating system of your DB instance in real time. When you want to see how different processes or threads use the CPU, Enhanced Monitoring metrics are useful.

Topics

- [Overview of Enhanced Monitoring \(p. 626\)](#)
- [Setting up and enabling Enhanced Monitoring \(p. 627\)](#)
- [Viewing OS metrics in the RDS console \(p. 630\)](#)
- [Viewing OS metrics using CloudWatch Logs \(p. 632\)](#)

Overview of Enhanced Monitoring

Amazon RDS provides metrics in real time for the operating system (OS) that your DB instance runs on. You can view all the system metrics and process information for your RDS DB instances on the console. You can manage which metrics you want to monitor for each instance and customize the dashboard according to your requirements.

RDS delivers the metrics from Enhanced Monitoring into your Amazon CloudWatch Logs account. You can create metrics filters in CloudWatch from CloudWatch Logs and display the graphs on the CloudWatch dashboard. You can consume the Enhanced Monitoring JSON output from CloudWatch Logs in a monitoring system of your choice. For more information, see [Enhanced Monitoring](#) in the Amazon RDS FAQs.

Topics

- [Differences between CloudWatch and Enhanced Monitoring metrics \(p. 626\)](#)
- [Retention of Enhanced Monitoring metrics \(p. 626\)](#)
- [Cost of Enhanced Monitoring \(p. 627\)](#)

Differences between CloudWatch and Enhanced Monitoring metrics

A *hypervisor* creates and runs virtual machines (VMs). Using a hypervisor, an instance can support multiple guest VMs by virtually sharing memory and CPU. CloudWatch gathers metrics about CPU utilization from the hypervisor for a DB instance. In contrast, Enhanced Monitoring gathers its metrics from an agent on the DB instance.

You might find differences between the CloudWatch and Enhanced Monitoring measurements, because the hypervisor layer performs a small amount of work. The differences can be greater if your DB instances use smaller instance classes. In this scenario, more virtual machines (VMs) are probably managed by the hypervisor layer on a single physical instance.

Retention of Enhanced Monitoring metrics

By default, Enhanced Monitoring metrics are stored for 30 days in the CloudWatch Logs. This retention period is different from typical CloudWatch metrics.

To modify the amount of time the metrics are stored in the CloudWatch Logs, change the retention for the `RDSOSMetrics` log group in the CloudWatch console. For more information, see [Change log data retention in CloudWatch logs](#) in the *Amazon CloudWatch Logs User Guide*.

Cost of Enhanced Monitoring

Enhanced Monitoring metrics are stored in the CloudWatch Logs instead of in CloudWatch metrics. The cost of Enhanced Monitoring depends on the following factors:

- You are charged for Enhanced Monitoring only if you exceed the free tier provided by Amazon CloudWatch Logs. Charges are based on CloudWatch Logs data transfer and storage rates.
- The amount of information transferred for an RDS instance is directly proportional to the defined granularity for the Enhanced Monitoring feature. A smaller monitoring interval results in more frequent reporting of OS metrics and increases your monitoring cost. To manage costs, set different granularities for different instances in your accounts.
- Usage costs for Enhanced Monitoring are applied for each DB instance that Enhanced Monitoring is enabled for. Monitoring a large number of DB instances is more expensive than monitoring only a few.
- DB instances that support a more compute-intensive workload have more OS process activity to report and higher costs for Enhanced Monitoring.

For more information about pricing, see [Amazon CloudWatch pricing](#).

Setting up and enabling Enhanced Monitoring

To use Enhanced Monitoring, you must create an IAM role, and then enable Enhanced Monitoring.

Topics

- [Creating an IAM role for Enhanced Monitoring \(p. 627\)](#)
- [Turning Enhanced Monitoring on and off \(p. 628\)](#)
- [Protecting against the confused deputy problem \(p. 630\)](#)

Creating an IAM role for Enhanced Monitoring

Enhanced Monitoring requires permission to act on your behalf to send OS metric information to CloudWatch Logs. You grant Enhanced Monitoring permissions using an AWS Identity and Access Management (IAM) role. You can either create this role when you enable Enhanced Monitoring or create it beforehand.

Topics

- [Creating the IAM role when you enable Enhanced Monitoring \(p. 627\)](#)
- [Creating the IAM role before you enable Enhanced Monitoring \(p. 628\)](#)

Creating the IAM role when you enable Enhanced Monitoring

When you enable Enhanced Monitoring in the RDS console, Amazon RDS can create the required IAM role for you. The role is named `rds-monitoring-role`. RDS uses this role for the specified DB instance or read replica.

To create the IAM role when enabling Enhanced Monitoring

1. Follow the steps in [Turning Enhanced Monitoring on and off \(p. 628\)](#).
2. Set **Monitoring Role** to **Default** in the step where you choose a role.

Creating the IAM role before you enable Enhanced Monitoring

You can create the required role before you enable Enhanced Monitoring. When you enable Enhanced Monitoring, specify your new role's name. You must create this required role if you enable Enhanced Monitoring using the AWS CLI or the RDS API.

The user that enables Enhanced Monitoring must be granted the `PassRole` permission. For more information, see Example 2 in [Granting a user permissions to pass a role to an AWS service](#) in the *IAM User Guide*.

To create an IAM role for Amazon RDS enhanced monitoring

1. Open the [IAM console](#) at <https://console.aws.amazon.com>.
2. In the navigation pane, choose **Roles**.
3. Choose **Create role**.
4. Choose the **AWS service** tab, and then choose **RDS** from the list of services.
5. Choose **RDS - Enhanced Monitoring**, and then choose **Next: Permissions**.
6. Ensure that the **Attached permissions policy** page shows **AmazonRDSEnhancedMonitoringRole**, and then choose **Next: Tags**.
7. On the **Add tags** page, choose **Next: Review**.
8. For **Role Name**, enter a name for your role. For example, enter **emaccess**.
The trusted entity for your role is the AWS service **monitoring.rds.amazonaws.com**.
9. Choose **Create role**.

Turning Enhanced Monitoring on and off

You can turn Enhanced Monitoring on and off using the AWS Management Console, AWS CLI, or RDS API. You choose the RDS DB instances on which you want to turn on Enhanced Monitoring. You can set different granularities for metric collection on each DB instance.

Console

You can enable Enhanced Monitoring when you create a DB cluster or read replica, or when you modify a DB cluster. If you modify a DB instance to enable Enhanced Monitoring, you don't need to reboot your DB instance for the change to take effect.

You can enable Enhanced Monitoring in the RDS console when you do one of the following actions in the **Databases** page:

- **Create a DB cluster** – Choose **Create database**.
- **Create a read replica** – Choose **Actions**, then **Create read replica**.
- **Modify a DB instance** – Choose **Modify**.

To turn Enhanced Monitoring on or off in the RDS console

1. Scroll to **Additional configuration**.
2. In **Monitoring**, choose **Enable Enhanced Monitoring** for your DB instance or read replica. To turn Enhanced Monitoring off, choose **Disable Enhanced Monitoring**.
3. Set the **Monitoring Role** property to the IAM role that you created to permit Amazon RDS to communicate with Amazon CloudWatch Logs for you, or choose **Default** to have RDS create a role for you named `rds-monitoring-role`.

- Set the **Granularity** property to the interval, in seconds, between points when metrics are collected for your DB instance or read replica. The **Granularity** property can be set to one of the following values: 1, 5, 10, 15, 30, or 60.

Note

The fastest that the RDS console refreshes is every 5 seconds. If you set the granularity to 1 second in the RDS console, you still see updated metrics only every 5 seconds. You can retrieve 1-second metric updates by using CloudWatch Logs.

AWS CLI

To enable Enhanced Monitoring using the AWS CLI, in the following commands, set the `--monitoring-interval` option to a value other than 0 and set the `--monitoring-role-arn` option to the role you created in [Creating an IAM role for Enhanced Monitoring \(p. 627\)](#).

- `create-db-instance`
- `create-db-instance-read-replica`
- `modify-db-instance`

The `--monitoring-interval` option specifies the interval, in seconds, between points when Enhanced Monitoring metrics are collected. Valid values for the option are 0, 1, 5, 10, 15, 30, and 60.

To turn off Enhanced Monitoring using the AWS CLI, set the `--monitoring-interval` option to 0 in these commands.

Example

The following example turn on Enhanced Monitoring for a DB instance:

For Linux, macOS, or Unix:

```
aws rds modify-db-instance \
--db-instance-identifier mydbinstance \
--monitoring-interval 30 \
--monitoring-role-arn arn:aws:iam::123456789012:role/emaccess
```

For Windows:

```
aws rds modify-db-instance ^
--db-instance-identifier mydbinstance ^
--monitoring-interval 30 ^
--monitoring-role-arn arn:aws:iam::123456789012:role/emaccess
```

RDS API

To turn on Enhanced Monitoring using the RDS API, set the `MonitoringInterval` parameter to a value other than 0 and set the `MonitoringRoleArn` parameter to the role you created in [Creating an IAM role for Enhanced Monitoring \(p. 627\)](#). Set these parameters in the following actions:

- `CreateDBInstance`
- `CreateDBInstanceReadReplica`
- `ModifyDBInstance`

The `MonitoringInterval` parameter specifies the interval, in seconds, between points when Enhanced Monitoring metrics are collected. Valid values are 0, 1, 5, 10, 15, 30, and 60.

To turn off Enhanced Monitoring using the RDS API, set `MonitoringInterval` to 0.

Protecting against the confused deputy problem

The confused deputy problem is a security issue where an entity that doesn't have permission to perform an action can coerce a more-privileged entity to perform the action. In AWS, cross-service impersonation can result in the confused deputy problem. Cross-service impersonation can occur when one service (the *calling service*) calls another service (the *called service*). The calling service can be manipulated to use its permissions to act on another customer's resources in a way it should not otherwise have permission to access. To prevent this, AWS provides tools that help you protect your data for all services with service principals that have been given access to resources in your account. For more information, see [The confused deputy problem](#).

To limit the permissions to the resource that Amazon RDS can give another service, we recommend using the `aws:SourceArn` and `aws:SourceAccount` global condition context keys in a trust policy for your Enhanced Monitoring role. If you use both global condition context keys, they must use the same account ID.

The most effective way to protect against the confused deputy problem is to use the `aws:SourceArn` global condition context key with the full ARN of the resource. For Amazon RDS, set `aws:SourceArn` to `arn:aws:rds:Region:my-account-id:db/dbname`.

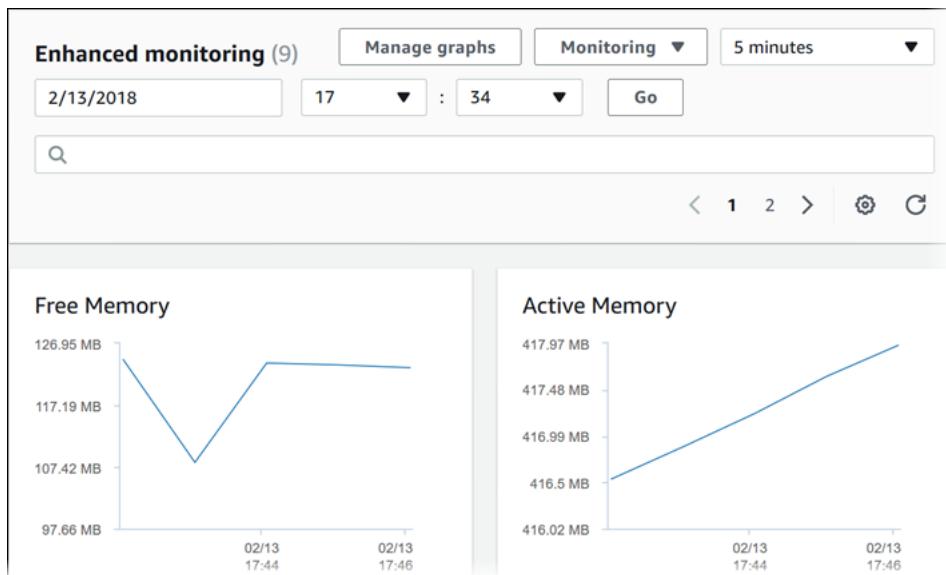
The following example uses the `aws:SourceArn` and `aws:SourceAccount` global condition context keys in a trust policy to prevent the confused deputy problem.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Principal": {  
                "Service": "rds.amazonaws.com"  
            },  
            "Action": "sts:AssumeRole",  
            "Condition": {  
                "StringLike": {  
                    "aws:SourceArn": "arn:aws:rds:Region:my-account-id:db/dbname"  
                },  
                "StringEquals": {  
                    "aws:SourceAccount": "my-account-id"  
                }  
            }  
        }  
    ]  
}
```

Viewing OS metrics in the RDS console

You can view OS metrics reported by Enhanced Monitoring in the RDS console by choosing **Enhanced monitoring for Monitoring**.

The Enhanced Monitoring page is shown following.



If you want to see details for the processes running on your DB instance, choose **OS process list** for **Monitoring**.

The **Process List** view is shown following.

Process List						
<input type="text"/> Filter process list						
NAME	VIRT	RES	CPU%	MEM%	VMLIMIT	
postgres [3181] ^t	283.55 MB	17.11 MB	0.02	1.72		
postgres: rdsadmin rdsadmin localhost(40156) idle [2953] ^t	384.7 MB	9.51 MB	0.02	0.95		

The Enhanced Monitoring metrics shown in the **Process list** view are organized as follows:

- **RDS child processes** – Shows a summary of the RDS processes that support the DB instance, for example `aurora` for Amazon Aurora DB clusters. Process threads appear nested beneath the parent process. Process threads show CPU utilization only as other metrics are the same for all threads for the process. The console displays a maximum of 100 processes and threads. The results are a combination of the top CPU consuming and memory consuming processes and threads. If there are more than 50 processes and more than 50 threads, the console displays the top 50 consumers in each category. This display helps you identify which processes are having the greatest impact on performance.
- **RDS processes** – Shows a summary of the resources used by the RDS management agent, diagnostics monitoring processes, and other AWS processes that are required to support RDS DB instances.
- **OS processes** – Shows a summary of the kernel and system processes, which generally have minimal impact on performance.

The items listed for each process are:

- **VIRT** – Displays the virtual size of the process.
- **RES** – Displays the actual physical memory being used by the process.
- **CPU%** – Displays the percentage of the total CPU bandwidth being used by the process.
- **MEM%** – Displays the percentage of the total memory being used by the process.

The monitoring data that is shown in the RDS console is retrieved from Amazon CloudWatch Logs. You can also retrieve the metrics for a DB instance as a log stream from CloudWatch Logs. For more information, see [Viewing OS metrics using CloudWatch Logs \(p. 632\)](#).

Enhanced Monitoring metrics are not returned during the following:

- A failover of the DB instance.
- Changing the instance class of the DB instance (scale compute).

Enhanced Monitoring metrics are returned during a reboot of a DB instance because only the database engine is rebooted. Metrics for the operating system are still reported.

Viewing OS metrics using CloudWatch Logs

After you have enabled Enhanced Monitoring for your DB instance, you can view the metrics for your DB instance using CloudWatch Logs, with each log stream representing a single DB instance being monitored. The log stream identifier is the resource identifier (`DbiResourceId`) for the DB instance.

To view Enhanced Monitoring log data

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. If necessary, choose the region that your DB instance is in. For more information, see [Regions and endpoints](#) in the *Amazon Web Services General Reference*.
3. Choose **Logs** in the navigation pane.
4. Choose **RDSOSMetrics** from the list of log groups.
5. Choose the log stream that you want to view from the list of log streams.

Metrics reference for Amazon Aurora

In this reference, you can find descriptions of Amazon Aurora metrics for Amazon CloudWatch, Performance Insights, and Enhanced Monitoring.

Topics

- [Amazon CloudWatch metrics for Amazon Aurora \(p. 633\)](#)
- [Amazon CloudWatch dimensions for Aurora \(p. 649\)](#)
- [Availability of Aurora metrics in the Amazon RDS console \(p. 649\)](#)
- [Amazon CloudWatch metrics for Performance Insights \(p. 652\)](#)
- [Performance Insights counter metrics \(p. 653\)](#)
- [OS metrics in Enhanced Monitoring \(p. 660\)](#)

Amazon CloudWatch metrics for Amazon Aurora

The AWS/RDS namespace includes the following metrics that apply to database entities running on Amazon Aurora. Some metrics apply to either Aurora MySQL, Aurora PostgreSQL, or both. Furthermore, some metrics are specific to a DB cluster, primary DB instance, replica DB instance, or all DB instances.

For Aurora global database metrics, see [Amazon CloudWatch metrics for write forwarding \(p. 263\)](#). For Aurora parallel query metrics, see [Monitoring parallel query \(p. 898\)](#).

Topics

- [Cluster-level metrics for Amazon Aurora \(p. 633\)](#)
- [Instance-level metrics for Amazon Aurora \(p. 639\)](#)

Cluster-level metrics for Amazon Aurora

The following table describes metrics that are specific to Aurora clusters.

Amazon Aurora cluster-level metrics

Metric	Console name	Description	Applies to	Units
AuroraGlobalDBDataTransferBytes	Aurora Global DB Data Transfer Bytes (Bytes)	In an Aurora Global Database, the amount of redo log data transferred from the master AWS Region to a secondary AWS Region.	Aurora MySQL and Aurora PostgreSQL	Bytes
AuroraGlobalDBProgressLag		In an Aurora Global Database, the measure of how far the secondary cluster is behind the primary cluster for both user transactions and system transactions.	Aurora PostgreSQL	Milliseconds
AuroraGlobalDBReplicatedWriteIO	Aurora Global DB Replicated Write IO	In an Aurora Global Database, the number of write I/O operations replicated from the	Aurora MySQL and Aurora PostgreSQL	Count

Metric	Console name	Description	Applies to	Units
		primary AWS Region to the cluster volume in a secondary AWS Region. The billing calculations for the secondary AWS Regions in a global database use VolumeWriteIOPS to account for writes performed within the cluster. The billing calculations for the primary AWS Region in a global database use VolumeWriteIOPS to account for the write activity within that cluster, and AuroraGlobalDBReplicatedWriteIO to account for cross-Region replication within the global database.		
AuroraGlobalDBReplicationLag	Aurora Global DB Replication Lag (Milliseconds)	For an Aurora Global Database, the amount of lag when replicating updates from the primary AWS Region.	Aurora MySQL and Aurora PostgreSQL	Milliseconds
AuroraGlobalDBRPOLag		In an Aurora Global Database, the recovery point objective (RPO) lag time. This metric measures how far the secondary cluster is behind the primary cluster for user transactions.	Aurora PostgreSQL	Milliseconds

Metric	Console name	Description	Applies to	Units
AuroraVolumeBytesLeftTotal		<p>The remaining available space for the cluster volume. As the cluster volume grows, this value decreases. If it reaches zero, the cluster reports an out-of-space error.</p> <p>If you want to detect whether your Aurora cluster is approaching the size limit of 128 tebibytes (TiB), this value is simpler and more reliable to monitor than VolumeBytesUsed. AuroraVolumeBytesLeftTotal takes into account storage used for internal housekeeping and other allocations that don't affect your storage billing.</p> <p>This parameter is available in more recent Aurora versions. For Aurora MySQL with MySQL 5.6 compatibility, use Aurora version 1.19.5 or higher. For Aurora MySQL with MySQL 5.7 compatibility, use Aurora version 2.04.5 or higher.</p>	Aurora MySQL	Bytes
BacktrackChangeRecordsCreated	Backtrack Change Records Creation Rate (Count)	The number of backtrack change records created over 5 minutes for your DB cluster.	Aurora MySQL	Count per 5 minutes
BacktrackChangeRecordsStored	Backtrack Change Records Stored (Count)	The number of backtrack change records used by your DB cluster.	Aurora MySQL	Count

Metric	Console name	Description	Applies to	Units
BackupRetentionPeriodStorageUsed	Backup Retention Period Storage Used (GiB)	The total amount of backup storage used to support the point-in-time restore feature within the Aurora DB cluster's backup retention window. This amount is included in the total reported by the <code>TotalBackupStorageBilled</code> metric. It is computed separately for each Aurora cluster. For instructions, see Understanding Aurora backup storage usage (p. 494) .	Aurora MySQL and Aurora PostgreSQL	Bytes
ServerlessDatabaseCapacity		The current capacity of an Aurora Serverless v1 DB cluster.	Aurora MySQL and Aurora PostgreSQL	Count
SnapshotStorageUsed	Snapshot Storage Used (GiB)	The total amount of backup storage consumed by all Aurora snapshots for an Aurora DB cluster outside its backup retention window. This amount is included in the total reported by the <code>TotalBackupStorageBilled</code> metric. It is computed separately for each Aurora cluster. For instructions, see Understanding Aurora backup storage usage (p. 494) .	Aurora MySQL and Aurora PostgreSQL	Bytes
TotalBackupStorageBilled	Total Backup Storage Used (GiB)	The total amount of backup storage in bytes for which you are billed for a given Aurora DB cluster. The metric includes the backup storage measured by the <code>BackupRetentionPeriodStorageUsed</code> and <code>SnapshotStorageUsed</code> metrics. This metric is computed separately for each Aurora cluster. For instructions, see Understanding Aurora backup storage usage (p. 494) .	Aurora MySQL and Aurora PostgreSQL	Bytes

Metric	Console name	Description	Applies to	Units
VolumeBytesUsed	Volume Bytes Used (GiB)	<p>The amount of storage used by your Aurora DB instance.</p> <p>This value affects the cost of the Aurora DB cluster (for pricing information, see the Amazon RDS product page).</p> <p>This value doesn't reflect some internal storage allocations that don't affect storage billing. Thus, you can anticipate out-of-space issues more accurately by testing whether <code>AuroraVolumeBytesLeftTotal</code> is approaching zero instead of comparing <code>VolumeBytesUsed</code> against the storage limit of 128 TiB.</p>	Aurora MySQL and Aurora PostgreSQL	Bytes

Metric	Console name	Description	Applies to	Units
VolumeReadIOPS	Volume Read IOPS (Count)	<p>The number of billed read I/O operations from a cluster volume within a 5-minute interval.</p> <p>Billed read operations are calculated at the cluster volume level, aggregated from all instances in the Aurora DB cluster, and then reported at 5-minute intervals. The value is calculated by taking the value of the Read operations metric over a 5-minute period. You can determine the amount of billed read operations per second by taking the value of the Billed read operations metric and dividing by 300 seconds. For example, if the Billed read operations returns 13,686, then the billed read operations per second is 45 ($13,686 / 300 = 45.62$).</p> <p>You accrue billed read operations for queries that request database pages that aren't in the buffer cache and must be loaded from storage. You might see spikes in billed read operations as query results are read from storage and then loaded into the buffer cache.</p> <p>Tip If your Aurora MySQL cluster uses parallel query, you might see an increase in VolumeReadIOPS values. Parallel queries don't use the buffer pool. Thus, although the queries are fast, this optimized processing</p>	Aurora MySQL and Aurora PostgreSQL	Count per 5 minutes

Metric	Console name	Description	Applies to	Units
		can result in an increase in read operations and associated charges.		
VolumeWriteIOPS	Volume Write IOPS (Count)	The number of write disk I/O operations to the cluster volume, reported at 5-minute intervals. For a detailed description of how billed write operations are calculated, see VolumeReadIOPs .	Aurora MySQL and Aurora PostgreSQL	Count per 5 minutes

Instance-level metrics for Amazon Aurora

The following instance-specific CloudWatch metrics apply to all Aurora MySQL and Aurora PostgreSQL instances unless noted otherwise.

Amazon Aurora instance-level metrics

Metric	Console Name	Description	Applies to	Units
AbortedClients		The number of client connections that have not been closed properly.	Aurora MySQL	Count
ActiveTransactions	Active Transactions (Count)	The average number of current transactions executing on an Aurora database instance per second. By default, Aurora doesn't enable this metric. To begin measuring this value, set <code>innodb_monitor_enable='all'</code> in the DB parameter group for a specific DB instance.	Aurora MySQL	Count per second
AuroraBinlogReplicaLag	Aurora Binlog Replica Lag (Seconds)	The amount of time that a binary log replica DB cluster running on Aurora MySQL-Compatible Edition lags behind the binary log replication source. A lag means that the source is generating records faster than the replica can apply them.	Primary for Aurora MySQL	Seconds

Metric	Console Name	Description	Applies to	Units
		<p>This metric reports different values depending on the engine version:</p> <p>Aurora MySQL version 1 and 2</p> <pre>The Seconds_Behind_Master field of the MySQL SHOW SLAVE STATUS</pre> <p>Aurora MySQL version 3</p> <pre>SHOW REPLICA STATUS</pre> <p>You can use this metric to monitor errors and replica lag in a cluster that acts as a binary log replica. The metric value indicates the following:</p> <ul style="list-style-type: none"> A high value <ul style="list-style-type: none"> The replica is lagging the replication source. 0 or a value close to 0 The replica process is active and current. -1 Aurora can't determine the lag, which can happen during replica setup or when the replica is in an error state. Because binary log replication only occurs on the writer instance of the cluster, we recommend using the version of this metric associated with the WRITER role. For more information about administering replication, see Replicating Amazon Aurora MySQL DB clusters across AWS Regions (p. 922). 		

Metric	Console Name	Description	Applies to	Units
		For more information about troubleshooting, see Amazon Aurora MySQL replication issues (p. 1818) .		
AuroraReplicaLag	Aurora Replica Lag (Milliseconds)	For an Aurora replica, the amount of lag when replicating updates from the primary instance.	Replica for Aurora MySQL and Aurora PostgreSQL	Milliseconds
AuroraReplicaLagMaximum	Replica Lag Maximum (Milliseconds)	The maximum amount of lag between the primary instance and each Aurora DB instance in the DB cluster.	Primary for Aurora MySQL and Aurora PostgreSQL	Milliseconds
AuroraReplicaLagMinimum	Replica Lag Minimum (Milliseconds)	The minimum amount of lag between the primary instance and each Aurora DB instance in the DB cluster.	Primary for Aurora MySQL and Aurora PostgreSQL	Milliseconds
BacktrackWindowActual	Backtrack Window Actual (Minutes)	The difference between the target backtrack window and the actual backtrack window.	Primary for Aurora MySQL	Minutes
BacktrackWindowAlert	Backtrack Window Alert (Count)	The number of times that the actual backtrack window is smaller than the target backtrack window for a given period of time.	Primary for Aurora MySQL	Count
BlockedTransactions	Blocked Transactions (Count)	The average number of transactions in the database that are blocked per second.	Aurora MySQL	Count per second
BufferCacheHitRatio	Buffer Cache Hit Ratio (Percent)	The percentage of requests that are served by the buffer cache.	Aurora MySQL and Aurora PostgreSQL	Percentage
CommitLatency	Commit Latency (Milliseconds)	The average duration of commit operations.	Aurora MySQL and Aurora PostgreSQL	Milliseconds
CommitThroughput	Commit Throughput (Count/Second)	The average number of commit operations per second.	Aurora MySQL and Aurora PostgreSQL	Count per second

Metric	Console Name	Description	Applies to	Units
CPUCreditBalance	CPU Credit Balance (Count)	<p>The number of CPU credits that an instance has accumulated, reported at 5-minute intervals. You can use this metric to determine how long a DB instance can burst beyond its baseline performance level at a given rate.</p> <p>This metric applies only to db.t2.small and db.t2.medium instances for Aurora MySQL, and to db.t3 instances for Aurora PostgreSQL.</p>	Aurora MySQL and Aurora PostgreSQL	Count
CPUCreditUsage	CPU Credit Usage (Count)	<p>The number of CPU credits consumed during the specified period, reported at 5-minute intervals. This metric measures the amount of time during which physical CPUs have been used for processing instructions by virtual CPUs allocated to the DB instance.</p> <p>This metric applies only to db.t2.small and db.t2.medium instances for Aurora MySQL, and to db.t3 instances for Aurora PostgreSQL.</p>	Aurora MySQL and Aurora PostgreSQL	Count
CPUUtilization	CPU Utilization (Percent)	The percentage of CPU used by an Aurora DB instance.	Aurora MySQL and Aurora PostgreSQL	Percentage

Metric	Console Name	Description	Applies to	Units
DatabaseConnections	DB Connections (Count)	<p>The number of client network connections to the database instance.</p> <p>The number of database sessions can be higher than the metric value because the metric value doesn't include the following:</p> <ul style="list-style-type: none"> • Sessions that no longer have a network connection but which the database hasn't cleaned up • Sessions created by the database engine for its own purposes • Sessions created by the database engine's parallel execution capabilities • Sessions created by the database engine job scheduler • Amazon Aurora connections 	Aurora MySQL and Aurora PostgreSQL	Count
DDLLatency	DDL Latency (Milliseconds)	The average duration of requests such as example, create, alter, and drop requests.	Aurora MySQL	Milliseconds
DDLThroughput	DDL (Count/Second)	The average number of DDL requests per second.	Aurora MySQL	Count per second
Deadlocks	Deadlocks (Count)	The average number of deadlocks in the database per second.	Aurora MySQL and Aurora PostgreSQL	Count per second
DeleteLatency	Delete Latency (Milliseconds)	The average duration of delete operations.	Aurora MySQL	Milliseconds
DeleteThroughput	Delete Throughput (Count/Second)	The average number of delete queries per second.	Aurora MySQL	Count per second
DiskQueueDepth	Queue Depth (Count)	The number of outstanding read/write requests waiting to access the disk.	Aurora PostgreSQL	Count

Metric	Console Name	Description	Applies to	Units
DMLLatency	DML Latency (Milliseconds)	The average duration of inserts, updates, and deletes.	Aurora MySQL	Milliseconds
DMLThroughput	DML Throughput (Count/Second)	The average number of inserts, updates, and deletes per second.	Aurora MySQL	Count per second
EBSByteBalance%	EBS Byte Balance (Percent)	<p>The percentage of throughput credits remaining in the burst bucket of your RDS database. This metric is available for basic monitoring only.</p> <p>To find the instance sizes that support this metric, see the instance sizes with an asterisk (*) in the EBS optimized by default table in <i>Amazon EC2 User Guide for Linux Instances</i>. The Sum statistic is not applicable to this metric.</p>	Aurora MySQL and Aurora PostgreSQL	Percent
EBSIOBalance%	EBS IO Balance (Percent)	<p>The percentage of I/O credits remaining in the burst bucket of your RDS database. This metric is available for basic monitoring only.</p> <p>To find the instance sizes that support this metric, see the instance sizes with an asterisk (*) in the EBS optimized by default table in <i>Amazon EC2 User Guide for Linux Instances</i>. The Sum statistic is not applicable to this metric.</p> <p>This metric is different from <code>BurstBalance</code>. To learn how to use this metric, see Improving application performance and reducing costs with Amazon EBS-Optimized Instance burst capability.</p>	Aurora MySQL and Aurora PostgreSQL	Percent
EngineUptime	Engine Uptime (Seconds)	The amount of time that the instance has been running.	Aurora MySQL and Aurora PostgreSQL	Seconds

Metric	Console Name	Description	Applies to	Units
FreeableMemory	Freeable Memory (MB)	The amount of available random access memory.	Aurora MySQL and Aurora PostgreSQL	Bytes
FreeLocalStorage		<p>The amount of local storage available.</p> <p>Unlike for other DB engines, for Aurora DB instances this metric reports the amount of storage available to each DB instance. This value depends on the DB instance class (for pricing information, see the Amazon RDS product page). You can increase the amount of free storage space for an instance by choosing a larger DB instance class for your instance.</p>	Aurora MySQL and Aurora PostgreSQL	Bytes
InsertLatency	Insert Latency (Milliseconds)	The average duration of insert operations.	Aurora MySQL	Milliseconds
InsertThroughput	Insert Throughput (Count/Second)	The average number of insert operations per second.	Aurora MySQL	Count per second
LoginFailures	Login Failures (Count)	The average number of failed login attempts per second.	Aurora MySQL	Count per second
MaximumUsedTransactionID	MaximumUsedTransactionID	<p>Age of the oldest unvacuumed transaction ID, in transactions.</p> <p>If this value reaches 2,146,483,648 ($2^{31} - 1,000,000$), the database is forced into read-only mode, to avoid transaction ID wraparound. For more information, see Preventing transaction ID wraparound failures in the PostgreSQL documentation.</p>	Aurora PostgreSQL	Count

Metric	Console Name	Description	Applies to	Units
NetworkReceiveThroughput	Network Receive Throughput (MB/Second)	The amount of network throughput received from clients by each instance in the Aurora MySQL DB cluster. This throughput doesn't include network traffic between instances in the Aurora DB cluster and the cluster volume.	Aurora MySQL and Aurora PostgreSQL	Bytes per second (console shows Megabytes per second)
NetworkThroughput	Network Throughput (Byte/Second)	The amount of network throughput both received from and transmitted to clients by each instance in the Aurora MySQL DB cluster. This throughput doesn't include network traffic between instances in the DB cluster and the cluster volume.	Aurora MySQL and Aurora PostgreSQL	Bytes per second
NetworkTransmitThroughput	Network Transmit Throughput (MB/Second)	The amount of network throughput sent to clients by each instance in the Aurora DB cluster. This throughput doesn't include network traffic between instances in the DB cluster and the cluster volume.	Aurora MySQL and Aurora PostgreSQL	Bytes per second (console shows Megabytes per second)
NumBinaryLogFile		The number of binlog files generated.	Aurora MySQL	Count
Queries	Queries (Count/Second)	The average number of queries executed per second.	Aurora MySQL	Count per second
RDSToAuroraPostgreSQLReplicaLag		The lag when replicating updates from the primary RDS PostgreSQL instance to other nodes in the cluster.	Replica for Aurora PostgreSQL	Seconds
ReadIOPS	Read IOPS (Count/Second)	The average number of disk I/O operations per second. Aurora PostgreSQL-Compatible Edition reports read and write IOPS separately, in 1-minute intervals.	Aurora PostgreSQL	Count per second

Metric	Console Name	Description	Applies to	Units
ReadLatency	Read Latency (Milliseconds)	The average amount of time taken per disk I/O operation.	Aurora MySQL and Aurora PostgreSQL	Seconds
ReadThroughput	Read Throughput (MB/Second)	The average number of bytes read from disk per second.	Aurora PostgreSQL	Bytes per second
ReplicationSlotDiskUsage		The amount of disk space consumed by replication slot files.	Aurora PostgreSQL	Bytes
ResultSetCacheHitRatio	Result Set Cache Hit Ratio (Percent)	The percentage of requests that are served by the Resultset cache.	Aurora MySQL	Percentage
RollbackSegmentHistoryListLength		The undo logs that record committed transactions with delete-marked records. These records are scheduled to be processed by the InnoDB purge operation.	Aurora MySQL	Count
RowLockTime		The total time spent acquiring row locks for InnoDB tables.	Aurora MySQL	Milliseconds
SelectLatency	Select Latency (Milliseconds)	The average amount of time for select operations.	Aurora MySQL	Milliseconds
SelectThroughput	Select Throughput (Count/Second)	The average number of select queries per second.	Aurora MySQL	Count per second
StorageNetworkReceiveThroughput	Network Receive Throughput (MB/Second)	The amount of network throughput received from the Aurora storage subsystem by each instance in the DB cluster.	Aurora MySQL and Aurora PostgreSQL	Bytes per second
StorageNetworkThroughput	Network Throughput (Byte/Second)	The amount of network throughput received from and sent to the Aurora storage subsystem by each instance in the Aurora MySQL DB cluster.	Aurora MySQL and Aurora PostgreSQL	Bytes per second
StorageNetworkTransmitThroughput	Network Transmit Throughput (MB/Second)	The amount of network throughput sent to the Aurora storage subsystem by each instance in the Aurora MySQL DB cluster.	Aurora MySQL and Aurora PostgreSQL	Bytes per second
SumBinaryLogSize		The total size of the binlog files.	Aurora MySQL	Bytes

Metric	Console Name	Description	Applies to	Units
SwapUsage	Swap Usage (MB)	The amount of swap space used. This metric is available for the Aurora PostgreSQL DB instance classes db.t3.medium, db.t3.large, db.r4.large, db.r4.xlarge, db.r5.large, db.r5.xlarge, db.r6g.large, and db.r6g.xlarge. For Aurora MySQL, this metric applies only to db.t* DB instance classes.	Aurora MySQL and Aurora PostgreSQL	Bytes
TransactionLogsDiskUsage	Transaction Logs Disk Usage (MB)	The amount of disk space consumed by transaction logs on the Aurora PostgreSQL DB instance. This metric is generated only when Aurora PostgreSQL is using logical replication or AWS Database Migration Service. By default, Aurora PostgreSQL uses log records, not transaction logs. When transaction logs aren't in use, the value for this metric is -1.	Primary for Aurora PostgreSQL	Bytes
UpdateLatency	Update Latency (Milliseconds)	The average amount of time taken for update operations.	Aurora MySQL	Milliseconds
UpdateThroughput	Update Throughput (Count/Second)	The average number of updates per second.	Aurora MySQL	Count per second
WriteIOPS	Volume Write IOPS (Count)	The number of Aurora storage write records generated per second. This is more or less the number of log records generated by the database. These do not correspond to 8K page writes, and do not correspond to network packets sent.	Aurora PostgreSQL	Count per second
WriteLatency	Write Latency (Milliseconds)	The average amount of time taken per disk I/O operation.	Aurora MySQL and Aurora PostgreSQL	Seconds

Metric	Console Name	Description	Applies to	Units
WriteThroughput	Write Throughput (MB/Second)	The average number of bytes written to persistent storage every second.	Aurora PostgreSQL	Bytes per second

Amazon CloudWatch dimensions for Aurora

You can filter Aurora metrics data by using any dimension in the following table.

Dimension	Filters the requested data for . . .
DBInstanceIdentifier	A specific DB instance.
DBClusterIdentifier	A specific Amazon Aurora DB cluster.
DBClusterIdentifier, Role	A specific Aurora DB cluster, aggregating the metric by instance role (WRITER/READER). For example, you can aggregate metrics for all READER instances that belong to a cluster.
DatabaseClass	All instances in a database class. For example, you can aggregate metrics for all instances that belong to the database class db.r5.large.
EngineName	The identified engine name only. For example, you can aggregate metrics for all instances that have the engine name mysql.
SourceRegion	The specified Region only. For example, you can aggregate metrics for all DB instances in the us-east-1 Region.

Availability of Aurora metrics in the Amazon RDS console

Not all metrics provided by Amazon Aurora are available in the Amazon RDS console. You can view these metrics using tools such as the AWS CLI and CloudWatch API. Also, some metrics in the Amazon RDS console are either shown only for specific instance classes, or with different names and units of measurement.

Topics

- [Aurora metrics available in the Last Hour view \(p. 649\)](#)
- [Aurora metrics available in specific cases \(p. 650\)](#)
- [Aurora metrics that aren't available in the console \(p. 651\)](#)

Aurora metrics available in the Last Hour view

You can view a subset of categorized Aurora metrics in the default Last Hour view in the Amazon RDS console. The following table lists the categories and associated metrics displayed in the Amazon RDS console for an Aurora instance.

Category	Metrics
SQL	ActiveTransactions

Category	Metrics
	<pre>BlockedTransactions BufferCacheHitRatio CommitLatency CommitThroughput DatabaseConnections DDLLatency DDLTroughput Deadlocks DMILatency DMLThroughput LoginFailures ResultSetCacheHitRatio SelectLatency SelectThroughput</pre>
System	<pre>AuroraReplicaLag AuroraReplicaLagMaximum AuroraReplicaLagMinimum CPUCreditBalance CPUCreditUsage CPUUtilization FreeableMemory FreeLocalStorage NetworkReceiveThroughput</pre>
Deployment	<pre>AuroraReplicaLag BufferCacheHitRatio ResultSetCacheHitRatio SelectThroughput</pre>

Aurora metrics available in specific cases

In addition, some Aurora metrics are either shown only for specific instance classes, or only for DB instances, or with different names and different units of measurement:

- The `CPUCreditBalance` and `CPUCreditUsage` metrics are displayed only for Aurora MySQL `db.t2` instance classes and for Aurora PostgreSQL `db.t3` instance classes.
- The following metrics that are displayed with different names, as listed:

Metric	Display name
<code>AuroraReplicaLagMaximum</code>	Replica lag maximum
<code>AuroraReplicaLagMinimum</code>	Replica lag minimum
<code>DDLThroughput</code>	DDL
<code>NetworkReceiveThroughput</code>	Network throughput
<code>VolumeBytesUsed</code>	[Billed] Volume bytes used
<code>VolumeReadIOPS</code>	[Billed] Volume read IOPs
<code>VolumeWriteIOPS</code>	[Billed] Volume write IOPs

- The following metrics apply to an entire Aurora DB cluster, but are displayed only when viewing DB instances for an Aurora DB cluster in the Amazon RDS console:
 - `VolumeBytesUsed`
 - `VolumeReadIOPS`
 - `VolumeWriteIOPS`
- The following metrics are displayed in megabytes, instead of bytes, in the Amazon RDS console:
 - `FreeableMemory`
 - `FreeLocalStorage`
 - `NetworkReceiveThroughput`
 - `NetworkTransmitThroughput`

Aurora metrics that aren't available in the console

The following Aurora metrics aren't available in the Amazon RDS console:

- `AuroraBinlogReplicaLag`
- `DeleteLatency`
- `DeleteThroughput`
- `EngineUptime`
- `InsertLatency`
- `InsertThroughput`
- `NetworkThroughput`
- `Queries`
- `UpdateLatency`
- `UpdateThroughput`

Amazon CloudWatch metrics for Performance Insights

Performance Insights automatically publishes metrics to Amazon CloudWatch. The same data can be queried from Performance Insights, but having the metrics in CloudWatch makes it easy to add CloudWatch alarms. It also makes it easy to add the metrics to existing CloudWatch Dashboards.

Metric	Description
DBLoad	The number of active sessions for the DB engine. Typically, you want the data for the average number of active sessions. In Performance Insights, this data is queried as <code>db.load.avg</code> .
DBLoadCPU	The number of active sessions where the wait event type is CPU. In Performance Insights, this data is queried as <code>db.load.avg</code> , filtered by the wait event type CPU.
DBLoadNonCPU	The number of active sessions where the wait event type is not CPU.

Note

These metrics are published to CloudWatch only if there is load on the DB instance.

You can examine these metrics using the CloudWatch console, the AWS CLI, or the CloudWatch API.

For example, you can get the statistics for the `DBLoad` metric by running the [get-metric-statistics](#) command.

```
aws cloudwatch get-metric-statistics \
--region us-west-2 \
--namespace AWS/RDS \
--metric-name DBLoad \
--period 60 \
--statistics Average \
--start-time 1532035185 \
--end-time 1532036185 \
--dimensions Name=DBInstanceIdentifier,Value=db-loadtest-0
```

This example generates output similar to the following.

```
{
  "Datapoints": [
    {
      "Timestamp": "2021-07-19T21:30:00Z",
      "Unit": "None",
      "Average": 2.1
    },
    {
      "Timestamp": "2021-07-19T21:34:00Z",
      "Unit": "None",
      "Average": 1.7
    },
    {
      "Timestamp": "2021-07-19T21:35:00Z",
      "Unit": "None",
      "Average": 1.7
    }
  ]
}
```

```

    "Average": 2.8
  },
  {
    "Timestamp": "2021-07-19T21:31:00Z",
    "Unit": "None",
    "Average": 1.5
  },
  {
    "Timestamp": "2021-07-19T21:32:00Z",
    "Unit": "None",
    "Average": 1.8
  },
  {
    "Timestamp": "2021-07-19T21:29:00Z",
    "Unit": "None",
    "Average": 3.0
  },
  {
    "Timestamp": "2021-07-19T21:33:00Z",
    "Unit": "None",
    "Average": 2.4
  }
],
"Label": "DBLoad"
}

```

For more information about CloudWatch, see [What is Amazon CloudWatch?](#) in the *Amazon CloudWatch User Guide*.

Performance Insights counter metrics

Counter metrics are operating system and database performance metrics in the Performance Insights dashboard. To help identify and analyze performance problems, you can correlate counter metrics with DB load.

Topics

- [Performance Insights operating system counters \(p. 653\)](#)
- [Performance Insights counters for Aurora MySQL \(p. 655\)](#)
- [Performance Insights counters for Aurora PostgreSQL \(p. 659\)](#)

Performance Insights operating system counters

The following operating system counters are available with Performance Insights for Aurora PostgreSQL. You can find definitions for these metrics in [Viewing OS metrics using CloudWatch Logs \(p. 632\)](#).

Counter	Type	Metric
active	memory	os.memory.active
buffers	memory	os.memory.buffers
cached	memory	os.memory.cached
dirty	memory	os.memory.dirty
free	memory	os.memory.free
hugePagesFree	memory	os.memory.hugePagesFree

Counter	Type	Metric
hugePagesRsvd	memory	os.memory.hugePagesRsvd
hugePagesSize	memory	os.memory.hugePagesSize
hugePagesSurp	memory	os.memory.hugePagesSurp
hugePagesTotal	memory	os.memory.hugePagesTotal
inactive	memory	os.memory.inactive
mapped	memory	os.memory.mapped
pageTables	memory	os.memory.pageTables
slab	memory	os.memory.slab
total	memory	os.memory.total
writeback	memory	os.memory.writeback
guest	cpuUtilization	os.cpuUtilization.guest
idle	cpuUtilization	os.cpuUtilization.idle
irq	cpuUtilization	os.cpuUtilization.irq
nice	cpuUtilization	os.cpuUtilization.nice
steal	cpuUtilization	os.cpuUtilization.steal
system	cpuUtilization	os.cpuUtilization.system
total	cpuUtilization	os.cpuUtilization.total
user	cpuUtilization	os.cpuUtilization.user
wait	cpuUtilization	os.cpuUtilization.wait
avgQueueLen	diskIO	os.diskIO.avgQueueLen
avgReqSz	diskIO	os.diskIO.avgReqSz
await	diskIO	os.diskIO.await
readIOsPS	diskIO	os.diskIO.readIOsPS
readKb	diskIO	os.diskIO.readKb
readKbPS	diskIO	os.diskIO.readKbPS
rrqmPS	diskIO	os.diskIO.rrqmPS
tps	diskIO	os.diskIO.tps
util	diskIO	os.diskIO.util
writelOsPS	diskIO	os.diskIO.writelOsPS
writeKb	diskIO	os.diskIO.writeKb
writeKbPS	diskIO	os.diskIO.writeKbPS

Counter	Type	Metric
wrqmPS	diskIO	os.diskIO.wrqmPS
blocked	tasks	os.tasks.blocked
running	tasks	os.tasks.running
sleeping	tasks	os.tasks.sleeping
stopped	tasks	os.tasks.stopped
total	tasks	os.tasks.total
zombie	tasks	os.tasks.zombie
one	loadAverageMinute	os.loadAverageMinute.one
fifteen	loadAverageMinute	os.loadAverageMinute.fifteen
five	loadAverageMinute	os.loadAverageMinute.five
cached	swap	os.swap.cached
free	swap	os.swap.free
in	swap	os.swap.in
out	swap	os.swap.out
total	swap	os.swap.total
maxFiles	fileSys	os.fileSys.maxFiles
usedFiles	fileSys	os.fileSys.usedFiles
usedFilePercent	fileSys	os.fileSys.usedFilePercent
usedPercent	fileSys	os.fileSys.usedPercent
used	fileSys	os.fileSys.used
total	fileSys	os.fileSys.total
rx	network	os.network.rx
tx	network	os.network.tx
numVCPU	general	os.general.numVCPU

Performance Insights counters for Aurora MySQL

The following database counters are available with Performance Insights for Aurora MySQL.

Topics

- [Native counters for Aurora MySQL \(p. 656\)](#)
- [Non-native counters for Aurora MySQL \(p. 657\)](#)

Native counters for Aurora MySQL

You can find definitions for these native metrics in [Server Status Variables](#) in the MySQL documentation.

Counter	Type	Unit	Metric
Com_analyze	SQL	Queries per second	db.SQL.Com_analyze
Com_optimize	SQL	Queries per second	db.SQL.Com_optimize
Com_select	SQL	Queries per second	db.SQL.Com_select
Innodb_rows_deleted	SQL	Rows per second	db.SQL.Innodb_rows_deleted
Innodb_rows_inserted	SQL	Rows per second	db.SQL.Innodb_rows_inserted
Innodb_rows_read	SQL	Rows per second	db.SQL.Innodb_rows_read
Innodb_rows_updated	SQL	Rows per second	db.SQL.Innodb_rows_updated
Questions	SQL	Queries per second	db.SQL.Questions
Select_full_join	SQL	Queries per second	db.SQL.Select_full_join
Select_full_range_join	SQL	Queries per second	db.SQL.Select_full_range_join
Select_range	SQL	Queries per second	db.SQL.Select_range
Select_range_check	SQL	Queries per second	db.SQL.Select_range_check
Select_scan	SQL	Queries per second	db.SQL.Select_scan
Slow_queries	SQL	Queries per second	db.SQL.Slow_queries
Sort_merge_passes	SQL	Queries per second	db.SQL.Sort_merge_passes
Sort_range	SQL	Queries per second	db.SQL.Sort_range
Sort_rows	SQL	Queries per second	db.SQL.Sort_rows
Sort_scan	SQL	Queries per second	db.SQL.Sort_scan

Counter	Type	Unit	Metric
Table_locks_immediate	Locks	Requests per second	db.Locks.Table_locks_immediate
Table_locks_waited	Locks	Requests per second	db.Locks.Table_locks_waited
Innodb_row_lock_time	Locks	Milliseconds (average)	db.Locks.Innodb_row_lock_time
Aborted_clients	Users	Connections	db.Users.Aborted_clients
Aborted_connects	Users	Connections	db.Users.Aborted_connects
Threads_created	Users	Connections	db.Users.Threads_created
Threads_running	Users	Connections	db.Users.Threads_running
Created_tmp_disk_tables	Temp	Tables per second	db.Temp.Created_tmp_disk_tables
Created_tmp_tables	Temp	Tables per second	db.Temp.Created_tmp_tables
Innodb_buffer_pool_pages_data	Cache	Pages	db.Cache.Innodb_buffer_pool_pages_data
Innodb_buffer_pool_pages_total	Cache	Pages	db.Cache.Innodb_buffer_pool_pages_total
Innodb_buffer_pool_read_requests	Cache	Pages per second	db.Cache.Innodb_buffer_pool_read_requests
Innodb_buffer_pool_reads	Cache	Pages per second	db.Cache.Innodb_buffer_pool_reads
Opened_tables	Cache	Tables	db.Cache.Opened_tables
Opened_table_definitions	Cache	Tables	db.Cache.Opened_table_definitions
Qcache_hits	Cache	Queries	db.Cache.Qcache_hits

Non-native counters for Aurora MySQL

Non-native counter metrics are counters defined by Amazon RDS. A non-native metric can be a metric that you get with a specific query. A non-native metric also can be a derived metric, where two or more native counters are used in calculations for ratios, hit rates, or latencies.

Counter	Type	Metric	Description	Definition
innodb_buffer_pool_hits	Cache	db.Cache.innodb_buffer_pool_hits	The buffer pool reads that InnoDB could satisfy from the buffer pool.	innodb_buffer_pool_read_requests - innodb_buffer_pool_reads
innodb_buffer_pool_hit_rate	Cache	db.Cache.innodb_buffer_pool_hit_rate	The percentage of reads that InnoDB could satisfy from the buffer pool.	100 * innodb_buffer_pool_read_requests / (innodb_buffer_pool_read_requests + innodb_buffer_pool_reads)

Counter	Type	Metric	Description	Definition
innodb_buffer_pool_usage	Cache	db.Cache.innodb_buffer_pool_usage	The percentage of the InnoDB buffer pool that contains data (pages). Note When using compressed tables, this value can vary. For more information, see the information about Innodb_buffer_pool_pages_data and Innodb_buffer_pool_pages_total in Server Status Variables in the MySQL documentation.	Innodb_buffer_pool_pages_data / Innodb_buffer_pool_pages_total * 100.0
query_cache_hit_rate	Cache	db.Cache.query_cache_hit_rate	The hit ratio for the MySQL result set cache (query cache).	Qcache_hits / (QCache_hits + Com_select) * 100
innodb_rows_changed	SQL	db.SQL.innodb_rows_changed	The total number of row operations.	db.SQL.Innodb_rows_inserted + db.SQL.Innodb_rows_deleted + db.SQL.Innodb_rows_updated
active_transactions	Transactions	db.Transactions.active_transactions	The total number of active transactions.	SELECT COUNT(1) AS active_transactions FROM INFORMATION_SCHEMA.INNODB_TRX
innodb_deadlocks	Locks	db.Lock.innodb_deadlocks	The total number of deadlocks.	SELECT COUNT AS innodb_deadlocks FROM INFORMATION_SCHEMA.INNODB_METR WHERE NAME='lock_deadlocks'
innodb_lock_timeouts	Locks	db.Lock.innodb_lock_timeouts	The total number of deadlocks that timed out.	SELECT COUNT AS innodb_lock_timeouts FROM INFORMATION_SCHEMA.INNODB_METR WHERE NAME='lock_timeouts'
innodb_row_lock_waits	Locks	db.Lock.innodb_row_lock_waits	The total number of row locks that resulted in a wait.	SELECT COUNT AS innodb_row_lock_waits FROM INFORMATION_SCHEMA.INNODB_METR WHERE NAME='lock_row_lock_waits'

Performance Insights counters for Aurora PostgreSQL

The following database counters are available with Performance Insights for Aurora PostgreSQL.

Topics

- [Native Counters for Aurora PostgreSQL \(p. 659\)](#)
- [Non-native counters for Aurora PostgreSQL \(p. 660\)](#)

Native Counters for Aurora PostgreSQL

You can find definitions for these native metrics in [Viewing Statistics](#) in the PostgreSQL documentation.

Counter	Type	Unit	Metric
tup_deleted	SQL	Tuples per second	db.SQL.tup_deleted
tup_fetched	SQL	Tuples per second	db.SQL.tup_fetched
tup_inserted	SQL	Tuples per second	db.SQL.tup_inserted
tup_returned	SQL	Tuples per second	db.SQL.tup_returned
tup_updated	SQL	Tuples per second	db.SQL.tup_updated
buffers_checkpoint	Checkpoint	Blocks per second	db.Checkpoint.buffers_checkpoint
checkpoints_req	Checkpoint	Checkpoints per minute	db.Checkpoint.checkpoints_req
checkpoint_sync_time	Checkpoint	Milliseconds per checkpoint	db.Checkpoint.checkpoint_sync_time
checkpoints_timed	Checkpoint	Checkpoints per minute	db.Checkpoint.checkpoints_timed
checkpoint_write_time	Checkpoint	Milliseconds per checkpoint	db.Checkpoint.checkpoint_write_time
maxwritten_clean	Checkpoint	Bgwriter clean stops per minute	db.Checkpoint.maxwritten_clean
active_transactions	Transactions	Transactions	db.Transactions.active_transactions
blocked_transactions	Transactions	Transactions	db.Transactions.blocked_transactions
max_used_xact_ids	Transactions	Transactions	db.Transactions.max_used_xact_ids
xact_commit	Transactions	Commits per second	db.Transactions.xact_commit
xact_rollback	Transactions	Rollbacks per second	db.Transactions.xact_rollback
blk_read_time	I/O	Milliseconds	db.IO.blk_read_time
blks_read	I/O	Blocks per second	db.IO.blks_read
buffers_backend	I/O	Blocks per second	db.IO.buffers_backend
buffers_backend_fsync	I/O	Blocks per second	db.IO.buffers_backend_fsync
buffers_clean	I/O	Blocks per second	db.IO.buffers_clean
blks_hit	Cache	Blocks per second	db.Cache.blks_hit

Counter	Type	Unit	Metric
buffers_alloc	Cache	Blocks per second	db.Cache.buffers_alloc
temp_files	Temp	Files per minute	db.Temp.temp_files
numbackends	User	Connections	db.User.numbackends
deadlocks	Concurrency	Deadlocks per minute	db.Concurrency.deadlocks
archived_count	WAL	Files per minute	db.WAL.archived_count
archive_failed_count	WAL	Files per minute	db.WAL.archive_failed_count

Non-native counters for Aurora PostgreSQL

Non-native counter metrics are counters defined by Amazon Aurora. A non-native metric can be a metric that you get with a specific query. A non-native metric also can be a derived metric, where two or more native counters are used in calculations for ratios, hit rates, or latencies.

Counter	Type	Metric	Description	Definition
checkpoint_sync_time	Checkpoint	db.Checkpoint.checkpoint_sync_time	The total time spent in milliseconds that has been spent in the portion of checkpoint processing where files are synchronized to disk.	checkpoint_sync_time / (checkpoints_timed + checkpoints_req)
checkpoint_write_time	Checkpoint	db.Checkpoint.checkpoint_write_time	The total time spent in milliseconds that has been spent in the portion of checkpoint processing where files are written to disk.	checkpoint_write_time / (checkpoints_timed + checkpoints_req)
read_latency	I/O	db.IO.read_latency	The time spent reading data file blocks by backends in this instance.	blk_read_time / blks_read

OS metrics in Enhanced Monitoring

Amazon Aurora provides metrics in real time for the operating system (OS) that your DB cluster runs on. Aurora delivers the metrics from Enhanced Monitoring to your Amazon CloudWatch Logs account. The following tables list the OS metrics available using Amazon CloudWatch Logs.

Topics

- [OS metrics for Aurora \(p. 661\)](#)

OS metrics for Aurora

Group	Metric	Console name	Description
General	engine	Not applicable	The database engine for the DB instance.
	instanceID	Not applicable	The DB instance identifier.
	instanceResID	Not applicable	An immutable identifier for the DB instance that is unique to an AWS Region, also used as the log stream identifier.
	numVCpus	Not applicable	The number of virtual CPUs for the DB instance.
	timestamp	Not applicable	The time at which the metrics were taken.
	uptime	Not applicable	The amount of time that the DB instance has been active.
	version	Not applicable	The version of the OS metrics' stream JSON format.
cpuUtilization	guest	CPU Guest	The percentage of CPU in use by guest programs.
	idle	CPU Idle	The percentage of CPU that is idle.
	irq	CPU IRQ	The percentage of CPU in use by software interrupts.
	nice	CPU Nice	The percentage of CPU in use by programs running at lowest priority.
	steal	CPU Steal	The percentage of CPU in use by other virtual machines.
	system	CPU System	The percentage of CPU in use by the kernel.
	total	CPU Total	The total percentage of the CPU in use. This value includes the nice value.
	user	CPU User	The percentage of CPU in use by user programs.
	wait	CPU Wait	The percentage of CPU unused while waiting for I/O access.
diskIO	avgQueueLen	Avg Queue Size	The number of requests waiting in the I/O device's queue.
	avgReqSz	Ave Request Size	The average request size, in kilobytes.
	await	Disk I/O Await	The number of milliseconds required to respond to requests, including queue time and service time.
	device	Not applicable	The identifier of the disk device in use.
	readIOsPS	Read IO/s	The number of read operations per second.

Group	Metric	Console name	Description
aurora_db_cluster	readKb	Read Total	The total number of kilobytes read.
	readKbPS	Read Kb/s	The number of kilobytes read per second.
	readLatency	Read Latency	The elapsed time between the submission of a read I/O request and its completion, in milliseconds. This metric is only available for Amazon Aurora.
	readThroughput	Read Throughput	The amount of network throughput used by requests to the DB cluster, in bytes per second. This metric is only available for Amazon Aurora.
	rrqmPS	Rrqms	The number of merged read requests queued per second.
	tps	TPS	The number of I/O transactions per second.
	util	Disk I/O Util	The percentage of CPU time during which requests were issued.
	writeIOsPS	Write IO/s	The number of write operations per second.
	writeKb	Write Total	The total number of kilobytes written.
	writeKbPS	Write Kb/s	The number of kilobytes written per second.
	writeLatency	Write Latency	The average elapsed time between the submission of a write I/O request and its completion, in milliseconds. This metric is only available for Amazon Aurora.
	writeThroughput	Write Throughput	The amount of network throughput used by responses from the DB cluster, in bytes per second. This metric is only available for Amazon Aurora.
	wrqmPS	Wrqms	The number of merged write requests queued per second.
fileSys	maxFiles	Max Inodes	The maximum number of files that can be created for the file system.
	mountPoint	Not applicable	The path to the file system.
	name	Not applicable	The name of the file system.
	total	Total Filesystem	The total number of disk space available for the file system, in kilobytes.
	used	Used Filesystem	The amount of disk space used by files in the file system, in kilobytes.
	usedFilePercent	Used %	The percentage of available files in use.
	usedFiles	Used Inodes	The number of files in the file system.

Group	Metric	Console name	Description
	usedPercent	Used Inodes %	The percentage of the file-system disk space in use.
loadAverage	fifteen	Load Avg 15 min	The number of processes requesting CPU time over the last 15 minutes.
	five	Load Avg 5 min	The number of processes requesting CPU time over the last 5 minutes.
	one	Load Avg 1 min	The number of processes requesting CPU time over the last minute.
memory	active	Active Memory	The amount of assigned memory, in kilobytes.
	buffers	Buffered Memory	The amount of memory used for buffering I/O requests prior to writing to the storage device, in kilobytes.
	cached	Cached Memory	The amount of memory used for caching file system-based I/O.
	dirty	Dirty Memory	The amount of memory pages in RAM that have been modified but not written to their related data block in storage, in kilobytes.
	free	Free Memory	The amount of unassigned memory, in kilobytes.
	hugePagesFree	Huge Pages Free	The number of free huge pages. Huge pages are a feature of the Linux kernel.
	hugePagesRsvd	Huge Pages Rsvd	The number of committed huge pages.
	hugePagesSz	Huge Pages Size	The size for each huge pages unit, in kilobytes.
	hugePagesSurp	Huge Pages Surp	The number of available surplus huge pages over the total.
	hugePagesTotal	Huge Pages Total	The total number of huge pages.
	inactive	Inactive Memory	The amount of least-frequently used memory pages, in kilobytes.
	mapped	Mapped Memory	The total amount of file-system contents that is memory mapped inside a process address space, in kilobytes.
	pageTables	Page Tables	The amount of memory used by page tables, in kilobytes.
	slab	Slab Memory	The amount of reusable kernel data structures, in kilobytes.
	total	Total Memory	The total amount of memory, in kilobytes.

Group	Metric	Console name	Description
	writeback	Writeback Memory	The amount of dirty pages in RAM that are still being written to the backing storage, in kilobytes.
network	interface	Not applicable	The identifier for the network interface being used for the DB instance.
	rx	RX	The number of bytes received per second.
	tx	TX	The number of bytes uploaded per second.
processList	cpuUsedPc	CPU %	The percentage of CPU used by the process.
	id	Not applicable	The identifier of the process.
	memoryUsedPmem%	MEM%	The percentage of memory used by the process.
	name	Not applicable	The name of the process.
	parentID	Not applicable	The process identifier for the parent process of the process.
	rss	RES	The amount of RAM allocated to the process, in kilobytes.
	tgid	Not applicable	The thread group identifier, which is a number representing the process ID to which a thread belongs. This identifier is used to group threads from the same process.
	vss	VIRT	The amount of virtual memory allocated to the process, in kilobytes.
swap	swap	Swap	The amount of swap memory available, in kilobytes.
	swap_in	Swaps in	The amount of memory, in kilobytes, swapped in from disk.
	swap_out	Swaps out	The amount of memory, in kilobytes, swapped out to disk.
	free	Free Swap	The amount of swap memory free, in kilobytes.
	committed	Committed Swap	The amount of swap memory, in kilobytes, used as cache memory.
tasks	blocked	Tasks Blocked	The number of tasks that are blocked.
	running	Tasks Running	The number of tasks that are running.
	sleeping	Tasks Sleeping	The number of tasks that are sleeping.
	stopped	Tasks Stopped	The number of tasks that are stopped.

Group	Metric	Console name	Description
	total	Tasks Total	The total number of tasks.
	zombie	Tasks Zombie	The number of child tasks that are inactive with an active parent task.

Monitoring events, logs, and streams in an Amazon Aurora DB cluster

When you monitor your Amazon Aurora databases and your other AWS solutions, your goal is to maintain the following:

- Reliability
- Availability
- Performance

[Monitoring metrics in an Amazon Aurora cluster \(p. 541\)](#) explains how to monitor your cluster using metrics. A complete solution must also monitor database events, log files, and activity streams. AWS provides you with the following monitoring tools:

- *Amazon EventBridge* is a serverless event bus service that makes it easy to connect your applications with data from a variety of sources. EventBridge delivers a stream of real-time data from your own applications, Software-as-a-Service (SaaS) applications, and AWS services and routes that data to targets such as AWS Lambda. This enables you to monitor events that happen in services, and build event-driven architectures. For more information, see the [Amazon EventBridge User Guide](#).
- *Amazon CloudWatch Logs* lets you monitor, store, and access your log files from Amazon Aurora instances, AWS CloudTrail, and other sources. Amazon CloudWatch Logs can monitor information in the log files and notify you when certain thresholds are met. You can also archive your log data in highly durable storage. For more information, see the [Amazon CloudWatch Logs User Guide](#).
- *AWS CloudTrail* captures API calls and related events made by or on behalf of your AWS account and delivers the log files to an Amazon S3 bucket that you specify. You can identify which users and accounts called AWS, the source IP address from which the calls were made, and when the calls occurred. For more information, see the [AWS CloudTrail User Guide](#).
- *Database Activity Streams* is an Amazon Aurora feature that provides a near-real-time stream of the activity in your DB cluster. Amazon Aurora pushes activities to an Amazon Kinesis data stream. The Kinesis stream is created automatically. From Kinesis, you can configure AWS services such as Amazon Kinesis Data Firehose and AWS Lambda to consume the stream and store the data.

Topics

- [Viewing logs, events, and streams in the Amazon RDS console \(p. 666\)](#)
- [Monitoring Amazon Aurora events \(p. 671\)](#)
- [Monitoring Amazon Aurora log files \(p. 695\)](#)
- [Monitoring Amazon Aurora API calls in AWS CloudTrail \(p. 710\)](#)
- [Monitoring Amazon Aurora with Database Activity Streams \(p. 714\)](#)

Viewing logs, events, and streams in the Amazon RDS console

Amazon RDS integrates with AWS services to show information about logs, events, and database activity streams in the RDS console.

The **Logs & events** tab for your Aurora DB cluster shows the following information:

- **Auto scaling policies and activities** – Shows policies and activities relating to the Aurora Auto Scaling feature. This information only appears in the **Logs & events** tab at the cluster level.
- **Amazon CloudWatch alarms** – Shows any metric alarms that you have configured for the DB instance in your Aurora cluster. If you haven't configured alarms, you can create them in the RDS console.
- **Recent events** – Shows a summary of events (environment changes) for your Aurora DB instance or cluster. For more information, see [Viewing Amazon RDS events \(p. 674\)](#).
- **Logs** – Shows database log files generated by a DB instance in your Aurora cluster. For more information, see [Monitoring Amazon Aurora log files \(p. 695\)](#).

The **Configuration** tab displays information about database activity streams.

To view logs, events, and streams for your Aurora DB cluster in the RDS console

1. Sign in to the AWS Management Console and open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. In the navigation pane, choose **Databases**.
3. Choose the name of the Aurora DB cluster that you want to monitor.

The database page appears. The following example shows an Amazon Aurora PostgreSQL DB cluster named apga.

The screenshot shows the Amazon RDS "Databases" page for the "apg" cluster. The cluster name "apg" is displayed prominently at the top. Below it, there is a "Related" section with a search bar labeled "Filter by databases". A table lists the cluster's components:

DB identifier	DB cluster identifier	Role	Engine
apg	apg	Regional cluster	Aurora PostgreSQL
apg-instance-1-us-east-1c	apg	Writer instance	Aurora PostgreSQL
apg-instance-1	apg	Reader instance	Aurora PostgreSQL
apg-instance-2	apg	Reader instance	Aurora PostgreSQL

At the bottom of the page, there are tabs for "Connectivity & security", "Monitoring", "Logs & events" (which is highlighted in orange), "Configuration", "Maintenance & backups", and "Tags".

4. Scroll down and choose **Configuration**.

The following example shows the status of the database activity streams for your cluster.

Configuration	Maintenance & backups	Tags
Availability	Encryption	
IAM DB authentication	Encryption	
Not enabled	Enabled	
Master username	AWS KMS key	
apga_admin	aws/rds 	
Master password	Database activity stream	
*****	Status	
Multi-AZ	Stopped	
3 Zones	Published logs	
	CloudWatch Logs	
	PostgreSQL	

5. Choose **Logs & events**.

The Logs & events section appears.

The screenshot shows the 'Logs & events' tab selected in the navigation bar. It displays three main sections:

- Auto scaling policies (0)**: A table with columns for Name, Scaling action, Target metric, and Target value. It includes a search bar, pagination, and buttons for Edit, Delete, and Add.
- Auto scaling activities (0)**: A table with columns for Start time, End time, Status, Description, and Status message. It includes a search bar, pagination, and a button for Add auto scaling policy.
- Recent events (3)**: A table with columns for Time and System notes. It includes a search bar, pagination, and a button for Filter by db events.

6. Choose a DB instance in your Aurora cluster, and then choose **Logs & events** for the instance.

The following example shows that the contents are different between the DB instance page and the DB cluster page. The DB instance page shows logs and alarms.

Amazon Aurora User Guide for Aurora
Viewing logs, events, and streams
in the Amazon RDS console

The screenshot shows the 'Logs & events' tab selected in the navigation bar. The interface is divided into three main sections: 'CloudWatch alarms', 'Recent events', and 'Logs'.

CloudWatch alarms (0): This section includes a search bar, a 'Create alarm' button, and a table header for 'Name' and 'State'. Below the table, it says 'Empty alarms table' and has a 'Create alarm' button.

Recent events (0): This section includes a search bar, a 'Create' button, and a table header for 'Time' and 'System notes'. Below the table, it says 'No events found.'

Logs (29): This section includes a search bar, buttons for 'View', 'Watch', and 'Download', and a table header for 'Name' and 'Last written'. The table lists three log files:

Name	Last written	Logs
error/postgres.log	Thu Feb 03 2022 12:18:27 GMT-0500	29.1 kB
error/postgresql.log.2022-02-03-1709	Thu Feb 03 2022 12:09:59 GMT-0500	4.3 kB
error/postgresql.log.2022-02-03-1710	Thu Feb 03 2022 12:10:58 GMT-0500	5.4 kB

Monitoring Amazon Aurora events

An *event* indicates a change in an environment. This can be an AWS environment, an SaaS partner service or application, or a custom application or service.

Topics

- [Overview of events for Aurora \(p. 671\)](#)
- [Viewing Amazon RDS events \(p. 674\)](#)
- [Using Amazon RDS event notification \(p. 675\)](#)
- [Creating a rule that triggers on an Amazon Aurora event \(p. 692\)](#)

Overview of events for Aurora

An *RDS event* indicates a change in the Aurora environment. For example, Amazon Aurora generates an event when a DB cluster is patched. Amazon Aurora delivers events to CloudWatch Events and EventBridge in near-real time.

Note

Amazon RDS emits events on a best effort basis. We recommend that you avoid writing programs that depend on the order or existence of notification events, because they might be out of sequence or missing.

Amazon RDS records events that relate to your DB clusters, DB instances, DB cluster snapshots, and DB parameter groups. This information includes the following:

- The date and time of the event
- The source name and source type of the event
- A message associated with the event.

The following examples illustrate different types of Aurora events.

Topics

- [Example of a DB cluster event \(p. 671\)](#)
- [Example of a DB instance event \(p. 672\)](#)
- [Example of a DB parameter group event \(p. 672\)](#)
- [Example of a DB cluster snapshot event \(p. 673\)](#)

Example of a DB cluster event

The following is an example of a DB cluster event in JSON format. The event shows that the cluster named `my-db-cluster` was patched. The event ID is `RDS-EVENT-0173`.

```
{  
    "version": "0",  
    "id": "844e2571-85d4-695f-b930-0153b71dcb42",  
    "detail-type": "RDS DB Cluster Event",  
    "source": "aws.rds",  
    "account": "123456789012",  
    "time": "2018-10-06T12:26:13Z",  
    "region": "us-east-1",  
    "resources": [  
        "arn:aws:rds:us-east-1:123456789012:cluster:my-db-cluster"  
    ],  
}
```

```

    "detail": {
        "EventCategories": [
            "notification"
        ],
        "SourceType": "CLUSTER",
        "SourceArn": "arn:aws:rds:us-east-1:123456789012:cluster:my-db-cluster",
        "Date": "2018-10-06T12:26:13.882Z",
        "Message": "Database cluster has been patched",
        "SourceIdentifier": "rds:my-db-cluster",
        "EventID": "RDS-EVENT-0173"
    }
}

```

Example of a DB instance event

The following is an example of a DB instance event in JSON format. The event shows that RDS performed a multi-AZ failover for the instance named `my-db-instance`. The event ID is `RDS-EVENT-0049`.

```

{
    "version": "0",
    "id": "68f6e973-1a0c-d37b-f2f2-94a7f62ffd4e",
    "detail-type": "RDS DB Instance Event",
    "source": "aws.rds",
    "account": "123456789012",
    "time": "2018-09-27T22:36:43Z",
    "region": "us-east-1",
    "resources": [
        "arn:aws:rds:us-east-1:123456789012:db:my-db-instance"
    ],
    "detail": {
        "EventCategories": [
            "failover"
        ],
        "SourceType": "DB_INSTANCE",
        "SourceArn": "arn:aws:rds:us-east-1:123456789012:db:my-db-instance",
        "Date": "2018-09-27T22:36:43.292Z",
        "Message": "A Multi-AZ failover has completed.",
        "SourceIdentifier": "rds:my-db-instance",
        "EventID": "RDS-EVENT-0049"
    }
}

```

Example of a DB parameter group event

The following is an example of a DB parameter group event in JSON format. The event shows that the parameter `time_zone` was updated in parameter group `my-db-param-group`. The event ID is `RDS-EVENT-0037`.

```

{
    "version": "0",
    "id": "844e2571-85d4-695f-b930-0153b71dcba2",
    "detail-type": "RDS DB Parameter Group Event",
    "source": "aws.rds",
    "account": "123456789012",
    "time": "2018-10-06T12:26:13Z",
    "region": "us-east-1",
    "resources": [
        "arn:aws:rds:us-east-1:123456789012:pg:my-db-param-group"
    ],
    "detail": {

```

```
    "EventCategories": [
        "configuration change"
    ],
    "SourceType": "DB_PARAM",
    "SourceArn": "arn:aws:rds:us-east-1:123456789012:pg:my-db-param-group",
    "Date": "2018-10-06T12:26:13.882Z",
    "Message": "Updated parameter time_zone to UTC with apply method immediate",
    "SourceIdentifier": "rds:my-db-param-group",
    "EventID": "RDS-EVENT-0037"
}
}
```

Example of a DB cluster snapshot event

The following is an example of a DB cluster snapshot event in JSON format. The event shows the creation of the snapshot named my-db-cluster-snapshot. The event ID is RDS-EVENT-0074.

```
{
    "version": "0",
    "id": "844e2571-85d4-695f-b930-0153b71dcba2",
    "detail-type": "RDS DB Cluster Snapshot Event",
    "source": "aws.rds",
    "account": "123456789012",
    "time": "2018-10-06T12:26:13Z",
    "region": "us-east-1",
    "resources": [
        "arn:aws:rds:us-east-1:123456789012:cluster-snapshot:rds:my-db-cluster-snapshot"
    ],
    "detail": {
        "EventCategories": [
            "backup"
        ],
        "SourceType": "CLUSTER_SNAPSHOT",
        "SourceArn": "arn:aws:rds:us-east-1:123456789012:cluster-snapshot:rds:my-db-cluster-snapshot",
        "Date": "2018-10-06T12:26:13.882Z",
        "SourceIdentifier": "rds:my-db-cluster-snapshot",
        "Message": "Creating manual cluster snapshot",
        "EventID": "RDS-EVENT-0074"
    }
}
```

Viewing Amazon RDS events

You can retrieve events for your RDS resources through the AWS Management Console, which shows events from the past 24 hours. You can also retrieve events for your RDS resources by using the [describe-events](#) AWS CLI command, or the [DescribeEvents](#) RDS API operation. If you use the AWS CLI or the RDS API to view events, you can retrieve events for up to the past 14 days.

Note

If you need to store events for longer periods of time, you can send Amazon RDS events to CloudWatch Events. For more information, see [Creating a rule that triggers on an Amazon Aurora event \(p. 692\)](#)

Console

To view all Amazon RDS instance events for the past 24 hours

1. Sign in to the AWS Management Console and open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. In the navigation pane, choose **Events**. The available events appear in a list.
3. Use the **Filter** list to filter the events by type, and use the text box to the right of the **Filter** list to further filter your results. For example, the following screenshot shows a list of events filtered by the characters **error**.

Name	Last written	Logs
error/mysql-error-running.log	Fri Dec 10 2021 12:30:00 GMT-0500	72.3 kB
error/mysql-error.log	Fri Dec 10 2021 12:36:40 GMT-0500	14.9 kB

AWS CLI

You can view all Amazon RDS instance events for the past 7 days by calling the [describe-events](#) AWS CLI command and setting the `--duration` parameter to 10080.

```
aws rds describe-events --duration 10080
```

API

You can view all Amazon RDS instance events for the past 14 days by calling the [DescribeEvents](#) RDS API operation and setting the `Duration` parameter to 20160.

Using Amazon RDS event notification

Amazon RDS uses the Amazon Simple Notification Service (Amazon SNS) to provide notification when an Amazon RDS event occurs. These notifications can be in any notification form supported by Amazon SNS for an AWS Region, such as an email, a text message, or a call to an HTTP endpoint.

Topics

- [Overview of Amazon RDS event notification \(p. 675\)](#)
- [Amazon RDS event categories and event messages \(p. 676\)](#)
- [Subscribing to Amazon RDS event notification \(p. 683\)](#)
- [Listing Amazon RDS event notification subscriptions \(p. 686\)](#)
- [Modifying an Amazon RDS event notification subscription \(p. 687\)](#)
- [Adding a source identifier to an Amazon RDS event notification subscription \(p. 688\)](#)
- [Removing a source identifier from an Amazon RDS event notification subscription \(p. 689\)](#)
- [Listing the Amazon RDS event notification categories \(p. 690\)](#)
- [Deleting an Amazon RDS event notification subscription \(p. 691\)](#)

Overview of Amazon RDS event notification

Amazon RDS groups events into categories that you can subscribe to so that you can be notified when an event in that category occurs. Amazon RDS event notification is only available for unencrypted SNS topics. If you specify an encrypted SNS topic, event notifications aren't sent for the topic.

RDS resources eligible for event subscription

For Amazon Aurora, events occur at both the DB cluster and the DB instance level. You can subscribe to an event category for the following resources:

- DB instance
- DB cluster
- DB cluster snapshot
- DB parameter group
- DB security group
- RDS Proxy

For example, if you subscribe to the backup category for a given DB instance, you're notified whenever a backup-related event occurs that affects the DB instance. If you subscribe to a configuration change category for a DB security group, you're notified when the DB security group is changed. You also receive notification when an event notification subscription changes.

You might want to create several different subscriptions. For example, you might create one subscription receiving all event notifications and another subscription that includes only critical events for your production DB instances.

Basic process for subscribing to Amazon RDS event notifications

The process for subscribing to Amazon RDS event notification is as follows:

1. You create an Amazon RDS event notification subscription by using the Amazon RDS console, AWS CLI, or API.

Amazon RDS uses the ARN of an Amazon SNS topic to identify each subscription. The Amazon RDS console creates the ARN for you when you create the subscription. Create the ARN by using the Amazon SNS console, the AWS CLI, or the Amazon SNS API.

2. Amazon RDS sends an approval email or SMS message to the addresses you submitted with your subscription. To confirm your subscription, choose the link in the notification you were sent.
3. When you have confirmed the subscription, the status of your subscription is updated in the Amazon RDS console's **My Event Subscriptions** section.
4. You then begin to receive event notifications.

To learn about identity and access management when using Amazon SNS, see [Identity and access management in Amazon SNS](#) in the *Amazon Simple Notification Service Developer Guide*.

You can use AWS Lambda to process event notifications from a DB instance. For more information, see [Using AWS Lambda with Amazon RDS](#) in the *AWS Lambda Developer Guide*.

Delivery of RDS event notifications

Amazon RDS sends notifications to the addresses that you provide when you create the subscription. Event notifications might take up to five minutes to be delivered.

Important

Amazon RDS doesn't guarantee the order of events sent in an event stream. The event order is subject to change.

When Amazon SNS sends a notification to a subscribed HTTP or HTTPS endpoint, the POST message sent to the endpoint has a message body that contains a JSON document. For more information, see [Amazon SNS message and JSON formats](#) in the *Amazon Simple Notification Service Developer Guide*.

You can configure SNS to notify you with text messages. For more information, see [Mobile text messaging \(SMS\)](#) in the *Amazon Simple Notification Service Developer Guide*.

To turn off notifications without deleting a subscription, choose **No** for **Enabled** in the Amazon RDS console. Or you can set the `Enabled` parameter to `false` using the AWS CLI or Amazon RDS API.

Billing for Amazon RDS event notifications

Billing for Amazon RDS event notification is through Amazon SNS. Amazon SNS fees apply when using event notification. For more information about Amazon SNS billing, see [Amazon Simple Notification Service pricing](#).

Amazon RDS event categories and event messages

Amazon RDS generates a significant number of events in categories that you can subscribe to using the Amazon RDS Console, AWS CLI, or the API. Each category applies to a source type.

Topics

- [DB instance events \(p. 677\)](#)
- [DB parameter group events \(p. 679\)](#)
- [DB security group events \(p. 680\)](#)
- [DB cluster events \(p. 680\)](#)
- [DB cluster snapshot events \(p. 682\)](#)
- [RDS Proxy events \(p. 682\)](#)

DB instance events

The following table shows the event category and a list of events when a DB instance is the source type.

Category	Amazon RDS event ID	Description
Availability	RDS-EVENT-0006	The DB instance restarted.
Availability	RDS-EVENT-0004	DB instance shutdown.
Availability	RDS-EVENT-0022	An error has occurred while restarting Aurora MySQL or MariaDB.
Backtrack	RDS-EVENT-0131	The actual Backtrack window is smaller than the target Backtrack window you specified. Consider reducing the number of hours in your target Backtrack window. For more information about backtracking, see Backtracking an Aurora DB cluster (p. 816) .
Backtrack	RDS-EVENT-0132	The actual Backtrack window is the same as the target Backtrack window.
Configuration change	RDS-EVENT-0009	The DB instance has been added to a security group.
Configuration change	RDS-EVENT-0012	Applying modification to database instance class.
Configuration change	RDS-EVENT-0011	A parameter group for this DB instance has changed.
Configuration change	RDS-EVENT-0092	A parameter group for this DB instance has finished updating.
Configuration change	RDS-EVENT-0033	There are [count] users that match the master user name. Users not tied to a specific host have been reset.
Configuration change	RDS-EVENT-0025	The DB instance has been converted to a Multi-AZ DB instance.
Configuration change	RDS-EVENT-0029	The DB instance has been converted to a Single-AZ DB instance.
Configuration change	RDS-EVENT-0014	The DB instance class for this DB instance has changed.
Configuration change	RDS-EVENT-0017	The storage settings for this DB instance have changed.
Configuration change	RDS-EVENT-0010	The DB instance has been removed from a security group.
Configuration change	RDS-EVENT-0016	The master password for the DB instance has been reset.
Configuration change	RDS-EVENT-0067	An attempt to reset the master password for the DB instance has failed.

Category	Amazon RDS event ID	Description
Configuration change	RDS-EVENT-0078	The Enhanced Monitoring configuration has been changed.
Creation	RDS-EVENT-0005	DB instance created.
Deletion	RDS-EVENT-0003	The DB instance has been deleted.
Failure	RDS-EVENT-0035	The DB instance has invalid parameters. For example, if the DB instance could not start because a memory-related parameter is set too high for this instance class, the customer action would be to modify the memory parameter and reboot the DB instance.
Failure	RDS-EVENT-0036	The DB instance is in an incompatible network. Some of the specified subnet IDs are invalid or do not exist.
Failure	RDS-EVENT-0079	Enhanced Monitoring cannot be enabled without the enhanced monitoring IAM role. For information on creating the enhanced monitoring IAM role, see To create an IAM role for Amazon RDS enhanced monitoring (p. 628) .
Failure	RDS-EVENT-0080	Enhanced Monitoring was disabled due to an error making the configuration change. It is likely that the enhanced monitoring IAM role is configured incorrectly. For information on creating the enhanced monitoring IAM role, see To create an IAM role for Amazon RDS enhanced monitoring (p. 628) .
Failure	RDS-EVENT-0082	Aurora was unable to copy backup data from an Amazon S3 bucket. It is likely that the permissions for Aurora to access the Amazon S3 bucket are configured incorrectly. For more information, see Migrating data from MySQL by using an Amazon S3 bucket (p. 784) .
Low storage	RDS-EVENT-0007	The allocated storage for the DB instance has been consumed. To resolve this issue, allocate additional storage for the DB instance. For more information, see the RDS FAQ . You can monitor the storage space for a DB instance using the Free Storage Space metric.
Low storage	RDS-EVENT-0089	The DB instance has consumed more than 90% of its allocated storage. You can monitor the storage space for a DB instance using the Free Storage Space metric.
Maintenance	RDS-EVENT-0026	Offline maintenance of the DB instance is taking place. The DB instance is currently unavailable.
Maintenance	RDS-EVENT-0027	Offline maintenance of the DB instance is complete. The DB instance is now available.
Maintenance	RDS-EVENT-0047	Patching of the DB instance has completed.

Category	Amazon RDS event ID	Description
Maintenance	RDS-EVENT-0155	The DB instance has a DB engine minor version upgrade available.
Notification	RDS-EVENT-0044	Operator-issued notification. For more information, see the event message.
Notification	RDS-EVENT-0048	Patching of the DB instance has been delayed.
Read replica	RDS-EVENT-0045	An error has occurred in the read replication process. For more information, see the event message. For information on troubleshooting read replica errors, see Troubleshooting a MySQL read replica problem .
Read replica	RDS-EVENT-0046	The read replica has resumed replication. This message appears when you first create a read replica, or as a monitoring message confirming that replication is functioning properly. If this message follows an RDS-EVENT-0045 notification, then replication has resumed following an error or after replication was stopped.
Read replica	RDS-EVENT-0057	Replication on the read replica was terminated.
Recovery	RDS-EVENT-0020	Recovery of the DB instance has started. Recovery time will vary with the amount of data to be recovered.
Recovery	RDS-EVENT-0021	Recovery of the DB instance is complete.
Recovery	RDS-EVENT-0023	A manual backup has been requested but Amazon RDS is currently in the process of creating a DB snapshot. Submit the request again after Amazon RDS has completed the DB snapshot.
Recovery	RDS-EVENT-0052	Recovery of the Multi-AZ instance has started. Recovery time will vary with the amount of data to be recovered.
Recovery	RDS-EVENT-0053	Recovery of the Multi-AZ instance is complete.
Restoration	RDS-EVENT-0019	The DB instance has been restored from a point-in-time backup.
Restoration	RDS-EVENT-0043	Restored from snapshot [snapshot_name]. The DB instance has been restored from a DB snapshot.

DB parameter group events

The following table shows the event category and a list of events when a DB parameter group is the source type.

Category	RDS event ID	Description
Configuration change	RDS-EVENT-0037	The parameter group was modified.

DB security group events

The following table shows the event category and a list of events when a DB security group is the source type.

Category	RDS event ID	Description
Configuration change	RDS-EVENT-0038	The security group has been modified.
Failure	RDS-EVENT-0039	The security group owned by [user] does not exist; authorization for the security group has been revoked.

DB cluster events

The following table shows the event category and a list of events when an Aurora DB cluster is the source type.

Note

No event category exists for Aurora Serverless in the DB cluster event type. The Aurora Serverless events range from RDS-EVENT-0141 to RDS-EVENT-0149.

Category	RDS event ID	Description
Configuration change	RDS-EVENT-0179	Database Activity Streams is started on your database cluster. For more information see Monitoring Amazon Aurora with Database Activity Streams (p. 714) .
Configuration change	RDS-EVENT-0180	Database Activity Streams is stopped on your database cluster. For more information see Monitoring Amazon Aurora with Database Activity Streams (p. 714) .
creation	RDS-EVENT-0170	DB cluster created.
deletion	RDS-EVENT-0171	DB cluster deleted.
Failover	RDS-EVENT-0069	A failover for the DB cluster has failed.
Failover	RDS-EVENT-0070	A failover for the DB cluster has restarted.
Failover	RDS-EVENT-0071	A failover for the DB cluster has finished.
Failover	RDS-EVENT-0072	A failover for the DB cluster has begun within the same Availability Zone.
Failover	RDS-EVENT-0073	A failover for the DB cluster has begun across Availability Zones.

Category	RDS event ID	Description
Failure	RDS-EVENT-0083	Aurora was unable to copy backup data from an Amazon S3 bucket. It is likely that the permissions for Aurora to access the Amazon S3 bucket are configured incorrectly. For more information, see Migrating data from MySQL by using an Amazon S3 bucket (p. 784) .
Failure	RDS-EVENT-0143	Scaling failed for the Aurora Serverless DB cluster.
Global failover	RDS-EVENT-0181	The failover of the global database has started. The process can be delayed because other operations are running on the DB cluster.
Global failover	RDS-EVENT-0182	The old primary instance in the global database isn't accepting writes. All volumes are synchronized.
Global failover	RDS-EVENT-0183	A replication lag is occurring during the synchronization phase of the global database failover.
Global failover	RDS-EVENT-0184	The volume topology of the global database is reestablished with the new primary volume.
Global failover	RDS-EVENT-0185	The global database failover is finished on the primary DB cluster. Replicas might take long to come online after the failover completes.
Global failover	RDS-EVENT-0186	The global database failover is canceled.
Global failover	RDS-EVENT-0187	The global failover to the DB cluster failed.
Maintenance	RDS-EVENT-0156	The DB cluster has a DB engine minor version upgrade available.
Notification	RDS-EVENT-0076	Migration to an Aurora DB cluster failed.
Notification	RDS-EVENT-0077	An attempt to convert a table from the source database to InnoDB failed during the migration to an Aurora DB cluster.
Notification	RDS-EVENT-0085	An error occurred while attempting to patch the Aurora DB cluster. Check your instance status, resolve the issue, and try again. For more information see Maintaining an Amazon Aurora DB cluster (p. 443) .
Notification	RDS-EVENT-0141	Scaling initiated for the Aurora Serverless DB cluster.
Notification	RDS-EVENT-0142	Scaling completed for the Aurora Serverless DB cluster.
Notification	RDS-EVENT-0144	Automatic pause initiated for the Aurora Serverless DB cluster.
Notification	RDS-EVENT-0145	The Aurora Serverless DB cluster paused.
Notification	RDS-EVENT-0146	Pause cancelled for the Aurora Serverless DB cluster.
Notification	RDS-EVENT-0147	Resume initiated for the Aurora Serverless DB cluster.

Category	RDS event ID	Description
Notification	RDS-EVENT-0148	Resume completed for the Aurora Serverless DB cluster.
Notification	RDS-EVENT-0149	Seamless scaling completed with the force option for the Aurora Serverless DB cluster. Connections might have been interrupted as required.
Notification	RDS-EVENT-0150	The DB cluster stopped.
Notification	RDS-EVENT-0151	The DB cluster started.
Notification	RDS-EVENT-0152	The DB cluster stop failed.
Notification	RDS-EVENT-0153	The DB cluster is being started due to it exceeding the maximum allowed time being stopped.
Notification	RDS-EVENT-0173	Patching of the DB cluster has completed.

DB cluster snapshot events

The following table shows the event category and a list of events when an Aurora DB cluster snapshot is the source type.

Category	RDS event ID	Description
Backup	RDS-EVENT-0074	Creation of a manual DB cluster snapshot has started.
Backup	RDS-EVENT-0075	A manual DB cluster snapshot has been created.
notification	RDS-EVENT-0162	DB cluster snapshot export task failed.
notification	RDS-EVENT-0163	DB cluster snapshot export task canceled.
notification	RDS-EVENT-0164	DB cluster snapshot export task completed.
backup	RDS-EVENT-0168	Creating automated cluster snapshot.
backup	RDS-EVENT-0169	Automated cluster snapshot created.
notification	RDS-EVENT-0172	Renamed DB cluster from [old DB cluster name] to [new DB cluster name].

RDS Proxy events

The following table shows the event category and a list of events when an RDS Proxy is the source type.

Category	RDS event ID	Description
Configuration change	RDS-EVENT-0204	RDS modified the DB proxy (RDS Proxy).
Configuration change	RDS-EVENT-0207	RDS modified the endpoint of the DB proxy (RDS Proxy).

Category	RDS event ID	Description
Configuration change	RDS-EVENT-0213	RDS detected the addition of the DB instance and automatically added it to the target group of the DB proxy (RDS Proxy).
Configuration change	RDS-EVENT-0214	RDS detected the deletion of the DB instance and automatically removed it from the target group of the DB proxy (RDS Proxy).
Configuration change	RDS-EVENT-XXXX	RDS detected the deletion of the DB cluster and automatically removed it from the target group of the DB proxy (RDS Proxy).
Creation	RDS-EVENT-0203	RDS created the DB proxy (RDS Proxy).
Creation	RDS-EVENT-0206	RDS created the endpoint for the DB proxy (RDS Proxy).
Deletion	RDS-EVENT-0205	RDS deleted the DB proxy (RDS Proxy).
Deletion	RDS-EVENT-0208	RDS deleted the endpoint of DB proxy (RDS Proxy).

Subscribing to Amazon RDS event notification

The simplest way to create a subscription is with the RDS console. If you choose to create event notification subscriptions using the CLI or API, you must create an Amazon Simple Notification Service topic and subscribe to that topic with the Amazon SNS console or Amazon SNS API. You will also need to retain the Amazon Resource Name (ARN) of the topic because it is used when submitting CLI commands or API operations. For information on creating an SNS topic and subscribing to it, see [Getting started with Amazon SNS](#) in the *Amazon Simple Notification Service Developer Guide*.

You can specify the type of source you want to be notified of and the Amazon RDS source that triggers the event. These are defined by the **SourceType** (type of source) and the **SourcelIdentifier** (the Amazon RDS source generating the event). For example, **SourceType** might be `SourceType = db-instance`, whereas **SourcelIdentifier** might be `SourceIdentifier = myDBInstance1`. The following table shows possible combinations.

SourceType	SourcelIdentifier	Description
Specified	Specified	You receive notice of all DB instance events for the specified source.
Specified	Not specified	You receive notice of the events for that source type for all your Amazon RDS sources.
Not specified	Not specified	You receive notice of all events from all Amazon RDS sources belonging to your customer account.

Console

To subscribe to RDS event notification

1. Sign in to the AWS Management Console and open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.

2. In navigation pane, choose **Event subscriptions**.
3. In the **Event subscriptions** pane, choose **Create event subscription**.
4. In the **Create event subscription** dialog box, do the following:
 - a. For **Name**, enter a name for the event notification subscription.
 - b. For **Send notifications to**, choose an existing Amazon SNS ARN for an Amazon SNS topic, or choose **create topic** to enter the name of a topic and a list of recipients.
 - c. For **Source type**, choose a source type.
 - d. Choose **Yes** to enable the subscription. If you want to create the subscription but to not have notifications sent yet, choose **No**.
 - e. Depending on the source type you selected, choose the event categories and sources that you want to receive event notifications for.
 - f. Choose **Create**.

The Amazon RDS console indicates that the subscription is being created.

Name	Status	Source Type	Enabled
Configchangerdpgres	active	Instances	Yes
Test	creating	Instances	Yes

AWS CLI

To subscribe to RDS event notification, use the AWS CLI [create-event-subscription](#) command. Include the following required parameters:

- `--subscription-name`
- `--sns-topic-arn`

Example

For Linux, macOS, or Unix:

```
aws rds create-event-subscription \
--subscription-name myeventsubscription \
--sns-topic-arn arn:aws:sns:us-east-1:802#####:myawsuser-RDS \
--enabled
```

For Windows:

```
aws rds create-event-subscription ^
--subscription-name myeventsubscription ^
--sns-topic-arn arn:aws:sns:us-east-1:802#####:myawsuser-RDS ^
--enabled
```

API

To subscribe to Amazon RDS event notification, call the Amazon RDS API function [CreateEventSubscription](#). Include the following required parameters:

- `SubscriptionName`

- SnsTopicArn

Listings Amazon RDS event notification subscriptions

You can list your current Amazon RDS event notification subscriptions.

Console

To list your current Amazon RDS event notification subscriptions

1. Sign in to the AWS Management Console and open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. In the navigation pane, choose **Event subscriptions**. The **Event subscriptions** pane shows all your event notification subscriptions.

Event subscriptions (2)		
	Name	Status
<input type="checkbox"/> Edit Delete Create event subscription		
<input type="checkbox"/>	Configchangerdpgres	 active
<input type="checkbox"/>	Postgresnotification	 active

AWS CLI

To list your current Amazon RDS event notification subscriptions, use the AWS CLI [describe-event-subscriptions](#) command.

Example

The following example describes all event subscriptions.

```
aws rds describe-event-subscriptions
```

The following example describes the *myfirsteventsubscription*.

```
aws rds describe-event-subscriptions --subscription-name myfirsteventsubscription
```

API

To list your current Amazon RDS event notification subscriptions, call the Amazon RDS API [DescribeEventSubscriptions](#) action.

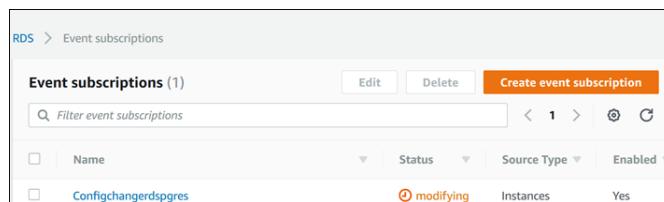
Modifying an Amazon RDS event notification subscription

After you have created a subscription, you can change the subscription name, source identifier, categories, or topic ARN.

Console

To modify an Amazon RDS event notification subscription

1. Sign in to the AWS Management Console and open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. In the navigation pane, choose **Event subscriptions**.
3. In the **Event subscriptions** pane, choose the subscription that you want to modify and choose **Edit**.
4. Make your changes to the subscription in either the **Target** or **Source** section.
5. Choose **Edit**. The Amazon RDS console indicates that the subscription is being modified.



AWS CLI

To modify an Amazon RDS event notification subscription, use the AWS CLI `modify-event-subscription` command. Include the following required parameter:

- `--subscription-name`

Example

The following code enables `myeventsSubscription`.

For Linux, macOS, or Unix:

```
aws rds modify-event-subscription \
--subscription-name myeventsSubscription \
--enabled
```

For Windows:

```
aws rds modify-event-subscription ^
--subscription-name myeventsSubscription ^
--enabled
```

API

To modify an Amazon RDS event, call the Amazon RDS API operation `ModifyEventSubscription`. Include the following required parameter:

- `SubscriptionName`

Adding a source identifier to an Amazon RDS event notification subscription

You can add a source identifier (the Amazon RDS source generating the event) to an existing subscription.

Console

You can easily add or remove source identifiers using the Amazon RDS console by selecting or deselecting them when modifying a subscription. For more information, see [Modifying an Amazon RDS event notification subscription \(p. 687\)](#).

AWS CLI

To add a source identifier to an Amazon RDS event notification subscription, use the AWS CLI [add-source-identifier-to-subscription](#) command. Include the following required parameters:

- `--subscription-name`
- `--source-identifier`

Example

The following example adds the source identifier `mysqldb` to the `myrdseventsSubscription` subscription.

For Linux, macOS, or Unix:

```
aws rds add-source-identifier-to-subscription \
--subscription-name myrdseventsSubscription \
--source-identifier mysqldb
```

For Windows:

```
aws rds add-source-identifier-to-subscription ^
--subscription-name myrdseventsSubscription ^
--source-identifier mysqldb
```

API

To add a source identifier to an Amazon RDS event notification subscription, call the Amazon RDS API [AddSourceIdentifierToSubscription](#). Include the following required parameters:

- `SubscriptionName`
- `SourceIdentifier`

Removing a source identifier from an Amazon RDS event notification subscription

You can remove a source identifier (the Amazon RDS source generating the event) from a subscription if you no longer want to be notified of events for that source.

Console

You can easily add or remove source identifiers using the Amazon RDS console by selecting or deselecting them when modifying a subscription. For more information, see [Modifying an Amazon RDS event notification subscription \(p. 687\)](#).

AWS CLI

To remove a source identifier from an Amazon RDS event notification subscription, use the AWS CLI `remove-source-identifier-from-subscription` command. Include the following required parameters:

- `--subscription-name`
- `--source-identifier`

Example

The following example removes the source identifier `mysqldb` from the `myrdseventsSubscription` subscription.

For Linux, macOS, or Unix:

```
aws rds remove-source-identifier-from-subscription \
--subscription-name myrdseventsSubscription \
--source-identifier mysqldb
```

For Windows:

```
aws rds remove-source-identifier-from-subscription ^
--subscription-name myrdseventsSubscription ^
--source-identifier mysqldb
```

API

To remove a source identifier from an Amazon RDS event notification subscription, use the Amazon RDS API `RemoveSourceIdentifierFromSubscription` command. Include the following required parameters:

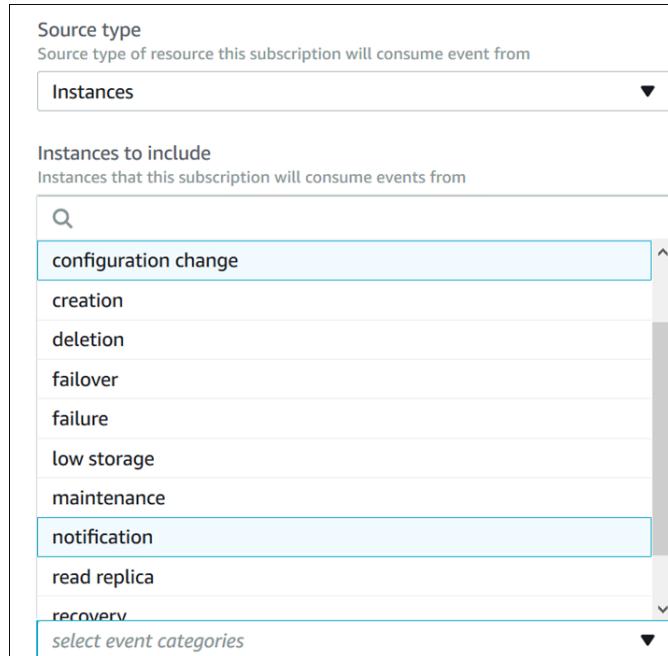
- `SubscriptionName`
- `SourceIdentifier`

Listing the Amazon RDS event notification categories

All events for a resource type are grouped into categories. To view the list of categories available, use the following procedures.

Console

When you create or modify an event notification subscription, the event categories are displayed in the Amazon RDS console. For more information, see [Modifying an Amazon RDS event notification subscription \(p. 687\)](#).



AWS CLI

To list the Amazon RDS event notification categories, use the AWS CLI [describe-event-categories](#) command. This command has no required parameters.

Example

```
aws rds describe-event-categories
```

API

To list the Amazon RDS event notification categories, use the Amazon RDS API [DescribeEventCategories](#) command. This command has no required parameters.

Deleting an Amazon RDS event notification subscription

You can delete a subscription when you no longer need it. All subscribers to the topic will no longer receive event notifications specified by the subscription.

Console

To delete an Amazon RDS event notification subscription

1. Sign in to the AWS Management Console and open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. In the navigation pane, choose **DB Event Subscriptions**.
3. In the **My DB Event Subscriptions** pane, choose the subscription that you want to delete.
4. Choose **Delete**.
5. The Amazon RDS console indicates that the subscription is being deleted.

The screenshot shows the 'Event subscriptions' section of the Amazon RDS console. It displays two entries in a table:

Name	Status
Configchangerdspgres	active
Postgresnotification	active

At the top of the table, there is a 'Delete' button which has been circled in red.

AWS CLI

To delete an Amazon RDS event notification subscription, use the AWS CLI `delete-event-subscription` command. Include the following required parameter:

- `--subscription-name`

Example

The following example deletes the subscription `myrdssubscription`.

```
aws rds delete-event-subscription --subscription-name myrdssubscription
```

API

To delete an Amazon RDS event notification subscription, use the RDS API `DeleteEventSubscription` command. Include the following required parameter:

- `SubscriptionName`

Creating a rule that triggers on an Amazon Aurora event

Using Amazon CloudWatch Events and Amazon EventBridge, you can automate AWS services and respond to system events such as application availability issues or resource changes.

Topics

- [Tutorial: log the state of an instance using EventBridge \(p. 692\)](#)

Tutorial: log the state of an instance using EventBridge

You can create an AWS Lambda function that logs the state changes for an instance. You can choose to create a rule that runs the function whenever there is a state transition or a transition to one or more states that are of interest.

In this tutorial, you log any state change of an existing RDS DB instance. The tutorial assumes that you have a small running test instance that you can shut down temporarily.

Important

Don't perform this tutorial on a running production instance.

Step 1: Create an AWS Lambda Function

Create a Lambda function to log the state change events. You specify this function when you create your rule.

To create a Lambda function

1. Open the AWS Lambda console at <https://console.aws.amazon.com/lambda/>.
2. If you're new to Lambda, you see a welcome page. Choose **Get Started Now**. Otherwise, choose **Create function**.
3. Choose **Author from scratch**.
4. On the **Create function** page, do the following:
 - a. Enter a name and description for the Lambda function. For example, name the function **RDSInstanceStateChange**.
 - b. In **Runtime**, select **Node.js 14.x**.
 - c. In **Execution role**, choose **Create a new role with basic Lambda permissions**. For **Existing role**, select your basic execution role. Otherwise, create a basic execution role.
 - d. Choose **Create function**.
5. On the **RDSInstanceStateChange** page, do the following:
 - a. In **Code source**, select **index.js**.
 - b. Right-click **index.js**, and choose **Open**.
 - c. In the **index.js** pane, delete the existing code.
 - d. Enter the following code:

```
console.log('Loading function');

exports.handler = async (event, context) => {
    console.log('Received event:', JSON.stringify(event));
};
```
 - e. Choose **Deploy**.

Step 2: Create a Rule

Create a rule to run your Lambda function whenever you launch an Amazon RDS instance.

To create the EventBridge rule

1. Open the Amazon EventBridge console at <https://console.aws.amazon.com/events/>.
2. In the navigation pane, choose **Rules**.
3. Choose **Create rule**.
4. Enter a name and description for the rule. For example, enter **RDSInstanceStateChangeRule**.
5. For **Define pattern**, do the following:
 - a. Choose **Event pattern**.
 - b. Choose **Pre-defined pattern by service**.
 - c. For **Service provider**, choose **AWS**.
 - d. For **Service Name**, choose **Relational Database Service (RDS)**.
 - e. For **Event type**, choose **RDS DB Instance Event**.
6. For **Select event bus**, choose **AWS default event bus**. When an AWS service in your account emits an event, it always goes to your account's default event bus.
7. For **Target**, choose **Lambda function**.
8. For **Function**, select the Lambda function that you created.
9. Choose **Create**.

Step 3: Test the Rule

To test your rule, shut down an RDS DB instance. After waiting a few minutes for the instance to shut down, verify that your Lambda function was invoked.

To test your rule by stopping a DB instance

1. Open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. Stop an RDS DB instance.
3. Open the Amazon EventBridge console at <https://console.aws.amazon.com/events/>.
4. In the navigation pane, choose **Rules**, choose the name of the rule that you created.
5. In **Rule details**, choose **Metrics for the rule**.

You are redirected to the Amazon CloudWatch console.
6. In **All metrics**, choose the name of the rule that you created.

The graph should indicate that the rule was invoked.
7. In the navigation pane, choose **Log groups**.
8. Choose the name of the log group for your Lambda function (`/aws/lambda/function-name`).
9. Choose the name of the log stream to view the data provided by the function for the instance that you launched. You should see a received event similar to the following:

```
{  
    "version": "0",  
    "id": "12a345b6-78c9-01d2-34e5-123f4ghi5j6k",  
    "detail-type": "RDS DB Instance Event",  
    "source": "aws.rds",  
    "account": "111111111111",  
    "time": "2021-03-19T19:34:09Z",  
    "region": "us-east-1",  
}
```

```
"resources": [
    "arn:aws:rds:us-east-1:111111111111:db:testdb"
],
"detail": {
    "EventCategories": [
        "notification"
    ],
    "SourceType": "DB_INSTANCE",
    "SourceArn": "arn:aws:rds:us-east-1:111111111111:db:testdb",
    "Date": "2021-03-19T19:34:09.293Z",
    "Message": "DB instance stopped",
    "SourceIdentifier": "testdb",
    "EventID": "RDS-EVENT-0087"
}
}
```

10. (Optional) When you're finished, you can open the Amazon RDS console and start the instance that you stopped.

Monitoring Amazon Aurora log files

You can view, download, and watch database logs using the AWS Management Console, the AWS Command Line Interface (AWS CLI), or the Amazon RDS API. Viewing, downloading, or watching transaction logs isn't supported.

Note

In some cases, logs contain hidden data. Therefore, the AWS Management Console might show content in a log file, but the log file might be empty when you download it.

Topics

- [Viewing and listing database log files \(p. 695\)](#)
- [Downloading a database log file \(p. 696\)](#)
- [Watching a database log file \(p. 697\)](#)
- [Publishing database logs to Amazon CloudWatch Logs \(p. 697\)](#)
- [Reading log file contents using REST \(p. 698\)](#)
- [Aurora MySQL database log files \(p. 700\)](#)
- [PostgreSQL database log files \(p. 706\)](#)

Viewing and listing database log files

You can view database log files for your DB engine by using the AWS Management Console. You can list what log files are available for download or monitoring by using the AWS CLI or Amazon RDS API.

Console

To view a database log file

1. Open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. In the navigation pane, choose **Databases**.
3. Choose the name of the DB instance that has the log file that you want to view.
4. Choose the **Logs & events** tab.
5. Scroll down to the **Logs** section.
6. In the **Logs** section, choose the log that you want to view, and then choose **View**.

AWS CLI

To list the available database log files for a DB instance, use the AWS CLI `describe-db-log-files` command.

The following example returns a list of log files for a DB instance named `my-db-instance`.

Example

```
aws rds describe-db-log-files --db-instance-identifier my-db-instance
```

RDS API

To list the available database log files for a DB instance, use the Amazon RDS API `DescribeDBLogFiles` action.

Downloading a database log file

You can use the AWS Management Console, AWS CLI or API to download a database log file.

Console

To download a database log file

1. Open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. In the navigation pane, choose **Databases**.
3. Choose the name of the DB instance that has the log file that you want to view.
4. Choose the **Logs & events** tab.
5. Scroll down to the **Logs** section.
6. In the **Logs** section, choose the button next to the log that you want to download, and then choose **Download**.
7. Open the context (right-click) menu for the link provided, and then choose **Save Link As**. Enter the location where you want the log file to be saved, and then choose **Save**.



AWS CLI

To download a database log file, use the AWS CLI command `download-db-log-file-portion`. By default, this command downloads only the latest portion of a log file. However, you can download an entire file by specifying the parameter `--starting-token 0`.

The following example shows how to download the entire contents of a log file called `log/ERROR.4` and store it in a local file called `errorlog.txt`.

Example

For Linux, macOS, or Unix:

```
aws rds download-db-log-file-portion \
    --db-instance-identifier myexampledb \
    --starting-token 0 --output text \
    --log-file-name log/ERROR.4 > errorlog.txt
```

For Windows:

```
aws rds download-db-log-file-portion ^
```

```
--db-instance-identifier myexampledb ^
--starting-token 0 --output text ^
--log-file-name log/ERROR.4 > errorlog.txt
```

RDS API

To download a database log file, use the Amazon RDS API [DownloadDBLogFilePortion](#) action.

Watching a database log file

You can monitor the contents of a log file by using the AWS Management Console.

Console

To watch a database log file

1. Open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. In the navigation pane, choose **Databases**.
3. Choose the name of the DB instance that has the log file that you want to view.
4. Choose the **Logs & events** tab.
5. In the **Logs** section, choose a log file, and then choose **Watch**.

Publishing database logs to Amazon CloudWatch Logs

In addition to viewing and downloading DB instance logs, you can publish logs to Amazon CloudWatch Logs. With CloudWatch Logs, you can perform real-time analysis of the log data, store the data in highly durable storage, and manage the data with the CloudWatch Logs Agent. AWS retains log data published to CloudWatch Logs for an indefinite time period unless you specify a retention period. For more information, see [Change log data retention in CloudWatch Logs](#).

Topics

- [Configuring CloudWatch log integration \(p. 697\)](#)
- [Engine-specific log information \(p. 698\)](#)

Configuring CloudWatch log integration

Before you enable log data publishing, make sure that you have a service-linked role in AWS Identity and Access Management (IAM). For more information about service-linked roles, see [Using service-linked roles for Amazon Aurora \(p. 1783\)](#).

To publish your database log files to CloudWatch Logs, choose which logs to publish. Make this choice in the **Advanced Settings** section when you create a new DB instance. You can also modify an existing DB instance to begin publishing.

Log exports

Select the log types to publish to Amazon CloudWatch Logs

- Audit log
- Error log
- General log
- Slow query log

IAM role

The following service-linked role is used for publishing logs to CloudWatch Logs.

RDS Service Linked Role

 Ensure that General, Slow Query, and Audit Logs are turned on. Error logs are enabled by default.

After you have enabled publishing, Amazon Aurora continuously streams all of the DB instance log records to a log group. For example, you have a log group `/aws/rds/cluster/<cluster_name>/log_type` for each type of log that you publish. This log group is in the same AWS Region as the database instance that generates the log.

After you have published log records, you can use CloudWatch Logs to search and filter the records. For more information about searching and filtering logs, see [Searching and filtering log data](#). For a tutorial explaining how to monitor RDS logs, see [Build proactive database monitoring for Amazon RDS with Amazon CloudWatch Logs, AWS Lambda, and Amazon SNS](#).

Engine-specific log information

For engine-specific information, see the following:

- [the section called “Publishing Aurora MySQL logs to CloudWatch Logs” \(p. 1017\)](#)
- [the section called “Publishing Aurora PostgreSQL logs to CloudWatch Logs” \(p. 1487\)](#)

Reading log file contents using REST

Amazon RDS provides a REST endpoint that allows access to DB instance log files. This is useful if you need to write an application to stream Amazon RDS log file contents.

The syntax is:

```
GET /v13/downloadCompleteLogFile/DBInstanceIdentifier/LogFileName HTTP/1.1
Content-type: application/json
host: rds.region.amazonaws.com
```

The following parameters are required:

- *DBInstanceIdentifier*—the name of the DB instance that contains the log file you want to download.
- *LogFileName*—the name of the log file to be downloaded.

The response contains the contents of the requested log file, as a stream.

The following example downloads the log file named *log/ERROR.6* for the DB instance named *sample-sql* in the *us-west-2* region.

```
GET /v13/downloadCompleteLogFile/sample-sql/log/ERROR.6 HTTP/1.1
host: rds.us-west-2.amazonaws.com
X-Amz-Security-Token: AQoDYXdzEIH//////////wEa0AIXLhngC5zp9CyB1R6abwKrXHVR5efnAVN3XvR7IwqKYalFSn6UyJuEFTft9nObglx4QJ+GXV9cpACkETq=
X-Amz-Date: 20140903T233749Z
X-Amz-Algorithm: AWS4-HMAC-SHA256
X-Amz-Credential: AKIADQKE4SARGYLE/20140903/us-west-2/rds/aws4_request
X-Amz-SignedHeaders: host
X-Amz-Content-SHA256: e3b0c44298fc1c229afbf4c8996fb92427ae41e4649b934de495991b7852b855
X-Amz-Expires: 86400
X-Amz-Signature: 353a4f14b3f250142d9afc34f9f9948154d46ce7d4ec091d0cdabbcf8b40c558
```

If you specify a nonexistent DB instance, the response consists of the following error:

- **DBInstanceNotFound**—*DB Instance Identifier* does not refer to an existing DB instance. (HTTP status code: 404)

Aurora MySQL database log files

You can monitor the Aurora MySQL logs directly through the Amazon RDS console, Amazon RDS API, AWS CLI, or AWS SDKs. You can also access MySQL logs by directing the logs to a database table in the main database and querying that table. You can use the `mysqlbinlog` utility to download a binary log.

For more information about viewing, downloading, and watching file-based database logs, see [Monitoring Amazon Aurora log files \(p. 695\)](#).

Topics

- [Overview of Aurora MySQL database logs \(p. 700\)](#)
- [Publishing Aurora MySQL logs to Amazon CloudWatch Logs \(p. 702\)](#)
- [Managing table-based Aurora MySQL logs \(p. 702\)](#)
- [Configuring Aurora MySQL binary logging \(p. 703\)](#)
- [Accessing MySQL binary logs \(p. 704\)](#)

Overview of Aurora MySQL database logs

You can monitor the following types of Aurora MySQL log files:

- Error log
- Slow query log
- General log
- The audit log

The Aurora MySQL error log is generated by default. You can generate the slow query and general logs by setting parameters in your DB parameter group.

Topics

- [Aurora MySQL error logs \(p. 700\)](#)
- [Aurora MySQL slow query and general logs \(p. 701\)](#)
- [Log rotation and retention \(p. 702\)](#)
- [Size limits on BLOBs \(p. 702\)](#)

Aurora MySQL error logs

Aurora MySQL writes errors in the `mysql-error.log` file. Each log file has the hour it was generated (in UTC) appended to its name. The log files also have a timestamp that helps you determine when the log entries were written.

Aurora MySQL writes to the error log only on startup, shutdown, and when it encounters errors. A DB instance can go hours or days without new entries being written to the error log. If you see no recent entries, it's because the server did not encounter an error that would result in a log entry.

Aurora MySQL writes `mysql-error.log` to disk every 5 minutes. MySQL appends the contents of the log to `mysql-error-running.log`.

Aurora MySQL rotates the `mysql-error-running.log` file every hour. Aurora MySQL removes the audit, general, slow query, and SDK logs after either 24 hours or when 15% of storage has been consumed.

Note

The log retention period is different between Amazon RDS and Aurora.

Aurora MySQL slow query and general logs

The Aurora MySQL slow query log and the general log can be written to a file or a database table by setting parameters in your DB parameter group. For information about creating and modifying a DB parameter group, see [Working with DB parameter groups and DB cluster parameter groups \(p. 339\)](#). You must set these parameters before you can view the slow query log or general log in the Amazon RDS console or by using the Amazon RDS API, Amazon RDS CLI, or AWS SDKs.

You can control Aurora MySQL logging by using the parameters in this list:

- `slow_query_log`: To create the slow query log, set to 1. The default is 0.
- `general_log`: To create the general log, set to 1. The default is 0.
- `long_query_time`: To prevent fast-running queries from being logged in the slow query log, specify a value for the shortest query run time to be logged, in seconds. The default is 10 seconds; the minimum is 0. If `log_output = FILE`, you can specify a floating point value that goes to microsecond resolution. If `log_output = TABLE`, you must specify an integer value with second resolution. Only queries whose run time exceeds the `long_query_time` value are logged. For example, setting `long_query_time` to 0.1 prevents any query that runs for less than 100 milliseconds from being logged.
- `log_queries_not_using_indexes`: To log all queries that do not use an index to the slow query log, set to 1. The default is 0. Queries that do not use an index are logged even if their run time is less than the value of the `long_query_time` parameter.
- `log_output option`: You can specify one of the following options for the `log_output` parameter.
 - **TABLE** – Write general queries to the `mysql.general_log` table, and slow queries to the `mysql.slow_log` table.
 - **FILE** – Write both general and slow query logs to the file system. Log files are rotated hourly.
 - **NONE** – Disable logging.

For Aurora MySQL 5.6, the default for `log_output` is **TABLE**. For Aurora MySQL 5.7, the default for `log_output` is **FILE**.

When logging is enabled, Amazon Aurora rotates table logs or deletes log files at regular intervals. This measure is a precaution to reduce the possibility of a large log file either blocking database use or affecting performance. **FILE** and **TABLE** logging approach rotation and deletion as follows:

- When **FILE** logging is enabled, log files are examined every hour and log files more than 30 days old are deleted. In some cases, the remaining combined log file size after the deletion might exceed the threshold of 2 percent of a DB instance's allocated space. In these cases, the oldest log files are deleted until the log file size no longer exceeds the threshold.
- When **TABLE** logging is enabled, in some cases log tables are rotated every 24 hours. This rotation occurs if the space used by the table logs is more than 20 percent of the allocated storage space or the size of all logs combined is greater than 10 GB. If the amount of space used for a DB instance is greater than 90 percent of the DB instance's allocated storage space, then the thresholds for log rotation are reduced. Log tables are then rotated if the space used by the table logs is more than 10 percent of the allocated storage space or the size of all logs combined is greater than 5 GB. You can subscribe to the `low_free_storage` event to be notified when log tables are rotated to free up space. For more information, see [Using Amazon RDS event notification \(p. 675\)](#).

When log tables are rotated, the current log table is copied to a backup log table and the entries in the current log table are removed. If the backup log table already exists, then it is deleted before the current log table is copied to the backup. You can query the backup log table if needed. The backup log table for the `mysql.general_log` table is named `mysql.general_log_backup`. The backup log table for the `mysql.slow_log` table is named `mysql.slow_log_backup`.

You can rotate the `mysql.general_log` table by calling the `mysql.rds_rotate_general_log` procedure. You can rotate the `mysql.slow_log` table by calling the `mysql.rds_rotate_slow_log` procedure.

Table logs are rotated during a database version upgrade.

To work with the logs from the Amazon RDS console, Amazon RDS API, Amazon RDS CLI, or AWS SDKs, set the `log_output` parameter to `FILE`. Like the MySQL error log, these log files are rotated hourly. The log files that were generated during the previous 30 days are retained. Note that the retention period is different between Amazon RDS and Aurora.

For more information about the slow query and general logs, go to the following topics in the MySQL documentation:

- [The slow query log](#)
- [The general query log](#)

Log rotation and retention

The Aurora MySQL slow query log, error log, and the general log file sizes are constrained to no more than 2 percent of the allocated storage space for a DB instance. To maintain this threshold, logs are automatically rotated every hour. Aurora MySQL removes logs after 24 hours or when 15% of disk space is reached. If the combined log file size exceeds the threshold after removing old log files, then the oldest log files are deleted until the log file size no longer exceeds the threshold.

Size limits on BLOBs

For Aurora MySQL, there is a size limit on BLOBs written to the redo log. To account for this limit, ensure that the `innodb_log_file_size` parameter for your Aurora MySQL DB instance is 10 times larger than the largest BLOB data size found in your tables, plus the length of other variable length fields (`VARCHAR`, `VARBINARY`, `TEXT`) in the same tables. For information on how to set parameter values, see [Working with DB parameter groups and DB cluster parameter groups \(p. 339\)](#). For information on the redo log BLOB size limit, go to [Changes in MySQL 5.6.20](#).

Publishing Aurora MySQL logs to Amazon CloudWatch Logs

You can configure your Aurora MySQL DB cluster to publish log data to a log group in Amazon CloudWatch Logs. With CloudWatch Logs, you can perform real-time analysis of the log data, and use CloudWatch to create alarms and view metrics. You can use CloudWatch Logs to store your log records in highly durable storage. For more information, see [Publishing Amazon Aurora MySQL logs to Amazon CloudWatch Logs \(p. 1017\)](#).

Managing table-based Aurora MySQL logs

You can direct the general and slow query logs to tables on the DB instance by creating a DB parameter group and setting the `log_output` server parameter to `TABLE`. General queries are then logged to the `mysql.general_log` table, and slow queries are logged to the `mysql.slow_log` table. You can query the tables to access the log information. Enabling this logging increases the amount of data written to the database, which can degrade performance.

Both the general log and the slow query logs are disabled by default. In order to enable logging to tables, you must also set the `general_log` and `slow_query_log` server parameters to 1.

Log tables keep growing until the respective logging activities are turned off by resetting the appropriate parameter to 0. A large amount of data often accumulates over time, which can use up a considerable

percentage of your allocated storage space. Amazon Aurora doesn't allow you to truncate the log tables, but you can move their contents. Rotating a table saves its contents to a backup table and then creates a new empty log table. You can manually rotate the log tables with the following command line procedures, where the command prompt is indicated by **PROMPT>**:

```
PROMPT> CALL mysql.rds_rotate_slow_log;
PROMPT> CALL mysql.rds_rotate_general_log;
```

To completely remove the old data and reclaim the disk space, call the appropriate procedure twice in succession.

Configuring Aurora MySQL binary logging

The *binary log* is a set of log files that contain information about data modifications made to an Aurora MySQL server instance. The binary log contains information such as the following:

- Events that describe database changes such as table creation or row modifications
- Information about the duration of each statement that updated data
- Events for statements that could have updated data but didn't

The binary log records statements that are sent during replication. It is also required for some recovery operations. For more information, see [The Binary Log](#) and [Binary Log Overview](#) in the MySQL documentation.

MySQL on Amazon Aurora supports the *row-based*, *statement-based*, and *mixed* binary logging formats for MySQL version 5.6 and later. The default binary logging format is mixed. For details on the different Aurora MySQL binary log formats, see [Binary logging formats](#) in the MySQL documentation.

If you plan to use replication, the binary logging format is important because it determines the record of data changes that is recorded in the source and sent to the replication targets. For information about the advantages and disadvantages of different binary logging formats for replication, see [Advantages and disadvantages of statement-based and row-based replication](#) in the MySQL documentation.

Important

Setting the binary logging format to row-based can result in very large binary log files. Large binary log files reduce the amount of storage available for a DB cluster and can increase the amount of time to perform a restore operation of a DB cluster.

Statement-based replication can cause inconsistencies between the source DB cluster and a read replica. For more information, see [Determination of safe and unsafe statements in binary logging](#) in the MySQL documentation.

To set the MySQL binary logging format

1. Open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. In the navigation pane, choose **Parameter groups**.
3. Choose the parameter group used by the DB cluster you want to modify.

You can't modify a default parameter group. If the DB cluster is using a default parameter group, create a new parameter group and associate it with the DB cluster.

For more information on parameter groups, see [Working with DB parameter groups and DB cluster parameter groups \(p. 339\)](#).

4. From **Parameter group actions**, choose **Edit**.
5. Set the `binlog_format` parameter to the binary logging format of your choice (**ROW**, **STATEMENT**, or **MIXED**). You can also use the value **OFF** to turn off binary logging.
6. Choose **Save changes** to save the updates to the DB cluster parameter group.

Important

Changing a DB cluster parameter group affects all DB clusters that use that parameter group. If you want to specify different binary logging formats for different Aurora MySQL DB clusters in an AWS Region, the DB clusters must use different DB cluster parameter groups. These parameter groups identify different logging formats. Assign the appropriate DB cluster parameter group to each DB clusters. For more information about Aurora MySQL parameters, see [Aurora MySQL configuration parameters \(p. 1042\)](#).

Accessing MySQL binary logs

You can use the mysqlbinlog utility to download or stream binary logs from Amazon RDS instances running MySQL 5.6 or later. The binary log is downloaded to your local computer, where you can perform actions such as replaying the log using the mysql utility. For more information about using the mysqlbinlog utility, go to [Using mysqlbinlog to back up binary log files](#).

To run the mysqlbinlog utility against an Amazon RDS instance, use the following options:

- Specify the `--read-from-remote-server` option.
- `--host`: Specify the DNS name from the endpoint of the instance.
- `--port`: Specify the port used by the instance.
- `--user`: Specify a MySQL user that has been granted the replication slave permission.
- `--password`: Specify the password for the user, or omit a password value so that the utility prompts you for a password.
- To have the file downloaded in binary format, specify the `--raw` option.
- `--result-file`: Specify the local file to receive the raw output.
- Specify the names of one or more binary log files. To get a list of the available logs, use the SQL command SHOW BINARY LOGS.
- To stream the binary log files, specify the `--stop-never` option.

For more information about mysqlbinlog options, go to [mysqlbinlog - utility for processing binary log files](#).

For example, see the following.

For Linux, macOS, or Unix:

```
mysqlbinlog \
  --read-from-remote-server \
  --host=MySQL56Instance1.cg034hpkmmt.region.rds.amazonaws.com \
  --port=3306 \
  --user ReplUser \
  --password \
  --raw \
  --result-file=/tmp/ \
  binlog.00098
```

For Windows:

```
mysqlbinlog ^
  --read-from-remote-server ^
  --host=MySQL56Instance1.cg034hpkmmt.region.rds.amazonaws.com ^
  --port=3306 ^
  --user ReplUser ^
  --password ^
  --raw ^
  --result-file=/tmp/ ^
```

```
binlog.00098
```

Amazon RDS normally purges a binary log as soon as possible, but the binary log must still be available on the instance to be accessed by mysqlbinlog. To specify the number of hours for RDS to retain binary logs, use the `mysql.rds_set_configuration` stored procedure and specify a period with enough time for you to download the logs. After you set the retention period, monitor storage usage for the DB instance to ensure that the retained binary logs don't take up too much storage.

Note

The `mysql.rds_set_configuration` stored procedure is only available for MySQL version 5.6 or later.

The following example sets the retention period to 1 day.

```
call mysql.rds_set_configuration('binlog retention hours', 24);
```

To display the current setting, use the `mysql.rds_show_configuration` stored procedure.

```
call mysql.rds_show_configuration;
```

PostgreSQL database log files

Aurora PostgreSQL generates query and error logs. You can use log messages to troubleshoot performance and auditing issues while using the database.

To view, download, and watch file-based database logs, see [Monitoring Amazon Aurora log files \(p. 695\)](#).

Topics

- [Overview of PostgreSQL logs \(p. 706\)](#)
- [Setting the log retention period \(p. 707\)](#)
- [Setting log file rotation \(p. 707\)](#)
- [Setting the message format \(p. 708\)](#)
- [Enabling query logging \(p. 708\)](#)

Overview of PostgreSQL logs

PostgreSQL generates event log files that contain useful information for DBAs.

Log contents

The default logging level captures errors that affect your server. By default, Aurora PostgreSQL logging parameters capture all server errors, including the following:

- Query failures
- Login failures
- Fatal server errors
- Deadlocks

To identify application issues, you can use the preceding error messages. For example, if you converted a legacy application from Oracle to Aurora PostgreSQL, some queries may not convert correctly. These incorrectly formatted queries generate error messages in the logs, which you can use to identify the problematic code.

You can modify PostgreSQL logging parameters to capture additional information, including the following:

- Connections and disconnections
- Checkpoints
- Schema modification queries
- Queries waiting for locks
- Queries consuming temporary disk storage
- Backend autovacuum process consuming resources

The preceding log information can help troubleshoot potential performance and auditing issues. For more information, see [Error reporting and logging](#) in the PostgreSQL documentation. For a useful AWS blog about PostgreSQL logging, see [Working with RDS and Aurora PostgreSQL logs: Part 1](#) and [Working with RDS and Aurora PostgreSQL logs: Part 2](#).

Parameter groups

Each Aurora PostgreSQL instance is associated with a *parameter group* that contains the engine specific configurations. The engine configurations also include several parameters that control PostgreSQL

logging behavior. AWS provides the parameter groups with default configuration settings to use for your instances. However, to change the default settings, you must create a clone of the default parameter group, modify it, and attach it to your instance.

To set logging parameters for a DB instance, set the parameters in a DB parameter group and associate that parameter group with the DB instance. For more information, see [Working with DB parameter groups and DB cluster parameter groups \(p. 339\)](#).

Setting the log retention period

To set the retention period for system logs, use the `rds.log_retention_period` parameter. You can find `rds.log_retention_period` in the DB parameter group associated with your DB instance. The unit for this parameter is minutes. For example, a setting of 1,440 retains logs for one day. The default value is 4,320 (three days). The maximum value is 10,080 (seven days). Your instance must have enough allocated storage to contain the retained log files.

Amazon Aurora compresses older PostgreSQL logs when storage for the DB instance reaches a threshold. Aurora compresses the files using the gzip compression utility; for information on gzip, see the [gzip](#) website. When storage for the DB instance is low and all available logs are compressed, you get a warning like the following.

Warning: local storage for PostgreSQL log files is critically low for this Aurora PostgreSQL instance, and could lead to a database outage.

Note

If storage gets too low, Aurora might delete compressed PostgreSQL logs before the retention period expires. If logs are deleted early, you get a message like the following.

The oldest PostgreSQL log files were deleted due to local storage constraints.

To retain older logs, publish them to Amazon CloudWatch Logs. For more information, see [Publishing Aurora PostgreSQL logs to Amazon CloudWatch Logs \(p. 1487\)](#). After you set up CloudWatch publishing, Aurora doesn't delete a log until it's published to CloudWatch Logs.

Setting log file rotation

To control PostgreSQL log file rotation, set two parameters in the DB parameter group associated with your DB instance: `log_rotation_age` and `log_rotation_size`. These two settings control when a new, distinct log file is created.

The log file names are based on the file name pattern of the `log_filename` parameter. For example, to provide log files names with a granularity of less than an hour, set `log_filename` to the minute format: `postgresql.log.%Y-%m-%d-%H%M`. Granularity of less than an hour is only supported for PostgreSQL version 10 and higher. To use a granularity in hours for log file names, set `log_filename` to the hour format: `postgresql.log.%Y-%m-%d-%H`.

To control log file rotation based on time, set the `log_rotation_age` parameter to anywhere from 1 minute to 1,440 minutes (24 hours). The `log_rotation_age` default is 60 minutes. If you set the `log_rotation_age` parameter to less than 60 minutes, also set the `log_filename` parameter to the minute format.

To control log file rotation based on file size, set the `log_rotation_size` parameter to anywhere from 50,000 to 1,000,000 KB. The default is 100,000 KB. We recommend that you also set the `log_filename` parameter to the minute format. Doing this makes sure that you can create a new log file in less than an hour if the `log_rotation_age` parameter is 60 minutes or greater.

Setting the message format

By default, Aurora PostgreSQL generates logs in standard error (stderr) format. In this format, each log message is prefixed with the information specified by the parameter `log_line_prefix`. Aurora only allows the following value for `log_line_prefix`:

```
%t:%r:%u@%d:[%p]:t
```

The preceding value maps to the following code:

```
log-time : remote-host : user-name @ db-name : [ process-id ]
```

For example, the following error message results from querying a column using the wrong name.

```
2019-03-10 03:54:59 UTC:10.0.0.123(52834):postgres@tstldb:[20175]:ERROR: column "wrong" does not exist at character 8
```

To specify the format for output logs, use the parameter `log_destination`. To make the instance generate both standard and CSV output files, set `log_destination` to `csvlog` in your instance parameter group. For a discussion of PostgreSQL logs, see [Working with RDS and Aurora PostgreSQL logs: Part 1](#).

Enabling query logging

To enable query logging for your PostgreSQL DB instance, set two parameters in the DB parameter group associated with your DB instance: `log_statement` and `log_min_duration_statement`.

The `log_statement` parameter controls which SQL statements are logged. The default value is `none`. We recommend that when you debug issues in your DB instance, set this parameter to `all` to log all statements. To log all data definition language (DDL) statements (`CREATE`, `ALTER`, `DROP`, and so on), set this value to `dd1`. To log all DDL and data modification language (DML) statements (`INSERT`, `UPDATE`, `DELETE`, and so on), set the value to `mod`.

Warning

Sensitive information such as passwords can be exposed if you set the `log_statement` parameter to `dd1`, `mod`, or `all`. To avoid this risk, set the `log_statement` to `none`. Also consider the following solutions:

- Encrypt the sensitive information on the client side and use the `ENCRYPTED` and `UNENCRYPTED` options of the `CREATE` and `ALTER` statements.
- Restrict access to the CloudWatch logs.
- Use stronger authentication mechanisms such as IAM.

For auditing, you can use the PostgreSQL `pgAudit` extension because it redacts the sensitive information for `CREATE` and `ALTER` commands.

The `log_min_duration_statement` parameter sets the limit in milliseconds of a statement to be logged. All SQL statements that run longer than the parameter setting are logged. This parameter is disabled and set to `-1` by default. Enabling this parameter can help you find unoptimized queries.

To set up query logging, take the following steps:

1. Set the `log_statement` parameter to `all`. The following example shows the information that is written to the `postgres.log` file.

```
2013-11-05 16:48:56 UTC::@[2952]:LOG: received SIGHUP, reloading configuration files
2013-11-05 16:48:56 UTC::@[2952]:LOG: parameter "log_statement" changed to "all"
```

Additional information is written to the `postgres.log` file when you run a query. The following example shows the type of information written to the file after a query.

```
2013-11-05 16:41:07 UTC::@[2955]:LOG: checkpoint starting: time
2013-11-05 16:41:07 UTC::@[2955]:LOG: checkpoint complete: wrote 1 buffers (0.3%),
0 transaction log file(s) added, 0 removed, 1 recycled; write=0.000 s, sync=0.003 s,
total=0.012 s; sync files=1, longest=0.003 s, average=0.003 s
2013-11-05 16:45:14 UTC:[local]:master@postgres:[8839]:LOG: statement: SELECT d.datname
as "Name",
    pg_catalog.pg_get_userbyid(d.datdba) as "Owner",
    pg_catalog.pg_encoding_to_char(d.encoding) as "Encoding",
    d.datcollate as "Collate",
    d.datctype as "Ctype",
    pg_catalog.array_to_string(d.datacl, E'\n') AS "Access privileges"
FROM pg_catalog.pg_database d
ORDER BY 1;
2013-11-05 16:45:
```

- Set the `log_min_duration_statement` parameter. The following example shows the information that is written to the `postgres.log` file when the parameter is set to 1.

```
2013-11-05 16:48:56 UTC::@[2952]:LOG: received SIGHUP, reloading configuration files
2013-11-05 16:48:56 UTC::@[2952]:LOG: parameter "log_min_duration_statement" changed to
"1"
```

Additional information is written to the `postgres.log` file when you run a query that exceeds the duration parameter setting. The following example shows the type of information written to the file after a query.

```
2013-11-05 16:51:10 UTC:[local]:master@postgres:[9193]:LOG: statement: SELECT
c2.relname, i.indisprimary, i.indisunique, i.indisclustered, i.indisvalid,
pg_catalog.pg_get_indexdef(i.indexrelid, 0, true),
    pg_catalog.pg_get_constraintdef(con.oid, true), contype, condeferrable, condeferred,
c2.reltablename
FROM pg_catalog.pg_class c, pg_catalog.pg_class c2, pg_catalog.pg_index i
    LEFT JOIN pg_catalog.pg_constraint con ON (conrelid = i.indrelid AND conindid =
i.indexrelid AND contype IN ('p','u','x'))
WHERE c.oid = '1255' AND c.oid = i.indrelid AND i.indexrelid = c2.oid
ORDER BY i.indisprimary DESC, i.indisunique DESC, c2.relname;
2013-11-05 16:51:10 UTC:[local]:master@postgres:[9193]:LOG: duration: 3.367 ms
2013-11-05 16:51:10 UTC:[local]:master@postgres:[9193]:LOG: statement: SELECT
c.oid::pg_catalog.regclass FROM pg_catalog.pg_class c, pg_catalog.pg_inherits i WHERE
c.oid=i.inhparent AND i.inhrelid = '1255' ORDER BY inhseqno;
2013-11-05 16:51:10 UTC:[local]:master@postgres:[9193]:LOG: duration: 1.002 ms
2013-11-05 16:51:10 UTC:[local]:master@postgres:[9193]:LOG: statement:
    SELECT c.oid::pg_catalog.regclass FROM pg_catalog.pg_class c,
    pg_catalog.pg_inherits i WHERE c.oid=i.inhrelid AND i.inhparent = '1255' ORDER BY
    c.oid::pg_catalog.regclass::pg_catalog.text;
2013-11-05 16:51:18 UTC:[local]:master@postgres:[9193]:LOG: statement: select proname
from pg_proc;
2013-11-05 16:51:18 UTC:[local]:master@postgres:[9193]:LOG: duration: 3.469 ms
```

Monitoring Amazon Aurora API calls in AWS CloudTrail

AWS CloudTrail is an AWS service that helps you audit your AWS account. AWS CloudTrail is turned on for your AWS account when you create it. For more information about CloudTrail, see the [AWS CloudTrail User Guide](#).

Topics

- [CloudTrail integration with Amazon Aurora \(p. 710\)](#)
- [Amazon Aurora log file entries \(p. 710\)](#)

CloudTrail integration with Amazon Aurora

All Amazon Aurora actions are logged by CloudTrail. CloudTrail provides a record of actions taken by a user, role, or an AWS service in Amazon Aurora.

CloudTrail events

CloudTrail captures API calls for Amazon Aurora as events. An *event* represents a single request from any source and includes information about the requested action, the date and time of the action, request parameters, and so on. Events include calls from the Amazon RDS console and from code calls to the Amazon RDS API operations.

Amazon Aurora activity is recorded in a CloudTrail event in **Event history**. You can use the CloudTrail console to view the last 90 days of recorded API activity and events in an AWS Region. For more information, see [Viewing events with CloudTrail event history](#).

CloudTrail trails

For an ongoing record of events in your AWS account, including events for Amazon Aurora, create a *trail*. A trail is a configuration that enables delivery of events to a specified Amazon S3 bucket. CloudTrail typically delivers log files within 15 minutes of account activity.

Note

If you don't configure a trail, you can still view the most recent events in the CloudTrail console in **Event history**.

You can create two types of trails for an AWS account: a trail that applies to all Regions, or a trail that applies to one Region. By default, when you create a trail in the console, the trail applies to all Regions.

Additionally, you can configure other AWS services to further analyze and act upon the event data collected in CloudTrail logs. For more information, see:

- [Overview for creating a trail](#)
- [CloudTrail supported services and integrations](#)
- [Configuring Amazon SNS notifications for CloudTrail](#)
- [Receiving CloudTrail log files from multiple Regions](#) and [Receiving CloudTrail log files from multiple accounts](#)

Amazon Aurora log file entries

CloudTrail log files contain one or more log entries. CloudTrail log files are not an ordered stack trace of the public API calls, so they do not appear in any specific order.

The following example shows a CloudTrail log entry that demonstrates the `CreateDBInstance` action.

```
{
    "eventVersion": "1.04",
    "userIdentity": {
        "type": "IAMUser",
        "principalId": "AKIAIOSFODNN7EXAMPLE",
        "arn": "arn:aws:iam::123456789012:user/johndoe",
        "accountId": "123456789012",
        "accessKeyId": "AKIAI44QH8DHBEXAMPLE",
        "userName": "johndoe"
    },
    "eventTime": "2018-07-30T22:14:06Z",
    "eventSource": "rds.amazonaws.com",
    "eventName": "CreateDBInstance",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "192.0.2.0",
    "userAgent": "aws-cli/1.15.42 Python/3.6.1 Darwin/17.7.0 botocore/1.10.42",
    "requestParameters": {
        "enableCloudwatchLogsExports": [
            "audit",
            "error",
            "general",
            "slowquery"
        ],
        "dBInstanceIdentifier": "test-instance",
        "engine": "mysql",
        "masterUsername": "myawsuser",
        "allocatedStorage": 20,
        "dBInstanceClass": "db.m1.small",
        "masterUserPassword": "*****"
    },
    "responseElements": {
        "dBInstanceArn": "arn:aws:rds:us-east-1:123456789012:db:test-instance",
        "storageEncrypted": false,
        "preferredBackupWindow": "10:27-10:57",
        "preferredMaintenanceWindow": "sat:05:47-sat:06:17",
        "backupRetentionPeriod": 1,
        "allocatedStorage": 20,
        "storageType": "standard",
        "engineVersion": "5.6.39",
        "dBInstancePort": 0,
        "optionGroupMemberships": [
            {
                "status": "in-sync",
                "optionGroupName": "default:mysql-5-6"
            }
        ],
        "dBParameterGroups": [
            {
                "dBParameterGroupName": "default.mysql5.6",
                "parameterApplyStatus": "in-sync"
            }
        ],
        "monitoringInterval": 0,
        "dBInstanceClass": "db.m1.small",
        "readReplicaDBInstanceIdentifiers": [],
        "dBSubnetGroup": {
            "dBSubnetGroupName": "default",
            "dBSubnetGroupDescription": "default",
            "subnets": [
                {
                    "subnetAvailabilityZone": {"name": "us-east-1b"},
                    "subnetIdentifier": "subnet-cbfff283",
                    "status": "in-sync"
                }
            ]
        }
    }
}
```

```

        "subnetStatus": "Active"
    },
    {
        "subnetAvailabilityZone": {"name": "us-east-1e"},
        "subnetIdentifier": "subnet-d7c825e8",
        "subnetStatus": "Active"
    },
    {
        "subnetAvailabilityZone": {"name": "us-east-1f"},
        "subnetIdentifier": "subnet-6746046b",
        "subnetStatus": "Active"
    },
    {
        "subnetAvailabilityZone": {"name": "us-east-1c"},
        "subnetIdentifier": "subnet-bac383e0",
        "subnetStatus": "Active"
    },
    {
        "subnetAvailabilityZone": {"name": "us-east-1d"},
        "subnetIdentifier": "subnet-42599426",
        "subnetStatus": "Active"
    },
    {
        "subnetAvailabilityZone": {"name": "us-east-1a"},
        "subnetIdentifier": "subnet-da327bf6",
        "subnetStatus": "Active"
    }
],
"vpcId": "vpc-136a4c6a",
"subnetGroupStatus": "Complete"
},
"masterUsername": "myawsuser",
"multiAZ": false,
"autoMinorVersionUpgrade": true,
"engine": "mysql",
"caCertificateIdentifier": "rds-ca-2015",
"dbiResourceId": "db-ETDZIIXHEWY5N7GXVC4SH7H5IA",
"dBSecurityGroups": [],
"pendingModifiedValues": {
    "masterUserPassword": "*****",
    "pendingCloudwatchLogsExports": {
        "logTypesToEnable": [
            "audit",
            "error",
            "general",
            "slowquery"
        ]
    }
},
"dBInstanceStatus": "creating",
"publiclyAccessible": true,
"domainMemberships": [],
"copyTagsToSnapshot": false,
"dBInstanceIdentifier": "test-instance",
"licenseModel": "general-public-license",
"iAMDatabaseAuthenticationEnabled": false,
"performanceInsightsEnabled": false,
"vpcSecurityGroups": [
{
    "status": "active",
    "vpcSecurityGroupId": "sg-f839b688"
}
]
},
"requestID": "daf2e3f5-96a3-4df7-a026-863f96db793e",
"eventID": "797163d3-5726-441d-80a7-6eeb7464acd4",

```

```
    "eventType": "AwsApiCall",
    "recipientAccountId": "123456789012"
}
```

As shown in the `userIdentity` element in the preceding example, every event or log entry contains information about who generated the request. The identity information helps you determine the following:

- Whether the request was made with root or IAM user credentials.
- Whether the request was made with temporary security credentials for a role or federated user.
- Whether the request was made by another AWS service.

For more information about the `userIdentity`, see the [CloudTrail userIdentity element](#). For more information about `CreateDBInstance` and other Amazon Aurora actions, see the [Amazon RDS API Reference](#).

Monitoring Amazon Aurora with Database Activity Streams

By using Database Activity Streams, you can monitor near-real-time streams of database activity.

Topics

- [Overview of Database Activity Streams \(p. 714\)](#)
- [Network prerequisites for Aurora MySQL database activity streams \(p. 717\)](#)
- [Starting a database activity stream \(p. 718\)](#)
- [Getting the status of a database activity stream \(p. 720\)](#)
- [Stopping a database activity stream \(p. 720\)](#)
- [Monitoring database activity streams \(p. 721\)](#)
- [Managing access to database activity streams \(p. 743\)](#)

Overview of Database Activity Streams

As an Amazon Aurora database administrator, you need to safeguard your database and meet compliance and regulatory requirements. One strategy is to integrate database activity streams with your monitoring tools. In this way, you monitor and set alarms for auditing activity in your Amazon Aurora cluster.

Security threats are both external and internal. To protect against internal threats, you can control administrator access to data streams by configuring the Database Activity Streams feature. DBAs don't have access to the collection, transmission, storage, and processing of the streams.

Topics

- [How database activity streams work \(p. 714\)](#)
- [Asynchronous and synchronous mode for database activity streams \(p. 715\)](#)
- [Requirements for database activity streams \(p. 716\)](#)
- [Supported Aurora engine versions for database activity streams \(p. 716\)](#)
- [Supported DB instance classes for database activity streams \(p. 716\)](#)
- [Supported AWS Regions for database activity streams \(p. 717\)](#)

How database activity streams work

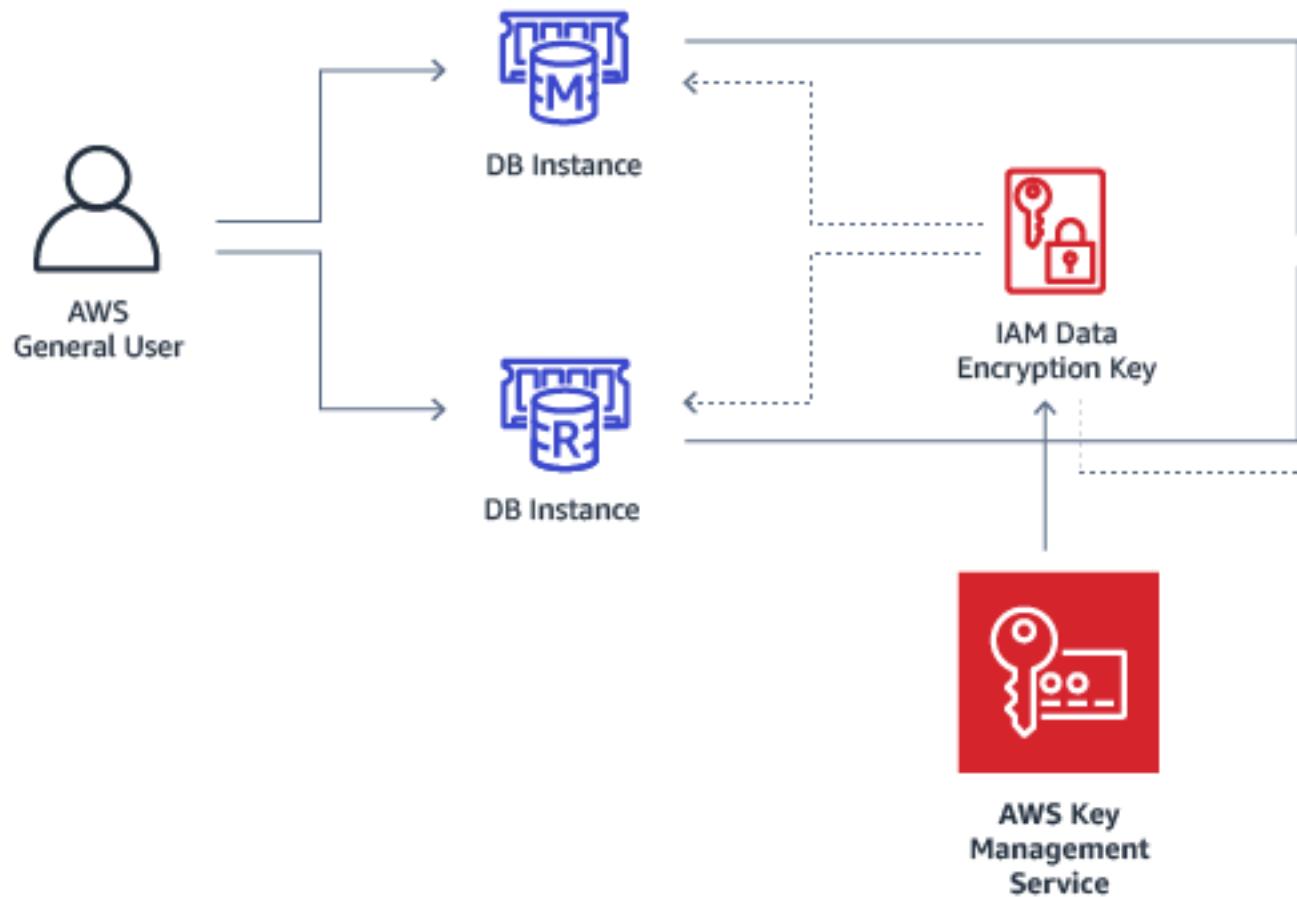
Amazon Aurora pushes activities to an Amazon Kinesis data stream in near real time. The Kinesis stream is created automatically. From Kinesis, you can configure AWS services such as Amazon Kinesis Data Firehose and AWS Lambda to consume the stream and store the data.

Important

Use of the Database Activity Streams feature in Amazon Aurora and Amazon RDS is free, but Amazon Kinesis charges for a data stream. For more information, see [Amazon Kinesis Data Streams pricing](#).

Applications for compliance management can also consume database activity streams. For Aurora PostgreSQL, compliance applications include IBM's Security Guardium and Imperva's SecureSphere Database Audit and Protection. These applications can use the stream to generate alerts and audit activity on your Aurora DB cluster.

The following graphic shows an Aurora DB cluster configured with Amazon Kinesis Data Firehose.



Asynchronous and synchronous mode for database activity streams

You can choose to have the database session handle database activity events in either of the following modes:

- **Asynchronous mode** – When a database session generates an activity stream event, the session returns to normal activities immediately. In the background, the activity stream event is made a durable record. If an error occurs in the background task, an RDS event is sent. This event indicates the beginning and end of any time windows where activity stream event records might have been lost.

Asynchronous mode favors database performance over the accuracy of the activity stream.

Note

Asynchronous mode is available for both Aurora PostgreSQL and Aurora MySQL.

- **Synchronous mode** – When a database session generates an activity stream event, the session blocks other activities until the event is made durable. If the event can't be made durable for some reason, the database session returns to normal activities. However, an RDS event is sent indicating that activity stream records might be lost for some time. A second RDS event is sent after the system is back to a healthy state.

The synchronous mode favors the accuracy of the activity stream over database performance.

Note

Synchronous mode is available for Aurora PostgreSQL. You can't use synchronous mode with Aurora MySQL.

Requirements for database activity streams

In Aurora, database activity streams have the following requirements and limitations.

Topics

- [Miscellaneous requirements \(p. 716\)](#)

Miscellaneous requirements

- Database activity streams require use of AWS Key Management Service (AWS KMS). AWS KMS is required because the activity streams are always encrypted.
- Database activity streams require use of Amazon Kinesis.

Supported Aurora engine versions for database activity streams

For Aurora PostgreSQL, database activity streams are supported for the following versions:

- All 13 versions
- All 12 versions
- Version 11.6 and higher 11 versions
- Version 10.11 and higher 10 versions

For more information about Aurora PostgreSQL versions, see [Amazon Aurora PostgreSQL releases and engine versions \(p. 1599\)](#).

For Aurora MySQL, database activity streams are supported for version 2.08 or higher, which is compatible with MySQL version 5.7.

Note

Database activity streams aren't supported in Aurora Serverless.

Supported DB instance classes for database activity streams

For Aurora MySQL, you can use database activity streams with the following DB instance classes:

- db.r6g
- db.r5
- db.r4
- db.r3
- db.x2g

For Aurora PostgreSQL, you can use database activity streams with the following DB instance classes:

- db.r6g
- db.r5

- db.r4
- db.x2g

Supported AWS Regions for database activity streams

Database activity streams are supported in all AWS Regions except the following:

- China (Beijing) Region, `cn-north-1`
- China (Ningxia) Region, `cn-northwest-1`
- AWS GovCloud (US-East), `us-gov-east-1`
- AWS GovCloud (US-West), `us-gov-west-1`

Network prerequisites for Aurora MySQL database activity streams

In the following section, you can find how to configure your virtual private cloud (VPC) for use with database activity streams.

Topics

- [Prerequisites for AWS KMS endpoints \(p. 717\)](#)
- [Prerequisites for public availability \(p. 717\)](#)
- [Prerequisites for private availability \(p. 717\)](#)

Prerequisites for AWS KMS endpoints

Instances in an Aurora MySQL cluster that use activity streams must be able to access AWS KMS endpoints. Make sure this requirement is satisfied before enabling database activity streams for your Aurora MySQL cluster. If the Aurora cluster is publicly available, this requirement is satisfied automatically.

Important

If the Aurora MySQL DB cluster can't access the AWS KMS endpoint, the activity stream stops. In that case, Aurora notifies you about this issue using RDS Events.

Prerequisites for public availability

For an Aurora DB cluster to be public, it must meet the following requirements:

- **Publicly Accessible** is **Yes** in the AWS Management Console cluster details page.
- The DB cluster is in an Amazon VPC public subnet. For more information about publicly accessible DB instances, see [Working with a DB instance in a VPC \(p. 1787\)](#). For more information about public Amazon VPC subnets, see [Your VPC and Subnets](#).

Prerequisites for private availability

If your Aurora DB cluster isn't publicly accessible, and it's in a VPC public subnet, it's private. To keep your cluster private and use it with database activity streams, you have the following options:

- Configure Network Address Translation (NAT) in your VPC. For more information, see [NAT Gateways](#).

- Create an AWS KMS endpoint in your VPC. This option is recommended because it's easier to configure.

To create an AWS KMS endpoint in your VPC

1. Open the Amazon VPC console at <https://console.aws.amazon.com/vpc/>.
2. In the navigation pane, choose **Endpoints**.
3. Choose **Create Endpoint**.
The **Create Endpoint** page appears.
4. Do the following:
 - In **Service category**, choose **AWS services**.
 - In **Service Name**, choose `com.amazonaws.region.kms`, where `region` is the AWS Region where your cluster is located.
 - For **VPC**, choose the VPC where your cluster is located.
5. Choose **Create Endpoint**.

For more information about configuring VPC endpoints, see [VPC Endpoints](#).

Starting a database activity stream

To monitor database activity for all instances of the DB cluster, start an activity stream at the cluster level. Any DB instances that you add to the cluster are also automatically monitored.

When you start an activity stream, each database activity event, such as a change or access, generates an activity stream event. SQL commands such as `CONNECT` and `SELECT` generate access events. SQL commands such as `CREATE` and `INSERT` generate change events.

Console

To start a database activity stream

1. Open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. In the navigation pane, choose **Databases**.
3. Choose the DB cluster for which you want to enable an activity stream.
4. For **Actions**, choose **Start activity stream**.

The **Start database activity stream: name** window appears, where `name` is your DB cluster.

5. Enter the following settings:

- For **AWS KMS key**, choose a key from the list of AWS KMS keys.

Note

If your Aurora MySQL cluster can't access KMS keys, follow the instructions in [Network prerequisites for Aurora MySQL database activity streams \(p. 717\)](#) to enable such access first.

Aurora uses the KMS key to encrypt the key that in turn encrypts database activity. Choose a KMS key other than the default key. For more information about encryption keys and AWS KMS, see [What is AWS Key Management Service?](#) in the *AWS Key Management Service Developer Guide*.

- For **Database activity stream mode**, choose **Asynchronous** or **Synchronous**.

Note

This choice applies only to Aurora PostgreSQL. For Aurora MySQL, you can use only asynchronous mode.

- Choose **Immediately**.

When you choose **Immediately**, the DB cluster restarts right away. If you choose **During the next maintenance window**, the DB cluster doesn't restart right away. In this case, the database activity stream doesn't start until the next maintenance window.

When you're done entering settings, choose **Start database activity stream**.

The status for the DB cluster shows that the activity stream is starting.

AWS CLI

To start database activity streams for a DB cluster , configure the DB cluster using the [start-activity-stream](#) AWS CLI command.

- `--kms-key-id key` – Specifies the KMS key identifier for encrypting messages in the database activity stream. The AWS KMS key identifier is the key ARN, key ID, alias ARN, or alias name for the AWS KMS key.
- `--resource-arn arn` – Specifies the Amazon Resource Name (ARN) of the DB cluster.
- `--region` – Identifies the AWS Region for the DB instance.
- `--mode sync-or-async` – Specifies either synchronous (`sync`) or asynchronous (`async`) mode. For Aurora PostgreSQL, you can choose either value. For Aurora MySQL, specify `async`.
- `--apply-immediately` – Applies the change immediately. This parameter is optional. If you don't specify this parameter, the database activity stream starts at the next maintenance interval.

For Linux, macOS, or Unix:

```
aws rds --region MY_REGION \
    start-activity-stream \
    --mode [sync | async] \
    --kms-key-id MY_KMS_KEY_ARN \
    --resource-arn MY_CLUSTER_ARN \
    --apply-immediately
```

For Windows:

```
aws rds --region MY_REGION ^
    start-activity-stream ^
    --mode [sync | async] ^
    --kms-key-id MY_KMS_KEY_ARN ^
    --resource-arn MY_CLUSTER_ARN ^
    --apply-immediately
```

RDS API

To start database activity streams for a DB cluster, configure the cluster using the [StartActivityStream](#) operation.

Call the action with the parameters below:

- Region
- Mode
- ApplyImmediately

Getting the status of a database activity stream

You can get the status of an activity stream using the console or AWS CLI.

Console

To get the status of a database activity stream

1. Open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. In the navigation pane, choose **Databases**, and then choose the DB cluster link.
3. Choose the **Configuration** tab, and check **Database activity stream** for status.

AWS CLI

You can get the activity stream configuration for a DB cluster as the response to a [describe-db-clusters](#) CLI request. In the following example, see the values for `ActivityStreamKinesisStreamName`, `ActivityStreamStatus`, `ActivityStreamKmsKeyId`, and `ActivityStreamMode`.

The request is as follows.

```
aws rds --region MY_REGION describe-db-clusters --db-cluster-identifier my-cluster
```

The response includes the following items for a database activity stream.

The following example shows a JSON response. These fields are the same for Aurora PostgreSQL and Aurora MySQL, except that `ActivityStreamMode` is always `async` for Aurora MySQL, while for Aurora PostgreSQL it might be `sync` or `async`.

```
{  
    "DBClusters": [  
        {  
            "DBClusterIdentifier": "my-cluster",  
            ...  
            "ActivityStreamKinesisStreamName": "aws-rds-das-cluster-  
A6TSYXITZCZXJHIRVFUBZ5LTWY",  
            "ActivityStreamStatus": "starting",  
            "ActivityStreamKmsKeyId": "12345678-abcd-efgh-ijkl-bd041f170262",  
            "ActivityStreamMode": "async",  
            "DbClusterResourceId": "cluster-ABCD123456"  
            ...  
        }  
    ]  
}
```

RDS API

You can get the activity stream configuration for a DB cluster as the response to a [DescribeDBClusters](#) operation.

Stopping a database activity stream

You can stop an activity stream using the console or AWS CLI.

If you delete your DB cluster, the activity stream is stopped and the underlying Amazon Kinesis stream is deleted automatically.

Console

To turn off an activity stream

1. Open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. In the navigation pane, choose **Databases**.
3. Choose a DB cluster that you want to stop the database activity stream for.
4. For **Actions**, choose **Stop activity stream**. The **Database Activity Stream** window appears.
 - a. Choose **Immediately**.

When you choose **Immediately**, the DB cluster restarts right away. If you choose **During the next maintenance window**, the DB cluster doesn't restart right away. In this case, the database activity stream doesn't stop until the next maintenance window.

- b. Choose **Continue**.

AWS CLI

To stop database activity streams for your DB cluster, configure the DB cluster using the AWS CLI command `stop-activity-stream`. Identify the AWS Region for the DB cluster using the `--region` parameter. The `--apply-immediately` parameter is optional.

For Linux, macOS, or Unix:

```
aws rds --region MY_REGION \
  stop-activity-stream \
  --resource-arn MY_CLUSTER_ARN \
  --apply-immediately
```

For Windows:

```
aws rds --region MY_REGION ^
  stop-activity-stream ^
  --resource-arn MY_CLUSTER_ARN ^
  --apply-immediately
```

RDS API

To stop database activity streams for your DB cluster, configure the cluster using the `StopActivityStream` operation. Identify the AWS Region for the DB cluster using the `Region` parameter. The `ApplyImmediately` parameter is optional.

Monitoring database activity streams

Database activity streams monitor and report activities. The stream of activity is collected and transmitted to Amazon Kinesis. From Kinesis, you can monitor the activity stream, or other services and applications can consume the activity stream for further analysis. You can find the underlying Kinesis stream name by using the AWS CLI command `describe-db-clusters` or the RDS API `DescribeDBClusters` operation.

Aurora manages the Kinesis stream for you as follows:

- Aurora creates the Kinesis stream automatically with a 24-hour retention period.

- Aurora scales the Kinesis stream if necessary.
- If you stop the database activity stream or delete the DB cluster, Aurora deletes the Kinesis stream.

The following categories of activity are monitored and put in the activity stream audit log:

- **SQL commands** – All SQL commands are audited, and also prepared statements, built-in functions, and functions in PL/SQL. Calls to stored procedures are audited. Any SQL statements issued inside stored procedures or functions are also audited.
- **Other database information** – Activity monitored includes the full SQL statement, the row count of affected rows from DML commands, accessed objects, and the unique database name. For Aurora PostgreSQL, database activity streams also monitor the bind variables and stored procedure parameters.

Important

The full SQL text of each statement is visible in the activity stream audit log, including any sensitive data. However, database user passwords are redacted if Aurora can determine them from the context, such as in the following SQL statement.

```
ALTER ROLE role-name WITH password
```

- **Connection information** – Activity monitored includes session and network information, the server process ID, and exit codes.

If an activity stream has a failure while monitoring your DB instance, you are notified through RDS events.

Topics

- [Accessing an activity stream from Kinesis \(p. 722\)](#)
- [Audit log contents and examples \(p. 723\)](#)
- [Processing a database activity stream using the AWS SDK \(p. 737\)](#)

Accessing an activity stream from Kinesis

When you enable an activity stream for a DB cluster, a Kinesis stream is created for you. From Kinesis, you can monitor your database activity in real time. To further analyze database activity, you can connect your Kinesis stream to consumer applications. You can also connect the stream to compliance management applications such as IBM's Security Guardium or Imperva's SecureSphere Database Audit and Protection.

To access an activity stream from Kinesis

1. Open the Kinesis console at <https://console.aws.amazon.com/kinesis>.
2. Choose your activity stream from the list of Kinesis streams.

An activity stream's name includes the prefix `aws-rds-das-cluster-` followed by the resource ID of the DB cluster. The following is an example.

```
aws-rds-das-cluster-NHVOV4PCLWHGF52NP
```

To use the Amazon RDS console to find the resource ID for the DB cluster, choose your DB cluster from the list of databases, and then choose the **Configuration** tab.

To use the AWS CLI to find the full Kinesis stream name for an activity stream, use a `describe-db-clusters` CLI request and note the value of `ActivityStreamKinesisStreamName` in the response.

3. Choose **Monitoring** to begin observing the database activity.

For more information about using Amazon Kinesis, see [What Is Amazon Kinesis Data Streams?](#).

Audit log contents and examples

Monitored events are represented in the database activity stream as JSON strings. The structure consists of a JSON object containing a `DatabaseActivityMonitoringRecord`, which in turn contains a `databaseActivityEventList` array of activity events.

Topics

- [Examples of an audit log for an activity stream \(p. 723\)](#)
- [DatabaseActivityMonitoringRecords JSON object \(p. 729\)](#)
- [databaseActivityEvents JSON Object \(p. 729\)](#)
- [databaseActivityEventList JSON array \(p. 730\)](#)

Examples of an audit log for an activity stream

Following are sample decrypted JSON audit logs of activity event records.

Example Activity event record of an Aurora PostgreSQL CONNECT SQL statement

Following is an activity event record of a login with the use of a CONNECT SQL statement (`command`) by a psql client (`clientApplication`).

```
{  
  "type": "DatabaseActivityMonitoringRecords",  
  "version": "1.1",  
  "databaseActivityEvents":  
  {  
    "type": "DatabaseActivityMonitoringRecord",  
    "clusterId": "cluster-4HNY5V4RRNPKKYB7ICFKE5JBQQ",  
    "instanceId": "db-FZJTMYKCXQBUUZ6VLU7NW3ITCM",  
    "databaseActivityEventList": [  
      {  
        "startTime": "2019-10-30 00:39:49.940668+00",  
        "logTime": "2019-10-30 00:39:49.990579+00",  
        "statementId": 1,  
        "substatementId": 1,  
        "objectType": null,  
        "command": "CONNECT",  
        "objectName": null,  
        "databaseName": "postgres",  
        "dbUserName": "rdsadmin",  
        "remoteHost": "172.31.3.195",  
        "remotePort": "49804",  
        "sessionId": "5ce5f7f0.474b",  
        "rowCount": null,  
        "commandText": null,  
        "paramList": [],  
        "pid": 18251,  
        "clientApplication": "psql",  
        "exitCode": null,  
        "class": "MISC",  
        "serverVersion": "2.3.1",  
        "serverType": "PostgreSQL",  
        "serviceName": "Amazon Aurora PostgreSQL-Compatible edition",  
        "serverHost": "172.31.3.192",  
        "serverPort": 5432  
      }  
    ]  
  }  
}
```

```

        "netProtocol": "TCP",
        "dbProtocol": "Postgres 3.0",
        "type": "record",
        "errorMessage": null
    }
]
},
"key": "decryption-key"
}

```

Example Activity event record of an Aurora MySQL CONNECT SQL statement

Following is an activity event record of a logon with the use of a CONNECT SQL statement (command) by a mysql client (clientApplication).

```

{
  "type": "DatabaseActivityMonitoringRecord",
  "clusterId": "cluster-some_id",
  "instanceId": "db-some_id",
  "databaseActivityEventList": [
    {
      "logTime": "2020-05-22 18:07:13.267214+00",
      "type": "record",
      "clientApplication": null,
      "pid": 2830,
      "dbUserName": "rdsadmin",
      "databaseName": "",
      "remoteHost": "localhost",
      "remotePort": "11053",
      "command": "CONNECT",
      "commandText": "",
      "paramList": null,
      "objectType": "TABLE",
      "objectName": "",
      "statementId": 0,
      "substatementId": 1,
      "exitCode": "0",
      "sessionId": "725121",
      "rowCount": 0,
      "serverHost": "master",
      "serverType": "MySQL",
      "serviceName": "Amazon Aurora MySQL",
      "serverVersion": "MySQL 5.7.12",
      "startTime": "2020-05-22 18:07:13.267207+00",
      "endTime": "2020-05-22 18:07:13.267213+00",
      "transactionId": "0",
      "dbProtocol": "MySQL",
      "netProtocol": "TCP",
      "errorMessage": "",
      "class": "MAIN"
    }
  ]
}

```

Example Activity event record of an Aurora PostgreSQL CREATE TABLE statement

Following is an example of a CREATE TABLE event for Aurora PostgreSQL.

```

{
  "type": "DatabaseActivityMonitoringRecords",
  "version": "1.1",
  "databaseActivityEvents": 

```

```
{
    "type": "DatabaseActivityMonitoringRecord",
    "clusterId": "cluster-4HNY5V4RRNPKKYB7ICFKE5JBQQ",
    "instanceId": "db-FZJTMYKCXQBUUZ6VLU7NW3ITCM",
    "databaseActivityEventList": [
        {
            "startTime": "2019-05-24 00:36:54.403455+00",
            "logTime": "2019-05-24 00:36:54.494235+00",
            "statementId": 2,
            "substatementId": 1,
            "objectType": null,
            "command": "CREATE TABLE",
            "objectName": null,
            "databaseName": "postgres",
            "dbUserName": "rdsadmin",
            "remoteHost": "172.31.3.195",
            "remotePort": "34534",
            "sessionId": "5ce73c6f.7e64",
            "rowCount": null,
            "commandText": "create table my_table (id serial primary key, name
varchar(32));",
            "paramList": [],
            "pid": 32356,
            "clientApplication": "psql",
            "exitCode": null,
            "class": "DDL",
            "serverVersion": "2.3.1",
            "serverType": "PostgreSQL",
            "serviceName": "Amazon Aurora PostgreSQL-Compatible edition",
            "serverHost": "172.31.3.192",
            "netProtocol": "TCP",
            "dbProtocol": "Postgres 3.0",
            "type": "record",
            "errorMessage": null
        }
    ],
    "key": "decryption-key"
}
```

Example Activity event record of an Aurora MySQL CREATE TABLE statement

Following is an example of a `CREATE TABLE` statement for Aurora MySQL. The operation is represented as two separate event records. One event has `"class": "MAIN"`. The other event has `"class": "AUX"`. The messages might arrive in any order. The `logTime` field of the `MAIN` event is always earlier than the `logTime` fields of any corresponding `AUX` events.

The following example shows the event with a `class` value of `MAIN`.

```
{
    "type": "DatabaseActivityMonitoringRecord",
    "clusterId": "cluster-some_id",
    "instanceId": "db-some_id",
    "databaseActivityEventList": [
        {
            "logTime": "2020-05-22 18:07:12.250221+00",
            "type": "record",
            "clientApplication": null,
            "pid": 2830,
            "dbUserName": "master",
            "databaseName": "test",
            "remoteHost": "localhost",
            "remotePort": "11054",
            "command": "QUERY",
            "sqlText": "CREATE TABLE test_table (id INT PRIMARY KEY, name VARCHAR(255))"
        }
    ]
}
```

```

    "commandText":"CREATE TABLE test1 (id INT)",
    "paramList":null,
    "objectType":"TABLE",
    "objectName":"test1",
    "statementId":65459278,
    "substatementId":1,
    "exitCode":"0",
    "sessionId":"725118",
    "rowCount":0,
    "serverHost":"master",
    "serverType":"MySQL",
    "serviceName":"Amazon Aurora MySQL",
    "serverVersion":"MySQL 5.7.12",
    "startTime":"2020-05-22 18:07:12.226384+00",
    "endTime":"2020-05-22 18:07:12.250222+00",
    "transactionId":"0",
    "dbProtocol":"MySQL",
    "netProtocol":"TCP",
    "errorMessage":"",
    "class":"MAIN"
}
]
}

```

The following example shows the corresponding event with a `class` value of `AUX`.

```

{
  "type":"DatabaseActivityMonitoringRecord",
  "clusterId":"cluster-some_id",
  "instanceId":"db-some_id",
  "databaseActivityEventList":[
    {
      "logTime":"2020-05-22 18:07:12.247182+00",
      "type":"record",
      "clientApplication":null,
      "pid":2830,
      "dbUserName":"master",
      "databaseName":"test",
      "remoteHost":"localhost",
      "remotePort":"11054",
      "command":"CREATE",
      "commandText":"test1",
      "paramList":null,
      "objectType":"TABLE",
      "objectName":"test1",
      "statementId":65459278,
      "substatementId":2,
      "exitCode":"",
      "sessionId":"725118",
      "rowCount":0,
      "serverHost":"master",
      "serverType":"MySQL",
      "serviceName":"Amazon Aurora MySQL",
      "serverVersion":"MySQL 5.7.12",
      "startTime":"2020-05-22 18:07:12.226384+00",
      "endTime":"2020-05-22 18:07:12.247182+00",
      "transactionId":"0",
      "dbProtocol":"MySQL",
      "netProtocol":"TCP",
      "errorMessage":"",
      "class":"AUX"
    }
  ]
}

```

Example Activity event record of an Aurora PostgreSQL SELECT statement

Following is an example of a SELECT event.

```
{
  "type": "DatabaseActivityMonitoringRecords",
  "version": "1.1",
  "databaseActivityEvents":
  {
    "type": "DatabaseActivityMonitoringRecord",
    "clusterId": "cluster-4HNY5V4RRNPKKYB7ICFKE5JBQQ",
    "instanceId": "db-FZJTMYKCXQBUUZ6VLU7NW3ITCM",
    "databaseActivityEventList": [
      {
        "startTime": "2019-05-24 00:39:49.920564+00",
        "logTime": "2019-05-24 00:39:49.940668+00",
        "statementId": 6,
        "substatementId": 1,
        "objectType": "TABLE",
        "command": "SELECT",
        "objectName": "public.my_table",
        "databaseName": "postgres",
        "dbUserName": "rdsadmin",
        "remoteHost": "172.31.3.195",
        "remotePort": "34534",
        "sessionId": "5ce73c6f.7e64",
        "rowCount": 10,
        "commandText": "select * from my_table;",
        "paramList": [],
        "pid": 32356,
        "clientApplication": "psql",
        "exitCode": null,
        "class": "READ",
        "serverVersion": "2.3.1",
        "serverType": "PostgreSQL",
        "serviceName": "Amazon Aurora PostgreSQL-Compatible edition",
        "serverHost": "172.31.3.192",
        "netProtocol": "TCP",
        "dbProtocol": "Postgres 3.0",
        "type": "record",
        "errorMessage": null
      }
    ]
  },
  "key": "decryption-key"
}
```

Example Activity event record of an Aurora MySQL SELECT statement

Following is an example of a SELECT event.

The following example shows the event with a class value of MAIN.

```
{
  "type": "DatabaseActivityMonitoringRecord",
  "clusterId": "cluster-some_id",
  "instanceId": "db-some_id",
  "databaseActivityEventList": [
    {
      "logTime": "2020-05-22 18:29:57.986467+00",
      "type": "record",
      "clientApplication": null,
      "pid": 2830,
```

```

        "dbUserName": "master",
        "databaseName": "test",
        "remoteHost": "localhost",
        "remotePort": "11054",
        "command": "QUERY",
        "commandText": "SELECT * FROM test1 WHERE id < 28",
        "paramList": null,
        "objectType": "TABLE",
        "objectName": "test1",
        "statementId": 65469218,
        "substatementId": 1,
        "exitCode": "0",
        "sessionId": "726571",
        "rowCount": 2,
        "serverHost": "master",
        "serverType": "MySQL",
        "serviceName": "Amazon Aurora MySQL",
        "serverVersion": "MySQL 5.7.12",
        "startTime": "2020-05-22 18:29:57.986364+00",
        "endTime": "2020-05-22 18:29:57.986467+00",
        "transactionId": "0",
        "dbProtocol": "MySQL",
        "netProtocol": "TCP",
        "errorMessage": "",
        "class": "MAIN"
    }
]
}

```

The following example shows the corresponding event with a `class` value of `AUX`.

```
{
    "type": "DatabaseActivityMonitoringRecord",
    "instanceId": "db-some_id",
    "databaseActivityEventList": [
        {
            "logTime": "2020-05-22 18:29:57.986399+00",
            "type": "record",
            "clientApplication": null,
            "pid": 2830,
            "dbUserName": "master",
            "databaseName": "test",
            "remoteHost": "localhost",
            "remotePort": "11054",
            "command": "READ",
            "commandText": "test1",
            "paramList": null,
            "objectType": "TABLE",
            "objectName": "test1",
            "statementId": 65469218,
            "substatementId": 2,
            "exitCode": "",
            "sessionId": "726571",
            "rowCount": 0,
            "serverHost": "master",
            "serverType": "MySQL",
            "serviceName": "Amazon Aurora MySQL",
            "serverVersion": "MySQL 5.7.12",
            "startTime": "2020-05-22 18:29:57.986364+00",
            "endTime": "2020-05-22 18:29:57.986399+00",
            "transactionId": "0",
            "dbProtocol": "MySQL",
            "netProtocol": "TCP",
            "errorMessage": "",
            "class": "AUX"
        }
    ]
}
```

```
    ]
}
```

DatabaseActivityMonitoringRecords JSON object

The database activity event records are in a JSON object that contains the following information.

JSON Field	Data Type	Description
<code>type</code>	string	The type of JSON record. The value is <code>DatabaseActivityMonitoringRecords</code> .
<code>version</code>	string	<p>The version of the database activity monitoring records.</p> <p>The version of the generated database activity records depends on the engine version of the DB cluster:</p> <ul style="list-style-type: none"> Version 1.1 database activity records are generated for Aurora PostgreSQL DB clusters running the engine versions 10.10 and later minor versions and engine versions 11.5 and later. Version 1.0 database activity records are generated for Aurora PostgreSQL DB clusters running the engine versions 10.7 and 11.4. <p>All of the following fields are in both version 1.0 and version 1.1 except where specifically noted.</p>
<code>databaseActivityEvents</code> <small>(String)</small>	string	A JSON object containing the activity events.
<code>key</code>	string	An encryption key you use to decrypt the databaseActivityEventList (p. 730) <code>databaseActivityEventList</code> JSON array.

databaseActivityEvents JSON Object

The `databaseActivityEvents` JSON object contains the following information.

Top-level fields in JSON record

Each event in the audit log is wrapped inside a record in JSON format. This record contains the following fields.

`type`

This field always has the value `DatabaseActivityMonitoringRecords`.

`version`

This field represents the version of the database activity stream data protocol or contract. It defines which fields are available.

Version 1.0 represents the original data activity streams support for Aurora PostgreSQL versions 10.7 and 11.4. Version 1.1 represents the data activity streams support for Aurora PostgreSQL versions 10.10 and higher and Aurora PostgreSQL 11.5 and higher. Version 1.1 includes the additional fields `errorMessage` and `startTime`. Version 1.2 represents the data activity streams

support for Aurora MySQL 2.08 and higher. Version 1.2 includes the additional fields `endTime` and `transactionId`.

databaseActivityEvents

An encrypted string representing one or more activity events. It's represented as a base64 byte array. When you decrypt the string, the result is a record in JSON format with fields as shown in the examples in this section.

key

The encrypted data key used to encrypt the `databaseActivityEvents` string. This is the same AWS KMS key that you provided when you started the database activity stream.

The following example shows the format of this record.

```
{
  "type": "DatabaseActivityMonitoringRecords",
  "version": "1.1",
  "databaseActivityEvents": "encrypted audit records",
  "key": "encrypted key"
}
```

Take the following steps to decrypt the contents of the `databaseActivityEvents` field:

1. Decrypt the value in the `key` JSON field using the KMS key you provided when starting database activity stream. Doing so returns the data encryption key in clear text.
2. Base64-decode the value in the `databaseActivityEvents` JSON field to obtain the ciphertext, in binary format, of the audit payload.
3. Decrypt the binary ciphertext with the data encryption key that you decoded in the first step.
4. Decompress the decrypted payload.
 - The encrypted payload is in the `databaseActivityEvents` field.
 - The `databaseActivityEventList` field contains an array of audit records. The `type` fields in the array can be `record` or `heartbeat`.

The audit log activity event record is a JSON object that contains the following information.

JSON Field	Data Type	Description
<code>type</code>	string	The type of JSON record. The value is <code>DatabaseActivityMonitoringRecord</code> .
<code>clusterId</code>	string	The DB cluster resource identifier. It corresponds to the DB cluster attribute <code>DbClusterResourceId</code> .
<code>instanceId</code>	string	The DB instance resource identifier. It corresponds to the DB instance attribute <code>DbiResourceId</code> .
<code>databaseActivityEventList</code> <small>(String)</small>		An array of activity audit records or heartbeat messages.

databaseActivityEventList JSON array

The audit log payload is an encrypted `databaseActivityEventList` JSON array. The following tables lists alphabetically the fields for each activity event in the decrypted `DatabaseActivityEventList` array of an audit log. The fields differ depending on whether you use Aurora PostgreSQL or Aurora MySQL. Consult the table that applies to your database engine.

Important

The event structure is subject to change. Aurora might add new fields to activity events in the future. In applications that parse the JSON data, make sure that your code can ignore or take appropriate actions for unknown field names.

databaseActivityEventList fields for Aurora PostgreSQL

Field	Data Type	Description
class	string	<p>The class of activity event. Valid values for Aurora PostgreSQL are the following:</p> <ul style="list-style-type: none"> • ALL • CONNECT – A connect or disconnect event. • DDL – A DDL statement that is not included in the list of statements for the ROLE class. • FUNCTION – A function call or a DO block. • MISC – A miscellaneous command such as DISCARD, FETCH, CHECKPOINT, or VACUUM. • NONE • READ – A SELECT or COPY statement when the source is a relation or a query. • ROLE – A statement related to roles and privileges including GRANT, REVOKE, and CREATE/ALTER/DROP ROLE. • WRITE – An INSERT, UPDATE, DELETE, TRUNCATE, or COPY statement when the destination is a relation.
clientApplication	string	The application the client used to connect as reported by the client. The client doesn't have to provide this information, so the value can be null.
command	string	The name of the SQL command without any command details.
commandText	string	<p>The actual SQL statement passed in by the user. For Aurora PostgreSQL, the value is identical to the original SQL statement. This field is used for all types of records except for connect or disconnect records, in which case the value is null.</p> <p>Important The full SQL text of each statement is visible in the activity stream audit log, including any sensitive data. However, database user passwords are redacted if Aurora can determine them from the context, such as in the following SQL statement.</p> <div style="border: 1px solid black; padding: 5px; margin-top: 10px;"> <pre>ALTER ROLE role-name WITH password</pre> </div>
databaseName	string	The database to which the user connected.
dbProtocol	string	The database protocol, for example Postgres 3.0.
dbUserName	string	The database user with which the client authenticated.
errorMessage	string	If there was any error, this field is populated with the error message that would've been generated by the DB server. The errorMessage value is null for normal statements that didn't result in an error.

Field	Data Type	Description
(version 1.1 database activity records only)		<p>An error is defined as any activity that would produce a client-visible PostgreSQL error log event at a severity level of <code>ERROR</code> or greater. For more information, see PostgreSQL Message Severity Levels. For example, syntax errors and query cancellations generate an error message.</p> <p>Internal PostgreSQL server errors such as background checkpointer process errors do not generate an error message. However, records for such events are still emitted regardless of the setting of the log severity level. This prevents attackers from turning off logging to attempt avoiding detection.</p> <p>See also the <code>exitCode</code> field.</p>
<code>exitCode</code>	int	<p>A value used for a session exit record. On a clean exit, this contains the exit code. An exit code can't always be obtained in some failure scenarios. Examples are if PostgreSQL does an <code>exit()</code> or if an operator performs a command such as <code>kill -9</code>.</p> <p>If there was any error, the <code>exitCode</code> field shows the SQL error code, <code>SQLSTATE</code>, as listed in PostgreSQL Error Codes.</p> <p>See also the <code>errorMessage</code> field.</p>
<code>logTime</code>	string	A timestamp as recorded in the auditing code path. This represents the SQL statement execution end time. See also the <code>startTime</code> field.
<code>netProtocol</code>	string	The network communication protocol.
<code>objectName</code>	string	The name of the database object if the SQL statement is operating on one. This field is used only where the SQL statement operates on a database object. If the SQL statement is not operating on an object, this value is null.
<code>objectType</code>	string	<p>The database object type such as table, index, view, and so on. This field is used only where the SQL statement operates on a database object. If the SQL statement is not operating on an object, this value is null. Valid values include the following:</p> <ul style="list-style-type: none"> • <code>COMPOSITE TYPE</code> • <code>FOREIGN TABLE</code> • <code>FUNCTION</code> • <code>INDEX</code> • <code>MATERIALIZED VIEW</code> • <code>SEQUENCE</code> • <code>TABLE</code> • <code>TOAST TABLE</code> • <code>VIEW</code> • <code>UNKNOWN</code>
<code>paramList</code>	string	An array of comma-separated parameters passed to the SQL statement. If the SQL statement has no parameters, this value is an empty array.

Field	Data Type	Description
pid	int	The process ID of the backend process that is allocated for serving the client connection.
remoteHost	string	Either the client IP address or hostname. For Aurora PostgreSQL, which one is used depends on the database's <code>log_hostname</code> parameter setting.
remotePort	string	The client port number.
rowCount	int	The number of rows returned by the SQL statement. For example, if a SELECT statement returns 10 rows, rowCount is 10. For INSERT or UPDATE statements, rowCount is 0.
serverHost	string	The database server host IP address.
serverType	string	The database server type, for example <code>PostgreSQL</code> .
serverVersion	string	The database server version, for example <code>2.3.1</code> for Aurora PostgreSQL.
serviceName	string	The name of the service, for example <code>Amazon Aurora PostgreSQL-Compatible edition</code> .
sessionId	int	A pseudo-unique session identifier.
sessionID	int	A pseudo-unique session identifier.
startTime	string	The time when execution began for the SQL statement.
(version 1.1 database activity records only)		To calculate the approximate execution time of the SQL statement, use <code>logTime - startTime</code> . See also the <code>logTime</code> field.
statementId	int	An identifier for the client's SQL statement. The counter is at the session level and increments with each SQL statement entered by the client.
substatementId	int	An identifier for a SQL substatement. This value counts the contained substatements for each SQL statement identified by the <code>statementId</code> field.
type	string	The event type. Valid values are <code>record</code> or <code>heartbeat</code> .

databaseActivityEventList fields for Aurora MySQL

Field	Data Type	Description
class	string	<p>The class of activity event.</p> <p>Valid values for Aurora MySQL are the following:</p> <ul style="list-style-type: none"> • <code>MAIN</code> – The primary event representing a SQL statement. • <code>AUX</code> – A supplemental event containing additional details. For example, a statement that renames an object might have an event with class <code>AUX</code> that reflects the new name.

Field	Data Type	Description
		To find MAIN and AUX events corresponding to the same statement, check for different events that have the same values for the <code>pid</code> field and for the <code>statementId</code> field.
<code>clientApplication</code>	string	The application the client used to connect as reported by the client. The client doesn't have to provide this information, so the value can be null.
<code>command</code>	string	<p>The general category of the SQL statement. The values for this field depend on the value of <code>class</code>.</p> <p>The values when <code>class</code> is <code>MAIN</code> include the following:</p> <ul style="list-style-type: none"> • <code>CONNECT</code> – When a client session is connected. • <code>QUERY</code> – A SQL statement. Accompanied by one or more events with a <code>class</code> value of <code>AUX</code>. • <code>DISCONNECT</code> – When a client session is disconnected. • <code>FAILED_CONNECT</code> – When a client attempts to connect but isn't able to. • <code>CHANGEUSER</code> – A state change that's part of the MySQL network protocol, not from a statement that you issue. <p>The values when <code>class</code> is <code>AUX</code> include the following:</p> <ul style="list-style-type: none"> • <code>READ</code> – A <code>SELECT</code> or <code>COPY</code> statement when the source is a relation or a query. • <code>WRITE</code> – An <code>INSERT</code>, <code>UPDATE</code>, <code>DELETE</code>, <code>TRUNCATE</code>, or <code>COPY</code> statement when the destination is a relation. • <code>DROP</code> – Deleting an object. • <code>CREATE</code> – Creating an object. • <code>RENAME</code> – Renaming an object. • <code>ALTER</code> – Changing the properties of an object.

Field	Data Type	Description
commandText	string	<p>For events with a <code>class</code> value of <code>MAIN</code>, this field represents the actual SQL statement passed in by the user. This field is used for all types of records except for connect or disconnect records, in which case the value is null.</p> <p>For events with a <code>class</code> value of <code>AUX</code>, this field contains supplemental information about the objects involved in the event.</p> <p>For Aurora MySQL, characters such as quotation marks are preceded by a backslash, representing an escape character.</p> <p>Important The full SQL text of each statement is visible in the audit log, including any sensitive data. However, database user passwords are redacted if Aurora can determine them from the context, such as in the following SQL statement.</p> <div style="border: 1px solid black; padding: 5px; margin-top: 10px;"> <pre>mysql> SET PASSWORD = '<i>my-password</i>';</pre> </div>
databaseName	string	The database to which the user connected.
dbProtocol	string	The database protocol. Currently, this value is always <code>MySQL</code> for Aurora MySQL.
dbUserName	string	The database user with which the client authenticated.
endTime (version 1.2 database activity records only)	string	<p>The time when execution ended for the SQL statement. It is represented in Coordinated Universal Time (UTC) format.</p> <p>To calculate the execution time of the SQL statement, use <code>endTime - startTime</code>. See also the <code>startTime</code> field.</p>
errorMessage (version 1.1 database activity records only)	string	<p>If there was any error, this field is populated with the error message that would've been generated by the DB server. The <code>errorMessage</code> value is null for normal statements that didn't result in an error.</p> <p>An error is defined as any activity that would produce a client-visible MySQL error log event at a severity level of <code>ERROR</code> or greater. For more information, see The Error Log in the <i>MySQL Reference Manual</i>. For example, syntax errors and query cancellations generate an error message.</p> <p>Internal MySQL server errors such as background checkpoint process errors do not generate an error message. However, records for such events are still emitted regardless of the setting of the log severity level. This prevents attackers from turning off logging to attempt avoiding detection.</p> <p>See also the <code>exitCode</code> field.</p>

Field	Data Type	Description
exitCode	int	A value used for a session exit record. On a clean exit, this contains the exit code. An exit code can't always be obtained in some failure scenarios. In such cases, this value might be zero or might be blank.
logTime	string	A timestamp as recorded in the auditing code path. It is represented in Coordinated Universal Time (UTC) format. For the most accurate way to calculate statement duration, see the startTime and endTime fields.
netProtocol	string	The network communication protocol. Currently, this value is always TCP for Aurora MySQL.
objectName	string	The name of the database object if the SQL statement is operating on one. This field is used only where the SQL statement operates on a database object. If the SQL statement isn't operating on an object, this value is blank. To construct the fully qualified name of the object, combine databaseName and objectName. If the query involves multiple objects, this field can be a comma-separated list of names.
objectType	string	<p>The database object type such as table, index, and so on. This field is used only where the SQL statement operates on a database object. If the SQL statement is not operating on an object, this value is null.</p> <p>Valid values for Aurora MySQL include the following:</p> <ul style="list-style-type: none"> • INDEX • TABLE • UNKNOWN
paramList	string	This field isn't used for Aurora MySQL and is always null.
pid	int	The process ID of the backend process that is allocated for serving the client connection. When the database server is restarted, the pid changes and the counter for the statementId field starts over.
remoteHost	string	Either the IP address or hostname of the client that issued the SQL statement. For Aurora MySQL, which one is used depends on the database's skip_name_resolve parameter setting. The value localhost indicates activity from the rdsadmin special user.
remotePort	string	The client port number.
rowCount	int	The number of table rows affected or retrieved by the SQL statement. This field is used only for SQL statements that are data manipulation language (DML) statements. If the SQL statement is not a DML statement, this value is null.
serverHost	string	The database server instance identifier. This value is represented differently for Aurora MySQL than for Aurora PostgreSQL. Aurora PostgreSQL uses an IP address instead of an identifier.
serverType	string	The database server type, for example MySQL.

Field	Data Type	Description
serverVersion	string	The database server version. Currently, this value is always MySQL 5.7.12 for Aurora MySQL.
serviceName	string	The name of the service. Currently, this value is always Amazon Aurora MySQL for Aurora MySQL.
sessionId	int	A pseudo-unique session identifier.
startTime (version 1.1 database activity records only)	string	The time when execution began for the SQL statement. It is represented in Coordinated Universal Time (UTC) format. To calculate the execution time of the SQL statement, use endTime - startTime. See also the endTime field.
statementId	int	An identifier for the client's SQL statement. The counter increments with each SQL statement entered by the client. The counter is reset when the DB instance is restarted.
substatementId	int	An identifier for a SQL substatement. This value is 1 for events with class MAIN and 2 for events with class AUX. Use the statementId field to identify all the events generated by the same statement.
transactionId (version 1.2 database activity records only)	int	An identifier for a transaction.
type	string	The event type. Valid values are record or heartbeat.

Processing a database activity stream using the AWS SDK

You can programmatically process an activity stream by using the AWS SDK. The following are fully functioning Java and Python examples of how you might process the Kinesis data stream.

Java

```

import java.io.ByteArrayInputStream;
import java.io.ByteArrayOutputStream;
import java.io.IOException;
import java.net.InetAddress;
import java.nio.ByteBuffer;
import java.nio.charset.StandardCharsets;
import java.security.NoSuchAlgorithmException;
import java.security.NoSuchProviderException;
import java.security.Security;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.UUID;
import java.util.zip.GZIPInputStream;

import javax.crypto.Cipher;
import javax.crypto.NoSuchPaddingException;
import javax.crypto.spec.SecretKeySpec;

import com.amazonaws.auth.AWSStaticCredentialsProvider;

```

```

import com.amazonaws.auth.BasicAWSCredentials;
import com.amazonaws.encryptionsdk.AwsCrypto;
import com.amazonaws.encryptionsdk.CryptoInputStream;
import com.amazonaws.encryptionsdk.jce.JceMasterKey;
import com.amazonaws.services.kinesis.clientlibrary.exceptions.InvalidStateException;
import com.amazonaws.services.kinesis.clientlibrary.exceptions.ShutdownException;
import com.amazonaws.services.kinesis.clientlibrary.exceptions.ThrottlingException;
import com.amazonaws.services.kinesis.clientlibrary.interfaces.IRecordProcessor;
import
    com.amazonaws.services.kinesis.clientlibrary.interfaces.IRecordProcessorCheckpointer;
import
    com.amazonaws.services.kinesis.clientlibrary.interfaces.IRecordProcessorFactory;
import
    com.amazonaws.services.kinesis.clientlibrary.lib.worker.InitialPositionInStream;
import
    com.amazonaws.services.kinesis.clientlibrary.lib.worker.KinesisClientLibConfiguration;
import com.amazonaws.services.kinesis.clientlibrary.lib.worker.ShutdownReason;
import com.amazonaws.services.kinesis.clientlibrary.lib.worker.Worker;
import com.amazonaws.services.kinesis.clientlibrary.lib.worker.Worker.Builder;
import com.amazonaws.services.kinesis.model.Record;
import com.amazonaws.services.kms.AWSKMS;
import com.amazonaws.services.kms.AWSKMSClientBuilder;
import com.amazonaws.services.kms.model.DecryptRequest;
import com.amazonaws.services.kms.model.DecryptResult;
import com.amazonaws.util.Base64;
import com.amazonaws.util.IOUtils;
import com.google.gson.Gson;
import com.google.gson.GsonBuilder;
import com.google.gson.annotations.SerializedName;
import org.bouncycastle.jce.provider.BouncyCastleProvider;

public class DemoConsumer {

    private static final String STREAM_NAME = "aws-rds-das-[cluster-external-resource-
id]";
    private static final String APPLICATION_NAME = "AnyApplication"; //unique
    application name for dynamo table generation that holds kinesis shard tracking
    private static final String AWS_ACCESS_KEY = "[AWS_ACCESS_KEY_TO_ACCESS_KINESIS]";
    private static final String AWS_SECRET_KEY = "[AWS_SECRET_KEY_TO_ACCESS_KINESIS]";
    private static final String DBC_RESOURCE_ID = "[cluster-external-resource-id]";
    private static final String REGION_NAME = "[region-name]"; //us-east-1, us-
east-2...
    private static final BasicAWSCredentials CREDENTIALS = new
    BasicAWSCredentials(AWS_ACCESS_KEY, AWS_SECRET_KEY);
    private static final AWSStaticCredentialsProvider CREDENTIALS_PROVIDER = new
    AWSStaticCredentialsProvider(CREDENTIALS);

    private static final AwsCrypto CRYPTO = new AwsCrypto();
    private static final AWSKMS KMS = AWSKMSClientBuilder.standard()
        .withRegion(REGION_NAME)
        .withCredentials(CREDENTIALS_PROVIDER).build();

    class Activity {
        String type;
        String version;
        String databaseActivityEvents;
        String key;
    }

    class ActivityEvent {
        @SerializedName("class") String _class;
        String clientApplication;
        String command;
        String commandText;
        String databaseName;
        String dbProtocol;
    }
}

```

```

        String dbUserName;
        String endTime;
        String errorMessage;
        String exitCode;
        String logTime;
        String netProtocol;
        String objectName;
        String objectType;
        List<String> paramList;
        String pid;
        String remoteHost;
        String remotePort;
        String rowCount;
        String serverHost;
        String serverType;
        String serverVersion;
        String serviceName;
        String sessionId;
        String startTime;
        String statementId;
        String substatementId;
        String transactionId;
        String type;
    }

    class ActivityRecords {
        String type;
        String clusterId;
        String instanceId;
        List<ActivityEvent> databaseActivityEventList;
    }

    static class RecordProcessorFactory implements IRecordProcessorFactory {
        @Override
        public IRecordProcessor createProcessor() {
            return new RecordProcessor();
        }
    }

    static class RecordProcessor implements IRecordProcessor {

        private static final long BACKOFF_TIME_IN_MILLIS = 3000L;
        private static final int PROCESSING_RETRIES_MAX = 10;
        private static final long CHECKPOINT_INTERVAL_MILLIS = 60000L;
        private static final Gson GSON = new GsonBuilder().serializeNulls().create();

        private static final Cipher CIPHER;
        static {
            Security.insertProviderAt(new BouncyCastleProvider(), 1);
            try {
                CIPHER = Cipher.getInstance("AES/GCM/NoPadding", "BC");
            } catch (NoSuchAlgorithmException | NoSuchPaddingException |
NoSuchProviderException e) {
                throw new ExceptionInInitializerError(e);
            }
        }

        private long nextCheckpointTimeInMillis;

        @Override
        public void initialize(String shardId) {
        }

        @Override
        public void processRecords(final List<Record> records, final
IRecordProcessorCheckpointer checkpointer) {
    }
}

```

```

        for (final Record record : records) {
            processSingleBlob(record.getData());
        }

        if (System.currentTimeMillis() > nextCheckpointTimeInMillis) {
            checkpoint(checkpointer);
            nextCheckpointTimeInMillis = System.currentTimeMillis() +
CHECKPOINT_INTERVAL_MILLIS;
        }
    }

    @Override
    public void shutdown(IRecordProcessorCheckpointer checkpointer, ShutdownReason
reason) {
    if (reason == ShutdownReason.TERMINATE) {
        checkpoint(checkpointer);
    }
}

private void processSingleBlob(final ByteBuffer bytes) {
    try {
        // JSON $Activity
        final Activity activity = Gson.fromJson(new String(bytes.array()),
StandardCharsets.UTF_8), Activity.class);

        // Base64.Decode
        final byte[] decoded = Base64.decode(activity.databaseActivityEvents);
        final byte[] decodedDataKey = Base64.decode(activity.key);

        Map<String, String> context = new HashMap<>();
        context.put("aws:rds:dbc-id", DBC_RESOURCE_ID);

        // Decrypt
        final DecryptRequest decryptRequest = new DecryptRequest()

.withCiphertextBlob(ByteBuffer.wrap(decodedDataKey)).withEncryptionContext(context);
        final DecryptResult decryptResult = KMS.decrypt(decryptRequest);
        final byte[] decrypted = decrypt(decoded,
getBytes(decryptResult.getPlaintext()));

        // GZip Decompress
        final byte[] decompressed = decompress(decrypted);
        // JSON $ActivityRecords
        final ActivityRecords activityRecords = Gson.fromJson(new
String(decompressed, StandardCharsets.UTF_8), ActivityRecords.class);

        // Iterate through $ActivityEvents
        for (final ActivityEvent event :
activityRecords.databaseActivityEventList) {
            System.out.println(Gson.toJson(event));
        }
    } catch (Exception e) {
        // Handle error.
        e.printStackTrace();
    }
}

private static byte[] decompress(final byte[] src) throws IOException {
    ByteArrayInputStream byteArrayInputStream = new ByteArrayInputStream(src);
    GZIPInputStream gzipInputStream = new
GZIPInputStream(byteArrayInputStream);
    return IOUtils.toByteArray(gzipInputStream);
}

private void checkpoint(IRecordProcessorCheckpointer checkpointer) {
    for (int i = 0; i < PROCESSING_RETRIES_MAX; i++) {

```

```

        try {
            checkpointer.checkpoint();
            break;
        } catch (ShutdownException se) {
            // Ignore checkpoint if the processor instance has been shutdown
            (fail over).
            System.out.println("Caught shutdown exception, skipping
checkpoint." + se);
            break;
        } catch (ThrottlingException e) {
            // Backoff and re-attempt checkpoint upon transient failures
            if (i >= (PROCESSING_RETRIES_MAX - 1)) {
                System.out.println("Checkpoint failed after " + (i + 1) +
"attempts." + e);
                break;
            } else {
                System.out.println("Transient issue when checkpointing -
attempt " + (i + 1) + " of " + PROCESSING_RETRIES_MAX + e);
            }
        } catch (InvalidStateException e) {
            // This indicates an issue with the DynamoDB table (check for
table, provisioned IOPS).
            System.out.println("Cannot save checkpoint to the DynamoDB table
used by the Amazon Kinesis Client Library." + e);
            break;
        }
        try {
            Thread.sleep(BACKOFF_TIME_IN_MILLIS);
        } catch (InterruptedException e) {
            System.out.println("Interrupted sleep" + e);
        }
    }
}

private static byte[] decrypt(final byte[] decoded, final byte[] decodedDataKey)
throws IOException {
    // Create a JCE master key provider using the random key and an AES-GCM
    encryption algorithm
    final JceMasterKey masterKey = JceMasterKey.getInstance(new
    SecretKeySpec(decodedDataKey, "AES"),
        "BC", "DataKey", "AES/GCM/NoPadding");
    try (final CryptoInputStream<JceMasterKey> decryptingStream =
CRYPTO.createDecryptingStream(masterKey, new ByteArrayInputStream(decoded));
        final ByteArrayOutputStream out = new ByteArrayOutputStream() {
            IOUtils.copy(decryptingStream, out);
            return out.toByteArray();
        }
    )
    public static void main(String[] args) throws Exception {
        final String workerId = InetAddress.getLocalHost().getCanonicalHostName() + ":" +
UUID.randomUUID();
        final KinesisClientLibConfiguration kinesisClientLibConfiguration =
            new KinesisClientLibConfiguration(APPLICATION_NAME, STREAM_NAME,
CREDENTIALS_PROVIDER, workerId);

        kinesisClientLibConfiguration.withInitialPositionInStream(InitialPositionInStream.LATEST);
        kinesisClientLibConfiguration.withRegionName(REGION_NAME);
        final Worker worker = new Builder()
            .recordProcessorFactory(new RecordProcessorFactory())
            .config(kinesisClientLibConfiguration)
            .build();

        System.out.printf("Running %s to process stream %s as worker %s...\n",
APPLICATION_NAME, STREAM_NAME, workerId);
    }
}

```

```

        try {
            worker.run();
        } catch (Throwable t) {
            System.err.println("Caught throwable while processing data.");
            t.printStackTrace();
            System.exit(1);
        }
        System.exit(0);
    }

    private static byte[] getByteArray(final ByteBuffer b) {
        byte[] byteArray = new byte[b.remaining()];
        b.get(byteArray);
        return byteArray;
    }
}

```

Python

```

import base64
import json
import zlib
import aws_encryption_sdk
from aws_encryption_sdk import CommitmentPolicy
from aws_encryption_sdk.internal.crypto import WrappingKey
from aws_encryption_sdk.key_providers.raw import RawMasterKeyProvider
from aws_encryption_sdk.identifiers import WrappingAlgorithm, EncryptionKeyType
import boto3

REGION_NAME = '<region>' # us-east-1
RESOURCE_ID = '<external-resource-id>' # cluster-ABCD123456
STREAM_NAME = 'aws-rds-das-' + RESOURCE_ID # aws-rds-das-cluster-ABCD123456

enc_client =
aws_encryption_sdk.EncryptionSDKClient(commitment_policy=CommitmentPolicy.REQUIRE_ENCRYPT_ALLOW_DECRYPT)

class MyRawMasterKeyProvider(RawMasterKeyProvider):
    provider_id = "BC"

    def __new__(cls, *args, **kwargs):
        obj = super(RawMasterKeyProvider, cls).__new__(cls)
        return obj

    def __init__(self, plain_key):
        RawMasterKeyProvider.__init__(self)
        self.wrapping_key =
WrappingKey(wrapping_algorithm=WrappingAlgorithm.AES_256_GCM_IV12_TAG16_NO_PADDING,
            wrapping_key=plain_key,
wrapping_key_type=EncryptionKeyType.SYMMETRIC)

    def _get_raw_key(self, key_id):
        return self.wrapping_key

def decrypt_payload(payload, data_key):
    my_key_provider = MyRawMasterKeyProvider(data_key)
    my_key_provider.add_master_key("DataKey")
    decrypted_plaintext, header = enc_client.decrypt(
        source=payload,
        materials_manager=aws_encryption_sdk.materials_managers.default.DefaultCryptoMaterialsManager(mast
            return decrypted_plaintext

```

```

def decrypt_decompress(payload, key):
    decrypted = decrypt_payload(payload, key)
    return zlib.decompress(decrypted, zlib.MAX_WBITS + 16)

def main():
    session = boto3.session.Session()
    kms = session.client('kms', region_name=REGION_NAME)
    kinesis = session.client('kinesis', region_name=REGION_NAME)

    response = kinesis.describe_stream(StreamName=STREAM_NAME)
    shard_iters = []
    for shard in response['StreamDescription']['Shards']:
        shard_iter_response = kinesis.get_shard_iterator(StreamName=STREAM_NAME,
                                                       ShardId=shard['ShardId'],
                                                       ShardIteratorType='LATEST')
        shard_iters.append(shard_iter_response['ShardIterator'])

    while len(shard_iters) > 0:
        next_shard_iters = []
        for shard_iter in shard_iters:
            response = kinesis.get_records(ShardIterator=shard_iter, Limit=10000)
            for record in response['Records']:
                record_data = record['Data']
                record_data = json.loads(record_data)
                payload_decoded =
                    base64.b64decode(record_data['databaseActivityEvents'])
                data_key_decoded = base64.b64decode(record_data['key'])
                data_key_decrypt_result = kms.decrypt(CiphertextBlob=data_key_decoded,
                                                       EncryptionContext={'aws:rds:dbc-
id': RESOURCE_ID})
                print decrypt_decompress(payload_decoded,
                                         data_key_decrypt_result['Plaintext']))
                if 'NextShardIterator' in response:
                    next_shard_iters.append(response['NextShardIterator'])
        shard_iters = next_shard_iters

if __name__ == '__main__':
    main()

```

Managing access to database activity streams

Any user with appropriate AWS Identity and Access Management (IAM) role privileges for database activity streams can create, start, stop, and modify the activity stream settings for a DB cluster. These actions are included in the audit log of the stream. For best compliance practices, we recommend that you don't provide these privileges to DBAs.

You set access to database activity streams using IAM policies. For more information about Aurora authentication, see [Identity and access management in Amazon Aurora \(p. 1724\)](#). For more information about creating IAM policies, see [Creating and using an IAM policy for IAM database access \(p. 1746\)](#).

Example Policy to allow configuring database activity streams

To give users fine-grained access to modify activity streams, use the service-specific operation context keys `rds:StartActivityStream` and `rds:StopActivityStream` in an IAM policy. The following IAM policy example allows a user or role to configure activity streams.

```
{
    "Version": "2012-10-17",
    "Statement": [

```

```
{  
    "Sid": "ConfigureActivityStreams",  
    "Effect": "Allow",  
    "Action": [  
        "rds:StartActivityStream",  
        "rds:StopActivityStream"  
    ],  
    "Resource": "*",  
}  
}  
]
```

Example Policy to allow starting database activity streams

The following IAM policy example allows a user or role to start activity streams.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "AllowStartActivityStreams",  
            "Effect": "Allow",  
            "Action": "rds:StartActivityStream",  
            "Resource": "*"  
        }  
    ]  
}
```

Example Policy to allow stopping database activity streams

The following IAM policy example allows a user or role to stop activity streams.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "AllowStopActivityStreams",  
            "Effect": "Allow",  
            "Action": "rds:StopActivityStream",  
            "Resource": "*"  
        }  
    ]  
}
```

Example Policy to deny starting database activity streams

The following IAM policy example prevents a user or role from starting activity streams.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "DenyStartActivityStreams",  
            "Effect": "Deny",  
            "Action": "rds:StartActivityStream",  
            "Resource": "*"  
        }  
    ]  
}
```

Example Policy to deny stopping database activity streams

The following IAM policy example prevents a user or role from stopping activity streams.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "DenyStopActivityStreams",  
            "Effect": "Deny",  
            "Action": "rds:StopActivityStream",  
            "Resource": "*"  
        }  
    ]  
}
```

Working with Amazon Aurora MySQL

Amazon Aurora MySQL is a fully managed, MySQL-compatible, relational database engine that combines the speed and reliability of high-end commercial databases with the simplicity and cost-effectiveness of open-source databases. Aurora MySQL is a drop-in replacement for MySQL and makes it simple and cost-effective to set up, operate, and scale your new and existing MySQL deployments, thus freeing you to focus on your business and applications. Amazon RDS provides administration for Aurora by handling routine database tasks such as provisioning, patching, backup, recovery, failure detection, and repair. Amazon RDS also provides push-button migration tools to convert your existing Amazon RDS for MySQL applications to Aurora MySQL.

Topics

- [Overview of Amazon Aurora MySQL \(p. 746\)](#)
- [Security with Amazon Aurora MySQL \(p. 774\)](#)
- [Updating applications to connect to Aurora MySQL DB clusters using new SSL/TLS certificates \(p. 778\)](#)
- [Migrating data to an Amazon Aurora MySQL DB cluster \(p. 781\)](#)
- [Managing Amazon Aurora MySQL \(p. 812\)](#)
- [Tuning Aurora MySQL with wait events and thread states \(p. 837\)](#)
- [Working with parallel query for Amazon Aurora MySQL \(p. 881\)](#)
- [Using Advanced Auditing with an Amazon Aurora MySQL DB cluster \(p. 914\)](#)
- [Single-master replication with Amazon Aurora MySQL \(p. 918\)](#)
- [Working with Aurora multi-master clusters \(p. 958\)](#)
- [Integrating Amazon Aurora MySQL with other AWS services \(p. 984\)](#)
- [Amazon Aurora MySQL lab mode \(p. 1032\)](#)
- [Best practices with Amazon Aurora MySQL \(p. 1033\)](#)
- [Amazon Aurora MySQL reference \(p. 1042\)](#)
- [Database engine updates for Amazon Aurora MySQL \(p. 1082\)](#)

Overview of Amazon Aurora MySQL

The following sections provide an overview of Amazon Aurora MySQL.

Topics

- [Amazon Aurora MySQL performance enhancements \(p. 746\)](#)
- [Amazon Aurora MySQL and spatial data \(p. 747\)](#)
- [Aurora MySQL version 3 compatible with MySQL 8.0 \(p. 748\)](#)
- [Aurora MySQL version 2 compatible with MySQL 5.7 \(p. 773\)](#)

Amazon Aurora MySQL performance enhancements

Amazon Aurora includes performance enhancements to support the diverse needs of high-end commercial databases.

Fast insert

Fast insert accelerates parallel inserts sorted by primary key and applies specifically to `LOAD DATA` and `INSERT INTO ... SELECT ...` statements. Fast insert caches the position of a cursor in an index traversal while executing the statement. This avoids unnecessarily traversing the index again.

You can monitor the following metrics to determine the effectiveness of fast insert for your DB cluster:

- `aurora_fast_insert_cache_hits`: A counter that is incremented when the cached cursor is successfully retrieved and verified.
- `aurora_fast_insert_cache_misses`: A counter that is incremented when the cached cursor is no longer valid and Aurora performs a normal index traversal.

You can retrieve the current value of the fast insert metrics using the following command:

```
mysql> show global status like 'Aurora_fast_insert%';
```

You will get output similar to the following:

Variable_name	Value
Aurora_fast_insert_cache_hits	3598300
Aurora_fast_insert_cache_misses	436401336

Amazon Aurora MySQL and spatial data

The following list summarizes the main Aurora MySQL spatial features and explains how they correspond to spatial features in MySQL:

- Aurora MySQL 1.x supports the same spatial data types and spatial relation functions as MySQL 5.6. For more information about these data types and functions, see [Spatial Data Types](#) and [Spatial Relation Functions](#) in the MySQL 5.6 documentation.
- Aurora MySQL 2.x supports the same spatial data types and spatial relation functions as MySQL 5.7. For more information about these data types and functions, see [Spatial Data Types](#) and [Spatial Relation Functions](#) in the MySQL 5.7 documentation.
- Aurora MySQL 3.x supports the same spatial data types and spatial relation functions as MySQL 8.0. For more information about these data types and functions, see [Spatial Data Types](#) and [Spatial Relation Functions](#) in the MySQL 8.0 documentation.
- Aurora MySQL supports spatial indexing on InnoDB tables. Spatial indexing improves query performance on large datasets for queries on spatial data. In MySQL, spatial indexing for InnoDB tables isn't available in MySQL 5.6, but is available in MySQL 5.7 and 8.0. Aurora MySQL uses a different spatial indexing strategy than MySQL for high performance with spatial queries. The Aurora spatial index implementation uses a space-filling curve on a B-tree, which is intended to provide higher performance for spatial range scans than an R-tree.

The following data definition language (DDL) statements are supported for creating indexes on columns that use spatial data types.

CREATE TABLE

You can use the `SPATIAL INDEX` keywords in a `CREATE TABLE` statement to add a spatial index to a column in a new table. Following is an example.

```
CREATE TABLE test (shape POLYGON NOT NULL, SPATIAL INDEX(shape));
```

ALTER TABLE

You can use the `SPATIAL INDEX` keywords in an `ALTER TABLE` statement to add a spatial index to a column in an existing table. Following is an example.

```
ALTER TABLE test ADD SPATIAL INDEX(shape);
```

CREATE INDEX

You can use the `SPATIAL` keyword in a `CREATE INDEX` statement to add a spatial index to a column in an existing table. Following is an example.

```
CREATE SPATIAL INDEX shape_index ON test (shape);
```

Aurora MySQL version 3 compatible with MySQL 8.0

You can use Aurora MySQL version 3 to get the latest MySQL-compatible features, performance enhancements, and bug fixes. Following, you can learn about Aurora MySQL version 3, with MySQL 8.0 compatibility. You can learn how to upgrade your clusters and applications to Aurora MySQL version 3.

Topics

- [Features from community MySQL 8.0 \(p. 748\)](#)
- [New parallel query optimizations \(p. 749\)](#)
- [Release notes for Aurora MySQL version 3 \(p. 749\)](#)
- [Comparison of Aurora MySQL version 2 and Aurora MySQL version 3 \(p. 749\)](#)
- [Comparison of Aurora MySQL version 3 and community MySQL 8.0 \(p. 756\)](#)
- [Upgrading to Aurora MySQL version 3 \(p. 760\)](#)

Features from community MySQL 8.0

The initial release of Aurora MySQL version 3 is compatible with community MySQL 8.0.23. MySQL 8.0 introduces several new features, including the following:

- JSON functions. For usage information, see [JSON Functions](#) in the *MySQL Reference Manual*.
- Window functions. For usage information, see [Window Functions](#) in the *MySQL Reference Manual*.
- Common table expressions (CTEs), using the `WITH` clause. For usage information, see [WITH \(Common Table Expressions\)](#) in the *MySQL Reference Manual*.
- Optimized `ADD COLUMN` and `RENAME COLUMN` clauses for the `ALTER TABLE` statement. These optimizations are called "instant DDL." Aurora MySQL version 3 is compatible with the community MySQL instant DDL feature. The former Aurora fast DDL feature isn't used. For usage information for instant DDL, see [Instant DDL \(Aurora MySQL version 3\) \(p. 832\)](#).
- Descending, functional, and invisible indexes. For usage information, see [Invisible Indexes, Descending Indexes](#), and [CREATE INDEX Statement](#) in the *MySQL Reference Manual*.

- Role-based privileges controlled through SQL statements. For more information on changes to the privilege model, see [Role-based privilege model \(p. 757\)](#).
- `NOWAIT` and `SKIP LOCKED` clauses with the `SELECT ... FOR SHARE` statement. These clauses avoid waiting for other transactions to release row locks. For usage information, see [Locking Reads](#) in the *MySQL Reference Manual*.
- Improvements to binary log (binlog) replication. For the Aurora MySQL details, see [Binary log replication \(p. 756\)](#). In particular, you can perform filtered replication. For usage information about filtered replication, see [How Servers Evaluate Replication Filtering Rules](#) in the *MySQL Reference Manual*.
- Hints. Some of the MySQL 8.0-compatible hints were already backported to Aurora MySQL version 2. For information about using hints with Aurora MySQL, see [Aurora MySQL hints \(p. 1074\)](#). For the full list of hints in community MySQL 8.0, see [Optimizer Hints](#) in the *MySQL Reference Manual*.

For the full list of features added to MySQL 8.0 community edition, see the blog post [The complete list of new features in MySQL 8.0](#).

Aurora MySQL version 3 also includes changes to keywords for inclusive language, backported from community MySQL 8.0.26. For details about those changes, see [Inclusive language changes for Aurora MySQL version 3 \(p. 751\)](#).

New parallel query optimizations

The Aurora parallel query optimization now applies to more SQL operations:

- Parallel query now applies to tables containing the data types `TEXT`, `BLOB`, `JSON`, `GEOMETRY`, and `VARCHAR` and `CHAR` longer than 768 bytes.
- Parallel query can optimize queries involving partitioned tables.
- Parallel query can optimize queries involving aggregate function calls in the select list and the `HAVING` clause.

For more information about these enhancements, see [Upgrading parallel query clusters to Aurora MySQL version 3 \(p. 893\)](#). For general information about Aurora parallel query, see [Working with parallel query for Amazon Aurora MySQL \(p. 881\)](#).

Release notes for Aurora MySQL version 3

For the release notes for all Aurora MySQL version 3 releases, see [Database engine updates for Amazon Aurora MySQL version 3 \(p. 1108\)](#).

Comparison of Aurora MySQL version 2 and Aurora MySQL version 3

Use the following to learn about changes to be aware of when you upgrade your Aurora MySQL version 2 cluster to version 3.

Topics

- [Feature differences between Aurora MySQL version 2 and 3 \(p. 750\)](#)
- [Instance class support \(p. 750\)](#)
- [Parameter changes for Aurora MySQL version 3 \(p. 751\)](#)
- [Status variables \(p. 751\)](#)
- [Inclusive language changes for Aurora MySQL version 3 \(p. 751\)](#)
- [AUTO_INCREMENT values \(p. 753\)](#)
- [Temporary tables on reader DB instances \(p. 754\)](#)

- [Storage engine for internal temporary tables \(p. 754\)](#)
- [Binary log replication \(p. 756\)](#)

Feature differences between Aurora MySQL version 2 and 3

The following Amazon Aurora MySQL features are supported in Aurora MySQL for MySQL 5.7, but these features aren't supported in Aurora MySQL for MySQL 8.0:

- Backtrack currently isn't available for Aurora MySQL version 3 clusters. We intend to make this feature available in a subsequent minor version.

If you have an Aurora MySQL version 2 cluster that uses backtrack, currently you can't use the snapshot restore method to upgrade to Aurora MySQL version 3. This limitation applies to all clusters that use backtrack clusters, regardless of whether the backtrack setting is turned on. For details about upgrade procedures, see [Upgrading to Aurora MySQL version 3 \(p. 760\)](#).

- You can't use Aurora MySQL version 3 for Aurora Serverless v1 clusters. Aurora MySQL version 3 works with Aurora Serverless v2, which is currently in preview.
- Lab mode doesn't apply to Aurora MySQL version 3. There aren't any lab mode features in Aurora MySQL version 3. Instant DDL supersedes the fast online DDL feature that was formerly available in lab mode. For an example, see [Instant DDL \(Aurora MySQL version 3\) \(p. 832\)](#).
- The query cache is removed from community MySQL 8.0 and also from Aurora MySQL version 3.
- Aurora MySQL version 3 is compatible with the community MySQL hash join feature. The Aurora-specific implementation of hash joins in Aurora MySQL version 2 isn't used. For information about using hash joins with Aurora parallel query, see [Turning on hash join for parallel query clusters \(p. 892\)](#) and [Aurora MySQL hints \(p. 1074\)](#). For general usage information about hash joins, see [Hash Join Optimization](#) in the *MySQL Reference Manual*.
- Currently, you can't restore a physical backup from the Percona XtraBackup tool to an Aurora MySQL version 3 cluster. We intend to support this feature in a subsequent minor version.
- The `mysql.lambda_async` stored procedure that was deprecated in Aurora MySQL version 2 is removed in version 3. For version 3, use the asynchronous function `lambda_async` instead.
- The default character set in Aurora MySQL version 3 is `utf8mb4`. In Aurora MySQL version 2, the default character set was `latin1`. For information about this character set, see [The utf8mb4 Character Set \(4-Byte UTF-8 Unicode Encoding\)](#) in the *MySQL Reference Manual*.
- The `innodb_flush_log_at_trx_commit` configuration parameter can't be modified. The default value of 1 always applies.

Some Aurora MySQL features are available for certain combinations of AWS Region and DB engine version. For details, see [Supported features in Amazon Aurora by AWS Region and Aurora DB engine \(p. 19\)](#).

Instance class support

Aurora MySQL version 3 supports a different set of instance classes than Aurora MySQL version 2 does:

- For larger instances, you can use the modern instance classes such as `db.r5`, `db.r6g`, and `db.x2g`.
- For smaller instances, you can use the modern instance classes such as `db.t3` and `db.t4g`.

The following instance classes from Aurora MySQL version 2 aren't available for Aurora MySQL version 3:

- `db.r4`
- `db.r3`
- `db.t3.small`
- `db.t2`

Check your administration scripts for any CLI statements that create Aurora MySQL DB instances and hardcode instance class names that aren't available for Aurora MySQL version 3. If necessary, modify the instance class names to ones that Aurora MySQL version 3 supports.

Tip

To check the instance classes that you can use for a specific combination of Aurora MySQL version and AWS Region, use the `describe-orderable-db-instance-options` AWS CLI command.

For full details about Aurora instance classes, see [Aurora DB instance classes \(p. 54\)](#).

Parameter changes for Aurora MySQL version 3

Aurora MySQL version 3 includes new cluster-level and instance-level configuration parameters. Aurora MySQL version 3 also removes some parameters that were present in Aurora MySQL version 2. Some parameter names are changed as a result of the initiative for inclusive language. For backward compatibility, you can still retrieve the parameter values using either the old names or the new names. However, you must use the new names to specify parameter values in a custom parameter group.

In Aurora MySQL version 3, the value of the `lower_case_table_names` parameter is set permanently at the time the cluster is created. If you use a nondefault value for this option, set up your Aurora MySQL version 3 custom parameter group before upgrading. Then specify the parameter group during the create cluster or snapshot restore operation.

For the full list of Aurora MySQL cluster parameters, see [Cluster-level parameters \(p. 1043\)](#). The table covers all the parameters from Aurora MySQL version 1, 2, and 3. The table includes notes showing which parameters are new in Aurora MySQL version 3 or were removed from Aurora MySQL version 3.

For the full list of Aurora MySQL instance parameters, see [Instance-level parameters \(p. 1049\)](#). The table covers all the parameters from Aurora MySQL version 1, 2, and 3. The table includes notes showing which parameters are new in Aurora MySQL version 3 and which parameters were removed from Aurora MySQL version 3. It also includes notes showing which parameters were modifiable in earlier versions but not Aurora MySQL version 3.

For information about parameter names that changed, see [Inclusive language changes for Aurora MySQL version 3 \(p. 751\)](#).

Status variables

For information about status variables that aren't applicable to Aurora MySQL, see [MySQL status variables that don't apply to Aurora MySQL \(p. 1062\)](#).

Inclusive language changes for Aurora MySQL version 3

Aurora MySQL version 3 is compatible with version 8.0.23 from the MySQL community edition. Aurora MySQL version 3 also includes changes from MySQL 8.0.26 related to keywords and system schemas for inclusive language. For example, the `SHOW REPLICA STATUS` command is now preferred instead of `SHOW SLAVE STATUS`.

The following Amazon CloudWatch metrics have new names in Aurora MySQL version 3.

In Aurora MySQL version 3, only the new metric names are available. Make sure to update any alarms or other automation that relies on metric names when you upgrade to Aurora MySQL version 3.

Old name	New name
<code>ForwardingMasterDMLLatency</code>	<code>ForwardingWriterDMLLatency</code>
<code>ForwardingMasterOpenSessions</code>	<code>ForwardingWriterOpenSessions</code>
<code>AuroraDMLRejectedMasterFull</code>	<code>AuroraDMLRejectedWriterFull</code>

Old name	New name
ForwardingMasterDMLThroughput	ForwardingWriterDMLThroughput

The following status variables have new names in Aurora MySQL version 3.

For compatibility, you can use either name in the initial Aurora MySQL version 3 release. The old status variable names are to be removed in a future release.

Name to be removed	New or preferred name
Aurora_fwd_master_dml_stmt	Aurora_fwd_writer_dml_stmt_duration
Aurora_fwd_master_dml_stmt	Aurora_fwd_writer_dml_stmt_count
Aurora_fwd_master_select_stmt	Aurora_fwd_writer_select_stmt_duration
Aurora_fwd_master_select_stmt	Aurora_fwd_writer_select_stmt_count
Aurora_fwd_master_errors_session_timeout	Aurora_writer_errors_session_timeout
Aurora_fwd_master_open_sessions	Aurora_fwd_writer_open_sessions
Aurora_fwd_master_errors_sessions_limit	Aurora_fwd_writer_errors_session_limit
Aurora_fwd_master_errors_rpc_timeout	Aurora_fwd_writer_errors_rpc_timeout

The following configuration parameters have new names in Aurora MySQL version 3.

For compatibility, you can check the parameter values in the `mysql` client by using either name in the initial Aurora MySQL version 3 release. You can only modify the values in a custom parameter group by using the new names. The old parameter names are to be removed in a future release.

Name to be removed	New or preferred name
aurora_fwd_master_idle_timeout	Aurora_fwd_writer_idle_timeout
aurora_fwd_master_max_connections	Aurora_fwd_writer_max_connections_pct
master_verify_checksum	source_verify_checksum
sync_master_info	sync_source_info
init_slave	init_replica
rpl_stop_slave_timeout	rpl_stop_replica_timeout
log_slow_slave_statements	log_slow_replica_statements
slave_max_allowed_packet	replica_max_allowed_packet
slave_compressed_protocol	replica_compressed_protocol
slave_exec_mode	replica_exec_mode
slave_type_conversions	replica_type_conversions
slave_sql_verify_checksum	replica_sql_verify_checksum

Name to be removed	New or preferred name
slave_parallel_type	replica_parallel_type
slave_preserve_commit_order	replica_preserve_commit_order
log_slave_updates	log_replica_updates
slave_allow_batching	replica_allow_batching
slave_load_tmpdir	replica_load_tmpdir
slave_net_timeout	replica_net_timeout
sql_slave_skip_counter	sql_replica_skip_counter
slave_skip_errors	replica_skip_errors
slave_checkpoint_period	replica_checkpoint_period
slave_checkpoint_group	replica_checkpoint_group
slave_transaction_retries	replica_transaction_retries
slave_parallel_workers	replica_parallel_workers
slave_pending_jobs_size_max	replica_pending_jobs_size_max
pseudo_slave_mode	pseudo_replica_mode

The following stored procedures have new names in Aurora MySQL version 3.

For compatibility, you can use either name in the initial Aurora MySQL version 3 release. The old procedure names are to be removed in a future release.

Name to be removed	New or preferred name
mysql.rds_set_master_auto_position	mysql.rds_set_source_auto_position
mysql.rds_set_external_master	mysql.rds_set_external_source
mysql.rds_set_external_master	mysql.rds_set_external_source_with_auto_position
mysql.rds_reset_external_master	mysql.rds_reset_external_source
mysql.rds_next_master_log	mysql.rds_next_source_log

AUTO_INCREMENT values

In Aurora MySQL version 3, Aurora preserves the AUTO_INCREMENT value for each table when it restarts each DB instance. In Aurora MySQL version 2, the AUTO_INCREMENT value wasn't preserved after a restart.

The AUTO_INCREMENT value isn't preserved when you set up a new cluster by restoring from a snapshot, performing a point-in-time recovery, and cloning a cluster. In these cases, the AUTO_INCREMENT value is initialized to the value based on the largest column value in the table at the time the snapshot was created. This behavior is different than in RDS for MySQL 8.0, where the AUTO_INCREMENT value is preserved during these operations.

Temporary tables on reader DB instances

You can't create temporary tables using the InnoDB storage engine on Aurora MySQL reader instances. On reader instances, the InnoDB storage engine is configured as read-only. Make sure that any `CREATE TEMPORARY TABLE` statements on reader instances run with the `NO_ENGINE_SUBSTITUTION` SQL mode turned off.

The error that you receive is different depending on whether you use a plain `CREATE TEMPORARY TABLE` statement or the variation `CREATE TEMPORARY TABLE AS SELECT`. The following examples show the different kinds of errors.

This temporary table behavior only applies to read-only instances. This first example confirms that's the kind of instance the session is connected to.

```
mysql> select @@innodb_read_only;
+-----+
| @@innodb_read_only |
+-----+
|           1 |
+-----+
```

For plain `CREATE TEMPORARY TABLE` statements, the statement fails when the `NO_ENGINE_SUBSTITUTION` SQL mode is turned on. It succeeds when that SQL mode is turned off.

```
mysql> set sql_mode = 'NO_ENGINE_SUBSTITUTION';

mysql> CREATE TEMPORARY TABLE tt2 (id int) ENGINE=InnoDB;
ERROR 3161 (HY000): Storage engine InnoDB is disabled (Table creation is disallowed).

mysql> SET sql_mode = '';

mysql> CREATE TEMPORARY TABLE tt4 (id int) ENGINE=InnoDB;

mysql> SHOW CREATE TABLE tt4\G
***** 1. row *****
    Table: tt4
Create Table: CREATE TEMPORARY TABLE `tt4` (
  `id` int DEFAULT NULL
) ENGINE=MyISAM DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci
```

For `CREATE TEMPORARY TABLE AS SELECT` statements, the statement fails whether or not the `NO_ENGINE_SUBSTITUTION` SQL mode is turned on. MySQL community edition doesn't support storage engine substitution with `CREATE TABLE AS SELECT` or `CREATE TEMPORARY TABLE AS SELECT` statements. For those statements, remove the `ENGINE=InnoDB` clause from your SQL code.

```
mysql> set sql_mode = 'NO_ENGINE_SUBSTITUTION';

mysql> CREATE TEMPORARY TABLE tt1 ENGINE=InnoDB AS SELECT * FROM t1;
ERROR 3161 (HY000): Storage engine InnoDB is disabled (Table creation is disallowed).

mysql> SET sql_mode = '';

mysql> CREATE TEMPORARY TABLE tt3 ENGINE=InnoDB AS SELECT * FROM t1;
ERROR 1874 (HY000): InnoDB is in read only mode.
```

Storage engine for internal temporary tables

In Aurora MySQL version 3, the way internal temporary tables work is different from earlier Aurora MySQL versions. Instead of choosing between the InnoDB and MyISAM storage engines for such temporary tables, now you choose between the `TempTable` and InnoDB storage engines.

With the `TempTable` storage engine, you can make an additional choice for how to handle certain data. The data affected overflows the memory pool that holds all the internal temporary tables for the DB instance.

Those choices can influence the performance for queries that generate high volumes of temporary data, for example while performing aggregations such as `GROUP BY` on large tables.

Tip

If your workload includes queries that generate internal temporary tables, confirm how your application performs with this change by running benchmarks and monitoring performance-related metrics.

In some cases, the amount of temporary data fits within the `TempTable` memory pool or only overflows the memory pool by a small amount. In these cases, we recommend using the `TempTable` setting for internal temporary tables and memory-mapped files to hold any overflow data. That setting is the default.

If a substantial amount of temporary data overflows the `TempTable` memory pool, we recommend using the `MEMORY` storage engine instead for internal temporary tables. Or you can choose `TempTable` for internal temporary tables and InnoDB tables to hold any overflow data.

You make the initial choice between the `TempTable` storage engine and the `MEMORY` storage engine for internal temporary tables. You do so by setting the parameter `internal_tmp_mem_storage_engine`. This parameter replaces the `internal_tmp_disk_storage_engine` parameter used in Aurora MySQL version 1 and 2.

The `TempTable` storage engine is the default. `TempTable` uses a common memory pool for all temporary tables that use this engine, instead of a maximum memory limit per table. The size of this memory pool is specified by the `temptable_max_ram` parameter. It defaults to 1 GB on DB instances with 16 or more GB of memory, and 16 MB on DB instances with less than 16 GB of memory. The size of the memory pool influences session-level memory consumption.

If you use the `TempTable` storage engine and the temporary data exceeds the size of the memory pool, Aurora MySQL stores the overflow data using a secondary mechanism.

You can set the parameters `temptable_use_mmap` and `temptable_max_mmap` to specify if the data overflows to memory-mapped temporary files or to InnoDB internal temporary tables on disk. The different data formats and overflow criteria of these overflow mechanisms can affect query performance. They do so by influencing the amount of data written to disk and the demand on disk storage throughput.

Aurora MySQL stores the overflow data differently depending on your choice of data overflow destination and whether the query runs on a writer or reader DB instance:

- On the writer instance, data that overflows to InnoDB internal temporary tables is stored in the Aurora cluster volume.
- On the writer instance, data that overflows to memory-mapped temporary files resides on local storage on the Aurora MySQL version 3 instance.
- On reader instances, overflow data always resides on memory-mapped temporary files on local storage. That's because read-only instances can't store any data on the Aurora cluster volume.

Note

The configuration parameters related to internal temporary tables apply differently to the writer and reader instances in your cluster. For reader instances, Aurora MySQL always uses the `TempTable` storage engine and a value of 1 for `temptable_use_mmap`. The size for `temptable_max_mmap` defaults to 1 GB, for both writer and reader instances, regardless of the DB instance memory size. You can adjust this value the same way as on the writer instance, except that you can't specify a value of zero for `temptable_max_mmap` on reader instances.

Binary log replication

In MySQL 8.0 community edition, binary log replication is turned on by default. In Aurora MySQL version 3, binary log replication is turned off by default.

Tip

If your high availability requirements are fulfilled by the Aurora built-in replication features, you can leave binary log replication turned off. That way, you can avoid the performance overhead of binary log replication. You can also avoid the associated monitoring and troubleshooting that are needed to manage binary log replication.

Aurora supports binary log replication from a MySQL 5.7-compatible source to Aurora MySQL version 3. The source system can be an Aurora MySQL DB cluster, an RDS for MySQL DB instance, or an on-premises MySQL instance.

As does community MySQL, Aurora MySQL supports replication from a source running a specific version to a target running the same major version or one major version higher. For example, replication from a MySQL 5.6-compatible system to Aurora MySQL version 3 isn't supported. Replicating from Aurora MySQL version 3 to a MySQL 5.7-compatible or MySQL 5.6-compatible system isn't supported. For details about using binary log replication, see [Replication between Aurora and MySQL or between Aurora and another Aurora DB cluster \(binary log replication\) \(p. 932\)](#).

Aurora MySQL version 3 includes improvements to binary log replication in community MySQL 8.0, such as filtered replication. For details about the community MySQL 8.0 improvements, see [How Servers Evaluate Replication Filtering Rules](#) in the *MySQL Reference Manual*.

Multithreaded replication

With Aurora MySQL version 3, Aurora MySQL supports multithreaded replication. For usage information, see [Multithreaded binary log replication \(Aurora MySQL version 3 and higher\) \(p. 948\)](#).

Note

We still recommend not using multithreaded replication with Aurora MySQL version 1 and version 2.

Transaction compression for binary log replication

For usage information about binary log compression, see [Binary Log Transaction Compression](#) in the *MySQL Reference Manual*.

The following limitations apply to binary log compression in Aurora MySQL version 3:

- Transactions whose binary log data is larger than the maximum allowed packet size aren't compressed, regardless of whether the Aurora MySQL binary log compression setting is turned on. Such transactions are replicated without being compressed.
- If you use a connector for change data capture (CDC) that doesn't support MySQL 8.0 yet, you can't use this feature. We recommend that you test any third-party connectors thoroughly with binary log compression before turning on binlog compression on systems that use binlog replication for CDC.

Comparison of Aurora MySQL version 3 and community MySQL 8.0

You can use the following information to learn about the changes to be aware of when you convert from a different MySQL 8.0-compatible system to Aurora MySQL version 3.

In general, Aurora MySQL version 3 supports the feature set of community MySQL 8.0.23. Some new features from MySQL 8.0 community edition don't apply to Aurora MySQL. Some of those features aren't

compatible with some aspect of Aurora, such as the Aurora storage architecture. Other features aren't needed because the Amazon RDS management service provides equivalent functionality. The following features in community MySQL 8.0 aren't supported or work differently in Aurora MySQL version 3.

For release notes for all Aurora MySQL version 3 releases, see [Database engine updates for Amazon Aurora MySQL version 3 \(p. 1108\)](#).

Topics

- [MySQL 8.0 features not available in Aurora MySQL version 3 \(p. 757\)](#)
- [Role-based privilege model \(p. 757\)](#)
- [Authentication \(p. 760\)](#)

MySQL 8.0 features not available in Aurora MySQL version 3

The following features from community MySQL 8.0 aren't available or work differently in Aurora MySQL version 3.

- Resource groups and associated SQL statements aren't supported in Aurora MySQL.
- The Aurora storage architecture means that you don't have to manually manage files and the underlying storage for your database. In particular, Aurora handles the undo tablespace differently than community MySQL does. This difference from community MySQL has the following consequences:
 - Aurora MySQL doesn't support named tablespaces.
 - The `innodb_undo_log_truncate` configuration setting is turned off and can't be turned on. Aurora has its own mechanism for reclaiming storage space.
 - Aurora MySQL doesn't have the `CREATE UNDO TABLESPACE`, `ALTER UNDO TABLESPACE ... SET INACTIVE`, and `DROP UNDO TABLESPACE` statements.
 - Aurora sets the number of undo tablespaces automatically and manages those tablespaces for you.
- TLS 1.3 isn't supported in Aurora MySQL version 3.
- The `aurora_hot_page_contention` status variable isn't available. The hot page contention feature isn't supported. For the full list of status variables not available in Aurora MySQL version 3, see [Status variables \(p. 751\)](#).
- You can't modify the settings of any MySQL plugins.
- The X plugin isn't supported.

Role-based privilege model

With Aurora MySQL version 3, you can't modify the tables in the `mysql` database directly. In particular, you can't set up users by inserting into the `mysql.user` table. Instead, you use SQL statements to grant role-based privileges. You can still query the `mysql` tables. If you use binary log replication, changes made directly to the `mysql` tables on the source cluster aren't replicated to the target cluster.

In some cases, your application might use shortcuts to create users or other objects by inserting into the `mysql` tables. If so, change your application code to use the corresponding statements such as `CREATE USER`.

To export metadata for database users during the migration from an external MySQL database, you can use `mysqldump` command instead of `mysqldump`. Use the following syntax.

```
mysqldump --exclude-databases=mysql --users
```

This statement dumps all databases except for the tables in the `mysql` system database. It also includes `CREATE USER` and `GRANT` statements to reproduce all MySQL users in the migrated database. You can

also use the [pt-show-grants tool](#) on the source system to list CREATE USER and GRANT statements to reproduce all the database users.

To simplify managing permissions for many users or applications, you can use the CREATE ROLE statement to create a role that has a set of permissions. Then you can use the GRANT and SET ROLE statements and the `current_role` function to assign roles to users or applications, switch the current role, and check which roles are in effect. For more information on the role-based permission system in MySQL 8.0, see [Using Roles](#) in the MySQL Reference Manual.

Aurora MySQL version 3 includes a special role that has all of the following privileges. This role is named `rds_superuser_role`. The primary administrative user for each cluster already has this role granted. The `rds_superuser_role` role includes the following privileges for all database objects:

- `ALTER`
- `APPLICATION_PASSWORD_ADMIN`
- `ALTER ROUTINE`
- `CONNECTION_ADMIN`
- `CREATE`
- `CREATE ROLE`
- `CREATE ROUTINE`
- `CREATE TABLESPACE`
- `CREATE TEMPORARY TABLES`
- `CREATE USER`
- `CREATE VIEW`
- `DELETE`
- `DROP`
- `DROP ROLE`
- `EVENT`
- `EXECUTE`
- `INDEX`
- `INSERT`
- `LOCK TABLES`
- `PROCESS`
- `REFERENCES`
- `RELOAD`
- `REPLICATION CLIENT`
- `REPLICATION SLAVE`
- `ROLE_ADMIN`
- `SET_USER_ID`
- `SELECT`
- `SHOW DATABASES`
- `SHOW VIEW`
- `TRIGGER`
- `UPDATE`
- `XA_RECOVER_ADMIN`

The role definition also includes `WITH GRANT OPTION` so that an administrative user can grant that role to other users. In particular, the administrator must grant any privileges needed to perform binary log replication with the Aurora MySQL cluster as the target.

Tip

To see the full details of the permissions, enter the following statements.

```
SHOW GRANTS FOR rds_superuser_role@'%';
SHOW GRANTS FOR name_of_administrative_user_for_your_cluster@'%';
```

Aurora MySQL version 3 also includes roles that you can use to access other AWS services. You can set these roles as an alternative to GRANT statements. For example, you specify GRANT AWS_LAMBDA_ACCESS TO *user* instead of GRANT INVOKE LAMBDA ON *.* TO *user*. For the procedures to access other AWS services, see [Integrating Amazon Aurora MySQL with other AWS services \(p. 984\)](#). Aurora MySQL version 3 includes the following roles related to accessing other AWS services:

- AWS_LAMBDA_ACCESS role, as an alternative to the INVOKE LAMBDA privilege. For usage information, see [Invoking a Lambda function from an Amazon Aurora MySQL DB cluster \(p. 1010\)](#).
- AWS_LOAD_S3_ACCESS role, as an alternative to the LOAD FROM S3 privilege. For usage information, see [Loading data into an Amazon Aurora MySQL DB cluster from text files in an Amazon S3 bucket \(p. 997\)](#).
- AWS_SELECT_S3_ACCESS role, as an alternative to the SELECT INTO S3 privilege. For usage information, see [Saving data from an Amazon Aurora MySQL DB cluster into text files in an Amazon S3 bucket \(p. 1004\)](#).
- AWS_SAGEMAKER_ACCESS role, as an alternative to the INVOKE SAGEMAKER privilege. For usage information, see [Using machine learning \(ML\) with Aurora MySQL \(p. 1020\)](#).
- AWS_COMPREHEND_ACCESS role, as an alternative to the INVOKE COMPREHEND privilege. For usage information, see [Using machine learning \(ML\) with Aurora MySQL \(p. 1020\)](#).

When you grant access by using roles in Aurora MySQL version 3, you also activate the role by using the SET ROLE *role_name* or SET ROLE ALL statement. The following example shows how. Substitute the appropriate role name for AWS_SELECT_S3_ACCESS.

```
# Grant role to user
mysql> GRANT AWS_SELECT_S3_ACCESS TO 'user'@'domain-or-ip-address'

# Check the current roles for your user. In this case, the AWS_SELECT_S3_ACCESS role has
# not been activated.
# Only the rds_superuser_role is currently in effect.
mysql> SELECT CURRENT_ROLE();
+-----+
| CURRENT_ROLE()          |
+-----+
| `rds_superuser_role`@`%` |
+-----+
1 row in set (0.00 sec)

# Activate all roles associated with this user using SET ROLE.
# You can activate specific roles or all roles.
# In this case, the user only has 2 roles, so we specify ALL.
mysql> SET ROLE ALL;
Query OK, 0 rows affected (0.00 sec)

# Verify role is now active
mysql> SELECT CURRENT_ROLE();
+-----+
| CURRENT_ROLE()          |
+-----+
| `AWS_LAMBDA_ACCESS`@`%`, `rds_superuser_role`@`%` |
+-----+
```

Authentication

In community MySQL 8.0, the default authentication plugin is `caching_sha2_password`. Aurora MySQL version 3 still uses the `mysql_native_password` plugin. You can't change the `default_authentication_plugin` setting.

Upgrading to Aurora MySQL version 3

For specific upgrade paths to upgrade your database from Aurora MySQL version 1 or 2 to version 3, you can use one of the following approaches:

- To upgrade an Aurora MySQL version 2 cluster to version 3, create a snapshot of the version 2 cluster and restore the snapshot to create a new version 3 cluster. Follow the procedure in [Restoring from a DB cluster snapshot \(p. 497\)](#). Currently, in-place upgrade isn't available from Aurora MySQL version 2 to Aurora MySQL version 3.
- To upgrade from Aurora MySQL version 1, first do an intermediate upgrade to Aurora MySQL version 2. To do the upgrade to Aurora MySQL version 2, use any of the upgrade methods in [Upgrading Amazon Aurora MySQL DB clusters \(p. 1088\)](#). Then use the snapshot restore technique to upgrade from Aurora MySQL version 2 to Aurora MySQL version 3. Snapshot restore isn't available from Aurora MySQL version 1 clusters (MySQL 5.6-compatible) to Aurora MySQL version 3.
- Currently, you can't clone a MySQL 5.7-compatible Aurora cluster to a MySQL 8.0-compatible one. Use the snapshot restore technique instead.
- If you have an Aurora MySQL version 2 cluster that uses backtrack, currently you can't use the snapshot restore method to upgrade to Aurora MySQL version 3. This limitation applies to all clusters that use backtrack, regardless of whether the backtrack setting is turned on. In this case, perform a logical dump and restore by using a tool such as the `mysqldump` command. For more information about using `mysqldump` for Aurora MySQL, see [Migrating from MySQL to Amazon Aurora by using mysqldump \(p. 796\)](#).

Tip

In some cases, you might specify the option to upload database logs to CloudWatch when you restore the snapshot. If so, examine the logs in CloudWatch to diagnose any issues that occur during the restore and associated upgrade operation. The CLI examples in this section demonstrate how to do so using the `--enable-cloudwatch-logs-exports` option.

Topics

- [Upgrade planning for Aurora MySQL version 3 \(p. 760\)](#)
- [Example of upgrading from Aurora MySQL version 2 to version 3 \(p. 761\)](#)
- [Example of upgrading from Aurora MySQL version 1 to version 3 \(p. 763\)](#)
- [Troubleshooting upgrade issues with Aurora MySQL version 3 \(p. 765\)](#)
- [Post-upgrade cleanup for Aurora MySQL version 3 \(p. 772\)](#)

Upgrade planning for Aurora MySQL version 3

To help you decide the right time and approach to upgrade, you can learn the differences between Aurora MySQL version 3 and your current Aurora and MySQL environment:

- If you are converting from RDS for MySQL 8.0 or community MySQL 8.0, see [Comparison of Aurora MySQL version 3 and community MySQL 8.0 \(p. 756\)](#).
- If you are upgrading from Aurora MySQL version 2, RDS for MySQL 5.7, or community MySQL 5.7, see [Comparison of Aurora MySQL version 2 and Aurora MySQL version 3 \(p. 749\)](#).
- Create new MySQL 8.0-compatible versions of any custom parameter groups. Apply any necessary custom parameter values to the new parameter groups. Consult [Parameter changes for Aurora MySQL version 3 \(p. 751\)](#) to learn about parameter changes.

Note

For most parameter settings, you can choose the custom parameter group either when you create the cluster or associate the parameter group with the cluster later. However, if you use a nondefault setting for the `lower_case_table_names` parameter, you must set up the custom parameter group with this setting in advance. Then specify the parameter group when you perform the snapshot restore to create the cluster. Any change to the `lower_case_table_names` parameter has no effect after the cluster is created.

You can also find more MySQL-specific upgrade considerations and tips in [Changes in MySQL 8.0](#) in the *MySQL Reference Manual*. For example, you can use the command `mysqlcheck --check-upgrade` to analyze your existing Aurora MySQL databases and identify potential upgrade issues.

Currently, the primary upgrade path from earlier Aurora MySQL versions to Aurora MySQL version 3 is by restoring a snapshot to create a new cluster. You can restore a snapshot of a cluster running any minor version of Aurora MySQL version 2 (MySQL 5.7-compatible) to Aurora MySQL version 3. To upgrade from Aurora MySQL version 1, you use a two-step process. First restore a snapshot to an Aurora MySQL version 2 cluster, then make a snapshot of that cluster and restore it to an Aurora MySQL version 3 cluster. For the upgrade procedure from Aurora MySQL version 1 or 2, see [Upgrading to Aurora MySQL version 3 \(p. 760\)](#). For general information about upgrading by restoring a snapshot, see [Upgrading Amazon Aurora MySQL DB clusters \(p. 1088\)](#).

After you finish the upgrade itself, you can follow the post-upgrade procedures in [Post-upgrade cleanup for Aurora MySQL version 3 \(p. 772\)](#). Finally, test your application's functionality and performance.

If you are converting from RDS from MySQL or community MySQL, follow the migration procedure explained in [Migrating data to an Amazon Aurora MySQL DB cluster \(p. 781\)](#). In some cases, you might use binary log replication to synchronize your data with an Aurora MySQL version 3 cluster as part of the migration. If so, the source system must run a version that's compatible with MySQL 5.7, or a MySQL 8.0-compatible version that is 8.0.23 or lower.

Example of upgrading from Aurora MySQL version 2 to version 3

The following AWS CLI example demonstrates the steps to upgrade an Aurora MySQL version 2 cluster to Aurora MySQL version 3.

The first step is to determine the version of the cluster that you want to upgrade. The following AWS CLI command shows how. The output confirms that the original cluster is a MySQL 5.7-compatible one that's running Aurora MySQL version 2.09.2.

This cluster has at least one DB instance. For the upgrade process to work properly, this original cluster requires a writer instance.

```
$ aws rds describe-db-clusters --db-cluster-id cluster-57-upgrade-ok \
--query '*[].EngineVersion' --output text
5.7.mysql_aurora.2.09.2
```

The following command shows how to check which upgrade paths are available from a specific version. In this case, the original cluster is running version 5.7.mysql_aurora.2.09.2. The output shows that this version can be upgraded to Aurora MySQL version 3.

If the original cluster uses a version number that is too low to upgrade to Aurora MySQL version 3, the upgrade requires an additional step. First, restore the snapshot to create a new cluster that could be upgraded to Aurora MySQL version 3. Then, take a snapshot of that intermediate cluster. Finally, restore the snapshot of the intermediate cluster to create a new Aurora MySQL version 3 cluster.

```
$ aws rds describe-db-engine-versions --engine aurora-mysql \
```

```
--engine-version 5.7.mysql_aurora.2.09.2 \
--query 'DBEngineVersions[ ].ValidUpgradeTarget[ ].EngineVersion'
[
    "5.7.mysql_aurora.2.10.0",
    "5.7.mysql_aurora.2.10.1",
    "8.0.mysql_aurora.3.01.0"
]
```

The following command creates a snapshot of the cluster to upgrade to Aurora MySQL version 3. The original cluster remains intact. You later create a new Aurora MySQL version 3 cluster from the snapshot.

```
aws rds create-db-cluster-snapshot --db-cluster-id cluster-57-upgrade-ok \
--db-cluster-snapshot-id cluster-57-upgrade-ok-snapshot
{
    "DBClusterSnapshotIdentifier": "cluster-57-upgrade-ok-snapshot",
    "DBClusterIdentifier": "cluster-57-upgrade-ok",
    "SnapshotCreateTime": "2021-10-06T23:20:18.087000+00:00"
}
```

The following command restores the snapshot to a new cluster that's running Aurora MySQL version 3.

```
$ aws rds restore-db-cluster-from-snapshot \
--snapshot-id cluster-57-upgrade-ok-snapshot \
--db-cluster-id cluster-80-restored --engine aurora-mysql \
--engine-version 8.0.mysql_aurora.3.01.0 \
--enable-cloudwatch-logs-exports '[{"error", "general", "slowquery", "audit"}]'
{
    "DBClusterIdentifier": "cluster-80-restored",
    "Engine": "aurora-mysql",
    "EngineVersion": "8.0.mysql_aurora.3.01.0",
    "Status": "creating"
}
```

Restoring the snapshot sets up the storage for the cluster and establishes the database version that the cluster can use. Because the compute capacity of the cluster is separate from the storage, you set up any DB instances for the cluster once the cluster itself is created. The following example creates a writer DB instance using one of the db.r5 instance classes.

Tip

You might have administration scripts that create DB instances using older instance classes such as db.r3, db.r4, db.t2, or db.t3. If so, modify your scripts to use one of the instance classes that are supported with Aurora MySQL version 3. For information about the instance classes that you can use with Aurora MySQL version 3, see [Instance class support \(p. 750\)](#).

```
$ aws rds create-db-instance --db-instance-identifier instance-running-version-3 \
--db-cluster-identifier cluster-80-restored \
--db-instance-class db.r5.xlarge --engine aurora-mysql
{
    "DBInstanceIdentifier": "instance-running-version-3",
    "DBClusterIdentifier": "cluster-80-restored",
    "DBInstanceClass": "db.r5.xlarge",
    "EngineVersion": "8.0.mysql_aurora.3.01.0",
    "DBInstanceStatus": "creating"
}
```

After the upgrade is finished, you can verify the Aurora-specific version number of the cluster by using the AWS CLI.

```
$ aws rds describe-db-clusters --db-cluster-id cluster-80-restored \
--query '*[ ].EngineVersion' --output text
```

```
8.0.mysql_aurora.3.01.0
```

You can also verify the MySQL-specific version of the database engine by calling the `version` function.

```
mysql> select version();
+-----+
| version() |
+-----+
| 8.0.23   |
+-----+
```

Example of upgrading from Aurora MySQL version 1 to version 3

The following example shows the two-stage upgrade process if the original snapshot is from a version that can't be directly restored to Aurora MySQL version 3. Instead, that snapshot is restored to a cluster running an intermediate version that you can upgrade to Aurora MySQL version 3. This intermediate cluster doesn't need any associated DB instances. Then, another snapshot is created from the intermediate cluster. Finally, the second snapshot is restored to create a new Aurora MySQL version 3 cluster and a writer DB instance.

The Aurora MySQL version 1 cluster that we start with is named `aurora-mysql-v1-to-v2`. It's running Aurora MySQL version 1.23.4. It has at least one DB instance in the cluster.

This example checks which Aurora MySQL version 2 versions can be upgraded to the `8.0.mysql_aurora.3.01.0` to use on the upgraded cluster. For this example, we choose version 2.10.0 as the intermediate version.

```
$ aws rds describe-db-engine-versions --engine aurora-mysql \
--query '*[]'.
{EngineVersion:EngineVersion,TargetVersions:ValidUpgradeTarget[*].EngineVersion} | \
[?contains(TargetVersions, `'8.0.mysql_aurora.3.01.0'`) == `true`]|[].EngineVersion' \
--output text
...
5.7.mysql_aurora.2.08.3
5.7.mysql_aurora.2.09.1
5.7.mysql_aurora.2.09.2
5.7.mysql_aurora.2.10.0
...
```

The following example verifies that Aurora MySQL version 1.23.4 to 2.10.0 is an available upgrade path. Thus, the Aurora MySQL version that we're running can be upgraded to 2.10.0. Then that cluster can be upgraded to 3.01.0.

```
aws rds describe-db-engine-versions --engine aurora \
--query '*[]'.
{EngineVersion:EngineVersion,TargetVersions:ValidUpgradeTarget[*].EngineVersion} | \
[?contains(TargetVersions, `'5.7.mysql_aurora.2.10.0'`) == `true`]|[].EngineVersion' \
--output text
...
5.6.mysql_aurora.1.22.5
5.6.mysql_aurora.1.23.0
5.6.mysql_aurora.1.23.1
5.6.mysql_aurora.1.23.2
5.6.mysql_aurora.1.23.3
5.6.mysql_aurora.1.23.4
...
```

The following example creates a snapshot named `aurora-mysql-v1-to-v2-snapshot` that's based on the original Aurora MySQL version 1 cluster.

```
$ aws rds create-db-cluster-snapshot \
--db-cluster-id aurora-mysql-v1-to-v2 \
--db-cluster-snapshot-id aurora-mysql-v1-to-v2-snapshot
{
  "DBClusterSnapshotIdentifier": "aurora-mysql-v1-to-v2-snapshot",
  "DBClusterIdentifier": "aurora-mysql-v1-to-v2"
}
```

The following example creates the intermediate Aurora MySQL version 2 cluster from the version 1 snapshot. This intermediate cluster is named `aurora-mysql-v2-to-v3`. It's running Aurora MySQL version 2.10.0.

The example also creates a writer instance for the cluster. For the upgrade process to work properly, this intermediate cluster requires a writer instance.

```
$ aws rds restore-db-cluster-from-snapshot \
--snapshot-id aurora-mysql-v1-to-v2-snapshot \
--db-cluster-id aurora-mysql-v2-to-v3 \
--engine aurora-mysql --engine-version 5.7.mysql_aurora.2.10.0 \
--enable-cloudwatch-logs-exports '[{"error","general","slowquery","audit"}'
{
  "DBCluster": {
    "AllocatedStorage": 1,
    "AvailabilityZones": [
      "us-east-1a",
      "us-east-1d",
      "us-east-1f"
    ],
    ...
  }
}

$ aws rds create-db-instance --db-instance-identifier upgrade-demo-instance \
--db-cluster-identifier aurora-mysql-v2-to-v3 \
--db-instance-class db.r5.xlarge \
--engine aurora-mysql
{
  "DBInstanceIdentifier": "upgrade-demo-instance",
  "DBInstanceClass": "db.r5.xlarge",
  "DBInstanceState": "creating"
}
```

The following example creates a snapshot from the intermediate Aurora MySQL version 2 cluster. This snapshot is named `aurora-mysql-v2-to-v3-snapshot`. This is the snapshot to be restored to create the Aurora MySQL version 3 cluster.

```
$ aws rds create-db-cluster-snapshot \
--db-cluster-id aurora-mysql-v2-to-v3 \
--db-cluster-snapshot-id aurora-mysql-v2-to-v3-snapshot
{
  "DBClusterSnapshotIdentifier": "aurora-mysql-v2-to-v3-snapshot",
  "DBClusterIdentifier": "aurora-mysql-v2-to-v3"
}
```

The following command creates the Aurora MySQL version 3 cluster. This cluster is named `aurora-mysql-v3-fully-upgraded`.

```
$ aws rds restore-db-cluster-from-snapshot \
--snapshot-id aurora-mysql-v2-to-v3-snapshot \
--db-cluster-id aurora-mysql-v3-fully-upgraded \
--engine aurora-mysql --engine-version 8.0.mysql_aurora.3.01.0 \
--enable-cloudwatch-logs-exports '[{"error","general","slowquery","audit"}']
```

```
{  
    "DBCluster": {  
        "AllocatedStorage": 1,  
        "AvailabilityZones": [  
            "us-east-1b",  
            "us-east-1c",  
            "us-east-1d"  
        ],  
        ...  
    },
```

Now that the Aurora MySQL version 3 cluster is created, the following example creates a writer DB instance for it. When the cluster and the writer instance become available, you can connect to the cluster and begin using it. All of the data from the original cluster is preserved through each of the snapshot stages.

```
$ aws rds create-db-instance \  
  --db-instance-identifier instance-also-running-v3 \  
  --db-cluster-identifier aurora-mysql-v3-fully-upgraded \  
  --db-instance-class db.r5.xlarge --engine aurora-mysql  
{  
    "DBInstanceIdentifier": "instance-also-running-v3",  
    "DBClusterIdentifier": "aurora-mysql-v3-fully-upgraded",  
    "DBInstanceClass": "db.r5.xlarge",  
    "EngineVersion": "8.0.mysql_aurora.3.01.0",  
    "DBInstanceState": "creating"  
}
```

Troubleshooting upgrade issues with Aurora MySQL version 3

If your upgrade to Aurora MySQL version 3 doesn't complete successfully, you can diagnose the cause of the problem. Then you can make any required changes to the original database schema or table data and run the upgrade process again.

If the upgrade process to Aurora MySQL version 3 fails, the problem is detected while creating and then upgrading the writer instance for the restored snapshot. Aurora leaves behind the original 5.7-compatible writer instance. That way, you can examine the log from the preliminary checks that Aurora runs before performing the upgrade. The following examples start with a 5.7-compatible database that requires some changes before it can be upgraded to Aurora MySQL version 3. The examples demonstrate how the first attempted upgrade doesn't succeed, how to examine the log file, and how to fix the problems and run a successful upgrade.

First, we create a new MySQL 5.7-compatible cluster named `problematic-57-80-upgrade`. As the name suggests, this cluster contains at least one schema object that causes a problem during an upgrade to a MySQL 8.0-compatible version.

```
$ aws rds create-db-cluster --engine aurora-mysql \  
  --engine-version 5.7.mysql_aurora.2.10.0 \  
  --db-cluster-identifier problematic-57-80-upgrade \  
  --master-username my_username \  
  --master-user-password my_password  
{  
    "DBClusterIdentifier": "problematic-57-80-upgrade",  
    "Status": "creating"  
}  
  
$ aws rds create-db-instance \  
  --db-instance-identifier instance-preupgrade \  
  --db-cluster-identifier problematic-57-80-upgrade \  
  --db-instance-class db.t2.small --engine aurora-mysql  
{  
    "DBInstanceIdentifier": "instance-preupgrade",
```

```

        "DBClusterIdentifier": "problematic-57-80-upgrade",
        "DBInstanceClass": "db.t2.small",
        "DBInstanceState": "creating"
    }

$ aws rds wait db-instance-available \
--db-instance-identifier instance-preupgrade

```

In the cluster that we intend to upgrade, we introduce a problematic table. Creating a `FULLTEXT` index and then dropping the index leaves behind some metadata that causes a problem during the upgrade.

```

$ mysql -u my_username -p \
-h problematic-57-80-upgrade.cluster-example123.us-east-1.rds.amazonaws.com

mysql> create database problematic_upgrade;
Query OK, 1 row affected (0.02 sec)

mysql> use problematic_upgrade;
Database changed
mysql> CREATE TABLE dangling_fulltext_index
    -> (id INT AUTO_INCREMENT PRIMARY KEY, txtcol TEXT NOT NULL)
    -> ENGINE=InnoDB;
Query OK, 0 rows affected (0.05 sec)

mysql> ALTER TABLE dangling_fulltext_index ADD FULLTEXT(txtcol);
Query OK, 0 rows affected, 1 warning (0.14 sec)

mysql> ALTER TABLE dangling_fulltext_index DROP INDEX txtcol;
Query OK, 0 rows affected (0.06 sec)

```

This example attempts to perform the upgrade procedure. We take a snapshot of the original cluster and wait for snapshot creation to complete. Then we restore the snapshot, specifying the MySQL 8.0-compatible version number. We also create the writer instance for the cluster. That is the point where the upgrade processing actually happens. Then we wait for the writer instance to become available. That's the point where the upgrade process is finished, whether it succeeded or failed.

Tip

If you restore the snapshot using the AWS Management Console, Aurora creates the writer instance automatically and upgrades it to the requested engine version.

```

$ aws rds create-db-cluster-snapshot --db-cluster-id problematic-57-80-upgrade \
--db-cluster-snapshot-id problematic-57-80-upgrade-snapshot
{
    "DBClusterSnapshotIdentifier": "problematic-57-80-upgrade-snapshot",
    "DBClusterIdentifier": "problematic-57-80-upgrade",
    "Engine": "aurora-mysql",
    "EngineVersion": "5.7.mysql_aurora.2.10.0"
}

$ aws rds wait db-cluster-snapshot-available \
--db-cluster-snapshot-id problematic-57-80-upgrade-snapshot

$ aws rds restore-db-cluster-from-snapshot \
--snapshot-id problematic-57-80-upgrade-snapshot \
--db-cluster-id cluster-80-attempt-1 --engine aurora-mysql \
--engine-version 8.0.mysql_aurora.3.01.0 \
--enable-cloudwatch-logs-exports '["error","general","slowquery","audit"]'
{
    "DBClusterIdentifier": "cluster-80-attempt-1",
    "Engine": "aurora-mysql",
    "EngineVersion": "8.0.mysql_aurora.3.01.0",
    "Status": "creating"
}

```

```
$ aws rds create-db-instance --db-instance-identifier instance-attempt-1 \
--db-cluster-identifier cluster-80-attempt-1 \
--db-instance-class db.r5.xlarge --engine aurora-mysql
{
    "DBInstanceIdentifier": "instance-attempt-1",
    "DBClusterIdentifier": "cluster-80-attempt-1",
    "DBInstanceClass": "db.r5.xlarge",
    "EngineVersion": "8.0.mysql_aurora.3.01.0",
    "DBInstanceState": "creating"
}

$ aws rds wait db-instance-available \
--db-instance-identifier instance-attempt-1
```

Now we examine the newly created cluster and associated instance to verify if the upgrade succeeded. The cluster and instance are still running a MySQL 5.7-compatible version. That means that the upgrade failed. When an upgrade fails, Aurora only leaves the writer instance behind so that you can examine any log files. You can't restart the upgrade process with that newly created cluster. After you correct the problem by making changes in your original cluster, you must run the upgrade steps again: make a new snapshot of the original cluster and restore it to another MySQL 8.0-compatible cluster.

```
$ aws rds describe-db-clusters \
--db-cluster-identifier cluster-80-attempt-1 \
--query '*[].[Status]' --output text
available
$ aws rds describe-db-clusters \
--db-cluster-identifier cluster-80-attempt-1 \
--query '*[].[EngineVersion]' --output text
5.7.mysql_aurora.2.10.0

$ aws rds describe-db-instances \
--db-instance-identifier instance-attempt-1 \
--query '*[].[{DBInstanceState:DBInstanceState}]' --output text
available
$ aws rds describe-db-instances \
--db-instance-identifier instance-attempt-1 \
--query '*[].[EngineVersion]' --output text
5.7.mysql_aurora.2.10.0
```

To get a summary of what happened during the upgrade process, we get a listing of events for the newly created writer instance. In this example, we list the events over the last 600 minutes to cover the whole time interval of the upgrade process. The events in the listing aren't necessarily in chronological order. The highlighted event shows the problem that confirms the cluster couldn't be upgraded.

```
$ aws rds describe-events \
--source-identifier instance-attempt-1 --source-type db-instance \
--duration 600
{
    "Events": [
        {
            "SourceIdentifier": "instance-attempt-1",
            "SourceType": "db-instance",
            "Message": "Binlog position from crash recovery is mysql-bin-changelog.000001
154",
            "EventCategories": [],
            "Date": "2021-12-03T20:26:17.862000+00:00",
            "SourceArn": "arn:aws:rds:us-east-1:123456789012:db:instance-attempt-1"
        },
        {
            "SourceIdentifier": "instance-attempt-1",
            "SourceType": "db-instance",
            "Message": "Binlog position from crash recovery is mysql-bin-changelog.000001
155",
            "EventCategories": [],
            "Date": "2021-12-03T20:26:17.862000+00:00",
            "SourceArn": "arn:aws:rds:us-east-1:123456789012:db:instance-attempt-1"
        }
    ]
}
```

```

    "Message": "Database cluster is in a state that cannot be upgraded:
PreUpgrade checks failed: Oscar PreChecker Found 1 errors",
    "EventCategories": [
        "maintenance"
    ],
    "Date": "2021-12-03T20:26:50.436000+00:00",
    "SourceArn": "arn:aws:rds:us-east-1:123456789012:db:instance-attempt-1"
},
{
    "SourceIdentifier": "instance-attempt-1",
    "SourceType": "db-instance",
    "Message": "DB instance created",
    "EventCategories": [
        "creation"
    ],
    "Date": "2021-12-03T20:26:58.830000+00:00",
    "SourceArn": "arn:aws:rds:us-east-1:123456789012:db:instance-attempt-1"
},
...

```

To diagnose the exact cause of the problem, examine the database logs for the newly created writer instance. When an upgrade to an 8.0-compatible version fails, the instance contains a log file with the file name `upgrade-prechecks.log`. This example shows how to detect the presence of that log and then download it to a local file for examination.

```

$ aws rds describe-db-log-files --db-instance-identifier instance-attempt-1 \
--query '*[].[LogFileName]' --output text
error/mysql-error-running.log
error/mysql-error-running.log.2021-12-03.20
error/mysql-error-running.log.2021-12-03.21
error/mysql-error.log
external/mysql-external.log
upgrade-prechecks.log

$ aws rds download-db-log-file-portion --db-instance-identifier instance-attempt-1 \
--log-file-name upgrade-prechecks.log --starting-token 0 \
--output text >upgrade_prechecks.log

```

The `upgrade-prechecks.log` file is in JSON format. We download it using the `--output text` option to avoid encoding JSON output within another JSON wrapper. For Aurora MySQL version 3 upgrades, this log always includes certain informational and warning messages. It only includes error messages if the upgrade fails. If the upgrade succeeds, the log file isn't produced at all. The following excerpts show the kinds of entries you can expect to find.

```

$ cat upgrade-prechecks.log
{
    "serverAddress": "/tmp%2Fmysql.sock",
    "serverVersion": "5.7.12",
    "targetVersion": "8.0.23",
    "auroraServerVersion": "2.10.0",
    "auroraTargetVersion": "3.01.0",
    "outfilePath": "/rdsdbdata/tmp/PreChecker.log",
    "checksPerformed": [

```

If `"detectedProblems"` is empty, the upgrade didn't encounter any occurrences of that type of problem. You can ignore those entries.

```

{
    "id": "oldTemporalCheck",
    "title": "Usage of old temporal type",
    "status": "OK",

```

```
    "detectedProblems": []
},
```

Checks for removed variables or changed default values aren't performed automatically. Aurora uses the parameter group mechanism instead of a physical configuration file. You always receive some messages with this `CONFIGURATION_ERROR` status, whether or not the variable changes have any effect on your database. You can consult the MySQL documentation for details about these changes.

```
{
  "id": "removedSysLogVars",
  "title": "Removed system variables for error logging to the system log configuration",
  "status": "CONFIGURATION_ERROR",
  "description": "To run this check requires full path to MySQL server configuration file to be specified at 'configPath' key of options dictionary",
  "documentationLink": "https://dev.mysql.com/doc/relnotes/mysql/8.0/en/news-8-0-13.html#mysqld-8-0-13-logging"
},
```

This warning about obsolete date and time data types occurs if the `SQL_MODE` setting in your parameter group is left at the default value. Your database might or might not contain columns with the affected types.

```
{
  "id": "zeroDatesCheck",
  "title": "Zero Date, Datetime, and Timestamp values",
  "status": "OK",
  "description": "Warning: By default zero date/datetime/timestamp values are no longer allowed in MySQL, as of 5.7.8 NO_ZERO_IN_DATE and NO_ZERO_DATE are included in SQL_MODE by default. These modes should be used with strict mode as they will be merged with strict mode in a future release. If you do not include these modes in your SQL_MODE setting, you are able to insert date/datetime/timestamp values that contain zeros. It is strongly advised to replace zero values with valid ones, as they may not work correctly in the future.",
  "documentationLink": "https://lefred.be/content/mysql-8-0-and-wrong-dates/",
  "detectedProblems": [
    {
      "level": "Warning",
      "dbObject": "global.sql_mode",
      "description": "does not contain either NO_ZERO_DATE or NO_ZERO_IN_DATE which allows insertion of zero dates"
    }
  ]
},
```

When the `detectedProblems` field contains entries with a `level` value of `Error`, that means that the upgrade can't succeed until you correct those issues.

```
{
  "id": "getDanglingFulltextIndex",
  "title": "Tables with dangling FULLTEXT index reference",
  "status": "OK",
  "description": "Error: The following tables contain dangling FULLTEXT index which is not supported. It is recommended to rebuild the table before upgrade.",
  "detectedProblems": [
    {
      "level": "Error",
      "dbObject": "problematic_upgrade.dangling_fulltext_index",
      "description": "Table `problematic_upgrade.dangling_fulltext_index` contains dangling FULLTEXT index. Kindly recreate the table before upgrade."
    }
  ]
},
```

```
        }
    ],
},
```

Tip

To summarize all of those errors and display the associated object and description fields, you can run the command `grep -A 2 '"level": "Error"` on the contents of the `upgrade-prechecks.log` file. Doing so displays each error line and the two lines after it, which contain the name of the corresponding database object and guidance about how to correct the problem.

```
$ cat upgrade-prechecks.log | grep -A 2 '"level": "Error"'
"level": "Error",
"dbObject": "problematic_upgrade.dangling_fulltext_index",
"description": "Table `problematic_upgrade.dangling_fulltext_index` contains
dangling FULLTEXT index. Kindly recreate the table before upgrade."
```

This `defaultAuthenticationPlugin` check always displays this warning message for Aurora MySQL version 3 upgrades. That's because Aurora MySQL version 3 uses the `mysql_native_password` plugin instead of `caching_sha2_password`. You don't need to take any action for this warning.

```
{
  "id": "defaultAuthenticationPlugin",
  "title": "New default authentication plugin considerations",
  "description": "Warning: The new default authentication plugin
'caching_sha2_password' offers more secure password hashing than previously
used 'mysql_native_password' (and consequent improved client connection
...
  "documentationLink": "https://dev.mysql.com/doc/refman/8.0/en/upgrading-from-previous-
series.html#upgrade-caching-sha2-password-compatibility-issues\nhttps://dev.mysql.com/
doc/refman/8.0/en/upgrading-from-previous-series.html#upgrade-caching-sha2-password-
replication"
}
```

The end of the `upgrade-prechecks.log` file summarizes how many checks encountered each type of minor or severe problem. A nonzero `errorCount` indicates that the upgrade failed.

```
],
  "errorCount": 1,
  "warningCount": 2,
  "noticeCount": 0,
  "Summary": "1 errors were found. Please correct these issues before
upgrading to avoid compatibility issues."
}
```

The next sequence of examples demonstrates how to fix this particular issue and run the upgrade process again. This time, the upgrade succeeds.

First, we go back to the original cluster and create a new table with the same structure and contents as the one with faulty metadata. In practice, you would probably rename this table back to the original table name after the upgrade.

```
$ mysql -u my_username -p \
-h problematic-57-80-upgrade.cluster-example123.us-east-1.rds.amazonaws.com

mysql> show databases;
+-----+
| Database      |
+-----+
| information_schema |
```

```

| mysql          |
| performance_schema |
| problematic_upgrade |
| sys            |
+-----+
5 rows in set (0.00 sec)

mysql> use problematic_upgrade;
mysql> show tables;
+-----+
| Tables_in_problematic_upgrade |
+-----+
| dangling_fulltext_index        |
+-----+
1 row in set (0.00 sec)

mysql> desc dangling_fulltext_index;
+-----+-----+-----+-----+-----+
| Field | Type   | Null | Key | Default | Extra       |
+-----+-----+-----+-----+-----+
| id    | int(11) | NO   | PRI | NULL    | auto_increment |
| txtcol | text    | NO   |     | NULL    |               |
+-----+-----+-----+-----+-----+
2 rows in set (0.00 sec)

mysql> CREATE TABLE recreated_table LIKE dangling_fulltext_index;
Query OK, 0 rows affected (0.03 sec)

mysql> INSERT INTO recreated_table SELECT * FROM dangling_fulltext_index;
Query OK, 0 rows affected (0.00 sec)

mysql> drop table dangling_fulltext_index;
Query OK, 0 rows affected (0.05 sec)

```

Now we go through the same process as before: creating a snapshot from the original cluster, restoring the snapshot to a new MySQL 8.0-compatible cluster, and creating a writer instance to complete the upgrade process.

```

$ aws rds create-db-cluster-snapshot --db-cluster-id problematic-57-80-upgrade \
--db-cluster-snapshot-id problematic-57-80-upgrade-snapshot-2
{
  "DBClusterSnapshotIdentifier": "problematic-57-80-upgrade-snapshot-2",
  "DBClusterIdentifier": "problematic-57-80-upgrade",
  "Engine": "aurora-mysql",
  "EngineVersion": "5.7.mysql_aurora.2.10.0"
}

$ aws rds wait db-cluster-snapshot-available \
--db-cluster-snapshot-id problematic-57-80-upgrade-snapshot-2

$ aws rds restore-db-cluster-from-snapshot \
--snapshot-id problematic-57-80-upgrade-snapshot-2 \
--db-cluster-id cluster-80-attempt-2 --engine aurora-mysql \
--engine-version 8.0.mysql_aurora.3.01.0 \
--enable-cloudwatch-logs-exports '[{"error", "general", "slowquery", "audit"}]'
{
  "DBClusterIdentifier": "cluster-80-attempt-2",
  "Engine": "aurora-mysql",
  "EngineVersion": "8.0.mysql_aurora.3.01.0",
  "Status": "creating"
}

$ aws rds create-db-instance --db-instance-identifier instance-attempt-2 \
--db-cluster-identifier cluster-80-attempt-2 \

```

```
--db-instance-class db.r5.xlarge --engine aurora-mysql
{
    "DBInstanceIdentifier": "instance-attempt-2",
    "DBClusterIdentifier": "cluster-80-attempt-2",
    "DBInstanceClass": "db.r5.xlarge",
    "EngineVersion": "8.0.mysql_aurora.3.01.0",
    "DBInstanceState": "creating"
}

$ aws rds wait db-instance-available \
--db-instance-identifier instance-attempt-2
```

This time, checking the version after the upgrade completes confirms that the version number changed to reflect Aurora MySQL version 3. We can connect to the database and confirm the MySQL engine version is an 8.0-compatible one. We confirm that the upgraded cluster contains the fixed version of the table that caused the original upgrade problem.

```
$ aws rds describe-db-clusters \
--db-cluster-identifier cluster-80-attempt-2 \
--query '*[].[EngineVersion]' --output text
8.0.mysql_aurora.3.01.0
$ aws rds describe-db-instances \
--db-instance-identifier instance-attempt-2 \
--query '*[].[EngineVersion]' --output text
8.0.mysql_aurora.3.01.0

$ mysql -h cluster-80-attempt-2.cluster-example123.us-east-1.rds.amazonaws.com \
-u my_username -p

mysql> select version();
+-----+
| version() |
+-----+
| 8.0.23   |
+-----+
1 row in set (0.00 sec)

mysql> show databases;
+-----+
| Database      |
+-----+
| information_schema |
| mysql          |
| performance_schema |
| problematic_upgrade |
| sys            |
+-----+
5 rows in set (0.00 sec)

mysql> use problematic_upgrade;
Database changed
mysql> show tables;
+-----+
| Tables_in_problematic_upgrade |
+-----+
| recreated_table                |
+-----+
1 row in set (0.00 sec)
```

Post-upgrade cleanup for Aurora MySQL version 3

After you finish upgrading any Aurora MySQL version 1 or 2 clusters to Aurora MySQL version 3, you can perform these other cleanup actions:

- Create new MySQL 8.0-compatible versions of any custom parameter groups. Apply any necessary custom parameter values to the new parameter groups.
- Update any CloudWatch alarms, setup scripts, and so on to use the new names for any metrics whose names were affected by inclusive language changes. For a list of such metrics, see [Inclusive language changes for Aurora MySQL version 3 \(p. 751\)](#).
- Update any AWS CloudFormation templates to use the new names for any configuration parameters whose names were affected by inclusive language changes. For a list of such parameters, see [Inclusive language changes for Aurora MySQL version 3 \(p. 751\)](#).

Spatial indexes

After upgrading to Aurora MySQL version 3, check if you need to drop or recreate objects and indexes related to spatial indexes. Before MySQL 8.0, Aurora could optimize spatial queries using indexes that didn't contain a spatial resource identifier (SRID). Aurora MySQL version 3 only uses spatial indexes containing SRIDs. During an upgrade, Aurora automatically drops any spatial indexes without SRIDs and prints warning messages in the database log. If you observe such warning messages, create new spatial indexes with SRIDs after the upgrade. For more information about changes to spatial functions and data types in MySQL 8.0, see [Changes in MySQL 8.0](#) in the *MySQL Reference Manual*.

Aurora MySQL version 2 compatible with MySQL 5.7

The following features are supported in MySQL 5.7.12 but are currently not supported in Aurora MySQL 5.7:

- Group replication plugin
- Increased page size
- InnoDB buffer pool loading at startup
- InnoDB full-text parser plugin
- Multisource replication
- Online buffer pool resizing
- Password validation plugin
- Query rewrite plugins
- Replication filtering
- The `CREATE TABLESPACE` SQL statement
- X Protocol

For more information about these features, see the [MySQL 5.7 documentation](#).

Comparison of Aurora MySQL 5.6 and Aurora MySQL 5.7

The following Amazon Aurora MySQL features are supported in Aurora MySQL 5.6, but these features are currently not supported in Aurora MySQL 5.7.

- Asynchronous key prefetch (AKP). For more information, see [Optimizing Amazon Aurora indexed join queries with asynchronous key prefetch \(p. 1036\)](#).
- Hash joins. For more information, see [Optimizing large Aurora MySQL join queries with hash joins \(p. 1038\)](#).
- Native functions for synchronously invoking AWS Lambda functions. You can asynchronously invoke AWS Lambda functions from Aurora MySQL 5.7. For more information, see [Invoking a Lambda function with an Aurora MySQL native function \(p. 1012\)](#).
- Scan batching. For more information, see [Aurora MySQL database engine updates 2017-12-11 \(p. 1225\)](#).

Currently, Aurora MySQL 5.7 does not support features added in Aurora MySQL version 1.16 and later. For information about Aurora MySQL version 1.16, see [Aurora MySQL database engine updates 2017-12-11 \(p. 1225\)](#).

The performance schema isn't available for early releases of Aurora MySQL 5.7. Upgrade to Aurora 2.03 or higher for performance schema support.

Security with Amazon Aurora MySQL

Security for Amazon Aurora MySQL is managed at three levels:

- To control who can perform Amazon RDS management actions on Aurora MySQL DB clusters and DB instances, you use AWS Identity and Access Management (IAM). When you connect to AWS using IAM credentials, your AWS account must have IAM policies that grant the permissions required to perform Amazon RDS management operations. For more information, see [Identity and access management in Amazon Aurora \(p. 1724\)](#).

If you are using IAM to access the Amazon RDS console, you must first sign on to the AWS Management Console with your IAM user credentials. Then go to the Amazon RDS console at <https://console.aws.amazon.com/rds/>.

- Aurora MySQL DB clusters must be created in an Amazon Virtual Private Cloud (VPC). To control which devices and Amazon EC2 instances can open connections to the endpoint and port of the DB instance for Aurora MySQL DB clusters in a VPC, you use a VPC security group. These endpoint and port connections can be made using Secure Sockets Layer (SSL). In addition, firewall rules at your company can control whether devices running at your company can open connections to a DB instance. For more information on VPCs, see [Amazon Virtual Private Cloud VPCs and Amazon Aurora \(p. 1787\)](#).

The supported VPC tenancy depends on the DB instance class used by your Aurora MySQL DB clusters. With default VPC tenancy, the VPC runs on shared hardware. With dedicated VPC tenancy, the VPC runs on a dedicated hardware instance. The burstable performance DB instance classes support default VPC tenancy only. The burstable performance DB instance classes include the db.t2, db.t3, and db.t4g DB instance classes. All other Aurora MySQL DB instance classes support both default and dedicated VPC tenancy.

For more information about instance classes, see [Aurora DB instance classes \(p. 54\)](#). For more information about default and dedicated VPC tenancy, see [Dedicated instances](#) in the [Amazon Elastic Compute Cloud User Guide](#).

- To authenticate login and permissions for an Amazon Aurora MySQL DB cluster, you can take either of the following approaches, or a combination of them:
 - You can take the same approach as with a standalone instance of MySQL.

Commands such as `CREATE USER`, `RENAME USER`, `GRANT`, `REVOKE`, and `SET PASSWORD` work just as they do in on-premises databases, as does directly modifying database schema tables. For more information, see [Access control and account management](#) in the MySQL documentation.

- You can also use IAM database authentication.

With IAM database authentication, you authenticate to your DB cluster by using an IAM user or IAM role and an authentication token. An *authentication token* is a unique value that is generated using the Signature Version 4 signing process. By using IAM database authentication, you can use the same credentials to control access to your AWS resources and your databases. For more information, see [IAM database authentication \(p. 1743\)](#).

Note

For more information, see [Security in Amazon Aurora \(p. 1706\)](#).

Master user privileges with Amazon Aurora MySQL

When you create an Amazon Aurora MySQL DB instance, the master user has the following default privileges:

- ALTER
- ALTER ROUTINE
- CREATE
- CREATE ROUTINE
- CREATE TEMPORARY TABLES
- CREATE USER
- CREATE VIEW
- DELETE
- DROP
- EVENT
- EXECUTE
- GRANT OPTION
- INDEX
- INSERT
- LOAD FROM S3
- LOCK TABLES
- PROCESS
- REFERENCES
- RELOAD
- REPLICATION CLIENT
- REPLICATION SLAVE
- SELECT
- SHOW DATABASES
- SHOW VIEW
- TRIGGER
- UPDATE

To provide management services for each DB cluster, the `rdsadmin` user is created when the DB cluster is created. Attempting to drop, rename, change the password, or change privileges for the `rdsadmin` account results in an error.

For management of the Aurora MySQL DB cluster, the standard `kill` and `kill_query` commands have been restricted. Instead, use the Amazon RDS commands `rds_kill` and `rds_kill_query` to terminate user sessions or queries on Aurora MySQL DB instances.

Note

Encryption of a database instance and snapshots is not supported for the China (Ningxia) region.

Using SSL/TLS with Aurora MySQL DB clusters

Amazon Aurora MySQL DB clusters support Secure Sockets Layer (SSL) and Transport Layer Security (TLS) connections from applications using the same process and public key as RDS for MySQL DB instances.

Amazon RDS creates an SSL/TLS certificate and installs the certificate on the DB instance when Amazon RDS provisions the instance. These certificates are signed by a certificate authority. The SSL/TLS

certificate includes the DB instance endpoint as the Common Name (CN) for the SSL/TLS certificate to guard against spoofing attacks. As a result, you can only use the DB cluster endpoint to connect to a DB cluster using SSL/TLS if your client supports Subject Alternative Names (SAN). Otherwise, you must use the instance endpoint of a writer instance.

For information about downloading certificates, see [Using SSL/TLS to encrypt a connection to a DB cluster \(p. 1714\)](#).

We recommend the MariaDB Connector/J client as a client that supports SAN with SSL. For more information, see the [MariaDB Connector/J download](#) page.

Topics

- [Requiring an SSL/TLS connection to an Aurora MySQL DB cluster \(p. 776\)](#)
- [TLS versions for Aurora MySQL \(p. 776\)](#)
- [Encrypting connections to an Aurora MySQL DB cluster \(p. 777\)](#)

Requiring an SSL/TLS connection to an Aurora MySQL DB cluster

You can require that all user connections to your Aurora MySQL DB cluster use SSL/TLS by using the `require_secure_transport` DB cluster parameter. By default, the `require_secure_transport` parameter is set to OFF. You can set the `require_secure_transport` parameter to ON to require SSL/TLS for connections to your DB cluster.

You can set the `require_secure_transport` parameter value by updating the DB cluster parameter group for your DB cluster. You don't need to reboot your DB cluster for the change to take effect. For more information on parameter groups, see [Working with DB parameter groups and DB cluster parameter groups \(p. 339\)](#).

Note

The `require_secure_transport` parameter is only available for Aurora MySQL version 5.7. You can set this parameter in a custom DB cluster parameter group. The parameter isn't available in DB instance parameter groups.

When the `require_secure_transport` parameter is set to ON for a DB cluster, a database client can connect to it if it can establish an encrypted connection. Otherwise, an error message similar to the following is returned to the client:

```
MySQL Error 3159 (HY000): Connections using insecure transport are prohibited while --require_secure_transport=ON.
```

TLS versions for Aurora MySQL

Aurora MySQL supports Transport Layer Security (TLS) versions 1.0, 1.1, and 1.2. The following table shows the TLS support for Aurora MySQL versions.

Aurora MySQL version	TLS 1.0	TLS 1.1	TLS 1.2
Aurora MySQL version 3	Supported	Supported	Supported
Aurora MySQL version 2	Supported	Supported	Supported
Aurora MySQL version 1	Supported	Supported for Aurora MySQL 1.23.1 and higher	Supported for Aurora MySQL 1.23.1 and higher

Although the community edition of MySQL 8.0 supports TLS 1.3, the MySQL 8.0-compatible Aurora MySQL version 3 currently doesn't support TLS 1.3.

For an Aurora MySQL 5.7 DB cluster, you can use the `tls_version` DB cluster parameter to indicate the permitted protocol versions. Similar client parameters exist for most client tools or database drivers. Some older clients might not support newer TLS versions. By default, the DB cluster attempts to use the highest TLS protocol version allowed by both the server and client configuration.

Set the `tls_version` DB cluster parameter to one of the following values:

- `TLSv1.2` – Only the TLS version 1.2 protocol is permitted for encrypted connections.
- `TLSv1.1` – TLS version 1.1 and 1.2 protocols are permitted for encrypted connections.
- `TLSv1` – TLS version 1.0, 1.1, and 1.2 protocols are permitted for encrypted connections.

If the parameter isn't set, then TLS version 1.0, 1.1, and 1.2 protocols are permitted for encrypted connections.

For information about modifying parameters in a DB cluster parameter group, see [Modifying parameters in a DB cluster parameter group \(p. 349\)](#). If you use the AWS CLI to modify the `tls_version` DB cluster parameter, the `ApplyMethod` must be set to `pending-reboot`. When the application method is `pending-reboot`, changes to parameters are applied after you stop and restart the DB clusters associated with the parameter group.

Note

The `tls_version` DB cluster parameter isn't available for Aurora MySQL 5.6.

Encrypting connections to an Aurora MySQL DB cluster

To encrypt connections using the default `mysql` client, launch the `mysql` client using the `--ssl-ca` parameter to reference the public key, for example:

For MySQL 5.7 and 8.0:

```
mysql -h myinstance.123456789012.rds-us-east-1.amazonaws.com  
--ssl-ca=full_path_to_CA_certificate --ssl-mode=VERIFY_IDENTITY
```

For MySQL 5.6:

```
mysql -h myinstance.123456789012.rds-us-east-1.amazonaws.com  
--ssl-ca=full_path_to_CA_certificate --ssl-verify-server-cert
```

Replace `full_path_to_CA_certificate` with the full path to your Certificate Authority (CA) certificate. For information about downloading a certificate, see [Using SSL/TLS to encrypt a connection to a DB cluster \(p. 1714\)](#).

You can require SSL/TLS connections for specific users accounts. For example, you can use one of the following statements, depending on your MySQL version, to require SSL/TLS connections on the user account `encrypted_user`.

For MySQL 5.7 and 8.0:

```
ALTER USER 'encrypted_user'@'%' REQUIRE SSL;
```

For MySQL 5.6:

```
GRANT USAGE ON *.* TO 'encrypted_user'@'%' REQUIRE SSL;
```

When you use an RDS proxy, you connect to the proxy endpoint instead of the usual cluster endpoint. You can make SSL/TLS required or optional for connections to the proxy, in the same way as for connections directly to the Aurora DB cluster. For information about using the RDS Proxy, see [Using Amazon RDS Proxy \(p. 288\)](#).

Note

For more information on SSL/TLS connections with MySQL, see the [MySQL documentation](#).

Updating applications to connect to Aurora MySQL DB clusters using new SSL/TLS certificates

As of September 19, 2019, Amazon RDS has published new Certificate Authority (CA) certificates for connecting to your Aurora DB clusters using Secure Socket Layer or Transport Layer Security (SSL/TLS). Following, you can find information about updating your applications to use the new certificates.

This topic can help you to determine whether any client applications use SSL/TLS to connect to your DB clusters. If they do, you can further check whether those applications require certificate verification to connect.

Note

Some applications are configured to connect to Aurora MySQL DB clusters only if they can successfully verify the certificate on the server.

For such applications, you must update your client application trust stores to include the new CA certificates.

After you update your CA certificates in the client application trust stores, you can rotate the certificates on your DB clusters. We strongly recommend testing these procedures in a development or staging environment before implementing them in your production environments.

For more information about certificate rotation, see [Rotating your SSL/TLS certificate \(p. 1715\)](#). For more information about downloading certificates, see [Using SSL/TLS to encrypt a connection to a DB cluster \(p. 1714\)](#). For information about using SSL/TLS with Aurora MySQL DB clusters, see [Using SSL/TLS with Aurora MySQL DB clusters \(p. 775\)](#).

Topics

- [Determining whether any applications are connecting to your Aurora MySQL DB cluster using SSL \(p. 778\)](#)
- [Determining whether a client requires certificate verification to connect \(p. 779\)](#)
- [Updating your application trust store \(p. 780\)](#)
- [Example Java code for establishing SSL connections \(p. 781\)](#)

Determining whether any applications are connecting to your Aurora MySQL DB cluster using SSL

If you are using Aurora MySQL version 2 (compatible with MySQL 5.7) and the Performance Schema is enabled, run the following query to check if connections are using SSL/TLS. For information about enabling the Performance Schema, see [Performance Schema quick start](#) in the MySQL documentation.

```
mysql> SELECT id, user, host, connection_type
```

```
FROM performance_schema.threads pst
INNER JOIN information_schema.processlist isp
ON pst.processlist_id = isp.id;
```

In this sample output, you can see both your own session (`admin`) and an application logged in as `webapp1` are using SSL.

```
+----+-----+-----+-----+
| id | user           | host          | connection_type |
+----+-----+-----+-----+
|  8 | admin          | 10.0.4.249:42590 | SSL/TLS        |
|  4 | event_scheduler | localhost      | NULL          |
| 10 | webapp1        | 159.28.1.1:42189 | SSL/TLS        |
+----+-----+-----+-----+
3 rows in set (0.00 sec)
```

If you are using Aurora MySQL version 1 (compatible with MySQL 5.6), then you can't determine from the server side whether applications are connecting with or without SSL. For those versions, you can determine whether SSL is used by examining the application's connection method. You can find more information on examining the client connection configuration in the following section.

Determining whether a client requires certificate verification to connect

You can check whether JDBC clients and MySQL clients require certificate verification to connect.

JDBC

The following example with MySQL Connector/J 8.0 shows one way to check an application's JDBC connection properties to determine whether successful connections require a valid certificate. For more information on all of the JDBC connection options for MySQL, see [Configuration properties](#) in the MySQL documentation.

When using the MySQL Connector/J 8.0, an SSL connection requires verification against the server CA certificate if your connection properties have `sslMode` set to `VERIFY_CA` or `VERIFY_IDENTITY`, as in the following example.

```
Properties properties = new Properties();
properties.setProperty("sslMode", "VERIFY_IDENTITY");
properties.put("user", DB_USER);
properties.put("password", DB_PASSWORD);
```

Note

If you use either the MySQL Java Connector v5.1.38 or later, or the MySQL Java Connector v8.0.9 or later to connect to your databases, even if you haven't explicitly configured your applications to use SSL/TLS when connecting to your databases, these client drivers default to using SSL/TLS. In addition, when using SSL/TLS, they perform partial certificate verification and fail to connect if the database server certificate is expired.

MySQL

The following examples with the MySQL Client show two ways to check a script's MySQL connection to determine whether successful connections require a valid certificate. For more information on all of the connection options with the MySQL Client, see [Client-side configuration for encrypted connections](#) in the MySQL documentation.

When using the MySQL 5.7 or MySQL 8.0 Client, an SSL connection requires verification against the server CA certificate if for the --ssl-mode option you specify VERIFY_CA or VERIFY_IDENTITY, as in the following example.

```
mysql -h mysql-database.rds.amazonaws.com -uadmin -ppassword --ssl-ca=/tmp/ssl-cert.pem --ssl-mode=VERIFY_CA
```

When using the MySQL 5.6 Client, an SSL connection requires verification against the server CA certificate if you specify the --ssl-verify-server-cert option, as in the following example.

```
mysql -h mysql-database.rds.amazonaws.com -uadmin -ppassword --ssl-ca=/tmp/ssl-cert.pem --ssl-verify-server-cert
```

Updating your application trust store

For information about updating the trust store for MySQL applications, see [Installing SSL certificates](#) in the MySQL documentation.

Note

When you update the trust store, you can retain older certificates in addition to adding the new certificates.

Updating your application trust store for JDBC

You can update the trust store for applications that use JDBC for SSL/TLS connections.

To update the trust store for JDBC applications

1. Download the 2019 root certificate that works for all AWS Regions and put the file in the trust store directory.

For information about downloading the root certificate, see [Using SSL/TLS to encrypt a connection to a DB cluster \(p. 1714\)](#).

2. Convert the certificate to .der format using the following command.

```
openssl x509 -outform der -in rds-ca-2019-root.pem -out rds-ca-2019-root.der
```

Replace the file name with the one that you downloaded.

3. Import the certificate into the key store using the following command.

```
keytool -import -alias rds-root -keystore clientkeystore -file rds-ca-2019-root.der
```

4. Confirm that the key store was updated successfully.

```
keytool -list -v -keystore clientkeystore.jks
```

Enter the metastore password when you are prompted for it.

Your output should contain the following.

```
rds-root, date, trustedCertEntry,  
Certificate fingerprint (SHA1):  
D4:0D:DB:29:E3:75:0D:FF:A6:71:C3:14:0B:BF:5F:47:8D:1C:80:96  
# This fingerprint should match the output from the below command  
openssl x509 -fingerprint -in rds-ca-2019-root.pem -noout
```

If you are using the mysql JDBC driver in an application, set the following properties in the application.

```
System.setProperty("javax.net.ssl.trustStore", certs);
System.setProperty("javax.net.ssl.trustStorePassword", "password");
```

When you start the application, set the following properties.

```
java -Djavax.net.ssl.trustStore=/path_to_truststore/MyTruststore.jks -
Djavax.net.ssl.trustStorePassword=my_truststore_password com.companyName.MyApplication
```

Example Java code for establishing SSL connections

The following code example shows how to set up the SSL connection that validates the server certificate using JDBC.

```
public class MySQLSSLTest {

    private static final String DB_USER = "user name";
    private static final String DB_PASSWORD = "password";
    // This key store has only the prod root ca.
    private static final String KEY_STORE_FILE_PATH = "file-path-to-keystore";
    private static final String KEY_STORE_PASS = "keystore-password";

    public static void test(String[] args) throws Exception {
        Class.forName("com.mysql.jdbc.Driver");

        System.setProperty("javax.net.ssl.trustStore", KEY_STORE_FILE_PATH);
        System.setProperty("javax.net.ssl.trustStorePassword", KEY_STORE_PASS);

        Properties properties = new Properties();
        properties.setProperty("sslMode", "VERIFY_IDENTITY");
        properties.put("user", DB_USER);
        properties.put("password", DB_PASSWORD);

        Connection connection = DriverManager.getConnection("jdbc:mysql://jagdeeps-ssl-
test.cni62e2e7kwh.us-east-1.rds.amazonaws.com:3306", properties);
        Statement stmt=connection.createStatement();

        ResultSet rs=stmt.executeQuery("SELECT 1 from dual");

        return;
    }
}
```

Important

After you have determined that your database connections use SSL/TLS and have updated your application trust store, you can update your database to use the rds-ca-2019 certificates. For instructions, see step 3 in [Updating your CA certificate by modifying your DB instance \(p. 1716\)](#).

Migrating data to an Amazon Aurora MySQL DB cluster

You have several options for migrating data from your existing database to an Amazon Aurora MySQL DB cluster. Your migration options also depend on the database that you are migrating from and the size of the data that you are migrating.

There are two different types of migration: physical and logical. Physical migration means that physical copies of database files are used to migrate the database. Logical migration means that the migration is accomplished by applying logical database changes, such as inserts, updates, and deletes.

Physical migration has the following advantages:

- Physical migration is faster than logical migration, especially for large databases.
- Database performance does not suffer when a backup is taken for physical migration.
- Physical migration can migrate everything in the source database, including complex database components.

Physical migration has the following limitations:

- The `innodb_page_size` parameter must be set to its default value (16KB).
- The `innodb_data_file_path` parameter must be configured with only one data file that uses the default data file name "ibdata1:12M:autoextend". Databases with two data files, or with a data file with a different name, can't be migrated using this method.

The following are examples of file names that are not allowed:

"`innodb_data_file_path=ibdata1:50M; ibdata2:50M:autoextend`" and
"`innodb_data_file_path=ibdata01:50M:autoextend`".

- The `innodb_log_files_in_group` parameter must be set to its default value (2).

Logical migration has the following advantages:

- You can migrate subsets of the database, such as specific tables or parts of a table.
- The data can be migrated regardless of the physical storage structure.

Logical migration has the following limitations:

- Logical migration is usually slower than physical migration.
- Complex database components can slow down the logical migration process. In some cases, complex database components can even block logical migration.

The following table describes your options and the type of migration for each option.

Migrating from	Migration type	Solution
An RDS for MySQL DB instance	Physical	You can migrate from an RDS for MySQL DB instance by first creating an Aurora MySQL read replica of a MySQL DB instance. When the replica lag between the MySQL DB instance and the Aurora MySQL read replica is 0, you can direct your client applications to read from the Aurora read replica and then stop replication to make the Aurora MySQL read replica a standalone Aurora MySQL DB cluster for reading and writing. For details, see Migrating data from a MySQL DB instance to an Amazon Aurora MySQL DB cluster by using an Aurora read replica (p. 803) .

Migrating from	Migration type	Solution
An RDS for MySQL DB snapshot	Physical	You can migrate data directly from an RDS for MySQL DB snapshot to an Amazon Aurora MySQL DB cluster. For details, see Migrating data from a MySQL DB instance to an Amazon Aurora MySQL DB cluster by using a DB snapshot (p. 797) .
A MySQL database external to Amazon RDS	Logical	You can create a dump of your data using the <code>mysqldump</code> utility, and then import that data into an existing Amazon Aurora MySQL DB cluster. For details, see Migrating from MySQL to Amazon Aurora by using mysqldump (p. 796) .
A MySQL database external to Amazon RDS	Physical	You can copy the backup files from your database to an Amazon Simple Storage Service (Amazon S3) bucket, and then restore an Amazon Aurora MySQL DB cluster from those files. This option can be considerably faster than migrating data using <code>mysqldump</code> . For details, see Migrating data from MySQL by using an Amazon S3 bucket (p. 784) .
A MySQL database external to Amazon RDS	Logical	You can save data from your database as text files and copy those files to an Amazon S3 bucket. You can then load that data into an existing Aurora MySQL DB cluster using the <code>LOAD DATA FROM S3</code> MySQL command. For more information, see Loading data into an Amazon Aurora MySQL DB cluster from text files in an Amazon S3 bucket (p. 997) .
A database that is not MySQL-compatible	Logical	You can use AWS Database Migration Service (AWS DMS) to migrate data from a database that is not MySQL-compatible. For more information on AWS DMS, see What is AWS database migration service?

Note

- If you are migrating a MySQL database external to Amazon RDS, the migration options described in the table are supported only if your database supports the InnoDB or MyISAM tablespaces.
- If the MySQL database you are migrating to Aurora MySQL uses memcached, remove memcached before migrating it.

Migrating data from an external MySQL database to an Amazon Aurora MySQL DB cluster

If your database supports the InnoDB or MyISAM tablespaces, you have these options for migrating your data to an Amazon Aurora MySQL DB cluster:

- You can create a dump of your data using the `mysqldump` utility, and then import that data into an existing Amazon Aurora MySQL DB cluster. For more information, see [Migrating from MySQL to Amazon Aurora by using mysqldump \(p. 796\)](#).
- You can copy the full and incremental backup files from your database to an Amazon S3 bucket, and then restore an Amazon Aurora MySQL DB cluster from those files. This option can be considerably faster than migrating data using `mysqldump`. For more information, see [Migrating data from MySQL by using an Amazon S3 bucket \(p. 784\)](#).

Migrating data from MySQL by using an Amazon S3 bucket

You can copy the full and incremental backup files from your source MySQL version 5.5, 5.6, or 5.7 database to an Amazon S3 bucket, and then restore an Amazon Aurora MySQL DB cluster from those files.

This option can be considerably faster than migrating data using `mysqldump`, because using `mysqldump` replays all of the commands to recreate the schema and data from your source database in your new Aurora MySQL DB cluster. By copying your source MySQL data files, Aurora MySQL can immediately use those files as the data for an Aurora MySQL DB cluster.

Note

The Amazon S3 bucket and the Amazon Aurora MySQL DB cluster must be in the same AWS Region.

Aurora MySQL doesn't restore everything from your database. You should save the database schema and values for the following items from your source MySQL database and add them to your restored Aurora MySQL DB cluster after it has been created:

- User accounts
- Functions
- Stored procedures
- Time zone information. Time zone information is loaded from the local operating system of your Amazon Aurora MySQL DB cluster. For more information, see [Local time zone for Amazon Aurora DB clusters \(p. 16\)](#).

You can't restore from an encrypted source database, but you can encrypt the data being migrated. You can also leave the data unencrypted during the migration process.

You can't migrate from a source database that has tables defined outside of the default MySQL data directory.

Also, decide whether you want to minimize downtime by using binary log replication during the migration process. If you use binary log replication, the external MySQL database remains open to transactions while the data is being migrated to the Aurora MySQL DB cluster. After the Aurora MySQL DB cluster has been created, you use binary log replication to synchronize the Aurora MySQL DB cluster with the transactions that happened after the backup. When the Aurora MySQL DB cluster is caught up with the MySQL database, you finish the migration by completely switching to the Aurora MySQL DB cluster for new transactions.

Topics

- [Before you begin \(p. 785\)](#)
- [Backing up files to be restored as an Amazon Aurora MySQL DB cluster \(p. 786\)](#)
- [Restoring an Amazon Aurora MySQL DB cluster from an Amazon S3 bucket \(p. 788\)](#)
- [Synchronizing the Amazon Aurora MySQL DB cluster with the MySQL database using replication \(p. 791\)](#)

Before you begin

Before you can copy your data to an Amazon S3 bucket and restore a DB cluster from those files, you must do the following:

- Install Percona XtraBackup on your local server.
- Permit Aurora MySQL to access your Amazon S3 bucket on your behalf.

Installing Percona XtraBackup

Amazon Aurora can restore a DB cluster from files that were created using Percona XtraBackup. You can install Percona XtraBackup from [Download Percona XtraBackup](#).

Note

For MySQL 5.7 migration, you must use Percona XtraBackup 2.4. For earlier MySQL versions, use Percona XtraBackup 2.3 or 2.4.

Required permissions

To migrate your MySQL data to an Amazon Aurora MySQL DB cluster, several permissions are required:

- The user that is requesting that Aurora create a new cluster from an Amazon S3 bucket must have permission to list the buckets for your AWS account. You grant the user this permission using an AWS Identity and Access Management (IAM) policy.
- Aurora requires permission to act on your behalf to access the Amazon S3 bucket where you store the files used to create your Amazon Aurora MySQL DB cluster. You grant Aurora the required permissions using an IAM service role.
- The user making the request must also have permission to list the IAM roles for your AWS account.
- If the user making the request is to create the IAM service role or request that Aurora create the IAM service role (by using the console), then the user must have permission to create an IAM role for your AWS account.
- If you plan to encrypt the data during the migration process, update the IAM policy of the user who will perform the migration to grant RDS access to the AWS KMS keys used for encrypting the backups. For instructions, see [Creating an IAM policy to access AWS KMS resources \(p. 990\)](#).

For example, the following IAM policy grants a user the minimum required permissions to use the console to list IAM roles, create an IAM role, list the Amazon S3 buckets for your account, and list the KMS keys.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": [  
                "iam>ListRoles",  
                "iam>CreateRole",  
                "iam>CreatePolicy",  
                "s3>ListBuckets",  
                "kms>ListKeys"  
            ]  
        }  
    ]  
}
```

```
        "iam:AttachRolePolicy",
        "s3>ListBucket",
        "kms>ListKeys"
    ],
    "Resource": "*"
}
]
```

Additionally, for a user to associate an IAM role with an Amazon S3 bucket, the IAM user must have the `iam:PassRole` permission for that IAM role. This permission allows an administrator to restrict which IAM roles a user can associate with Amazon S3 buckets.

For example, the following IAM policy allows a user to associate the role named `S3Access` with an Amazon S3 bucket.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowS3AccessRole",
            "Effect": "Allow",
            "Action": "iam:PassRole",
            "Resource": "arn:aws:iam::123456789012:role/S3Access"
        }
    ]
}
```

For more information on IAM user permissions, see [Managing access using policies \(p. 1726\)](#).

Creating the IAM service role

You can have the AWS Management Console create a role for you by choosing the **Create a New Role** option (shown later in this topic). If you select this option and specify a name for the new role, then Aurora creates the IAM service role required for Aurora to access your Amazon S3 bucket with the name that you supply.

As an alternative, you can manually create the role using the following procedure.

To create an IAM role for Aurora to access Amazon S3

1. Complete the steps in [Creating an IAM policy to access Amazon S3 resources \(p. 986\)](#).
2. Complete the steps in [Creating an IAM role to allow Amazon Aurora to access AWS services \(p. 991\)](#).
3. Complete the steps in [Associating an IAM role with an Amazon Aurora MySQL DB cluster \(p. 992\)](#).

Backing up files to be restored as an Amazon Aurora MySQL DB cluster

You can create a full backup of your MySQL database files using Percona XtraBackup and upload the backup files to an Amazon S3 bucket. Alternatively, if you already use Percona XtraBackup to back up your MySQL database files, you can upload your existing full and incremental backup directories and files to an Amazon S3 bucket.

Creating a full backup with Percona XtraBackup

To create a full backup of your MySQL database files that can be restored from Amazon S3 to create an Amazon Aurora MySQL DB cluster, use the Percona XtraBackup utility (`xtrabackup`) to back up your database.

For example, the following command creates a backup of a MySQL database and stores the files in the /on-premises/s3-restore/backup folder.

```
xtrabackup --backup --user=<myuser> --password=<password> --target-dir=</on-premises/s3-restore/backup>
```

If you want to compress your backup into a single file (which can be split, if needed), you can use the --stream option to save your backup in one of the following formats:

- Gzip (.gz)
- tar (.tar)
- Percona xbstream (.xbstream)

The following command creates a backup of your MySQL database split into multiple Gzip files.

```
xtrabackup --backup --user=<myuser> --password=<password> --stream=tar \  
--target-dir=</on-premises/s3-restore/backup> | gzip - | split -d --bytes=500MB \  
- </on-premises/s3-restore/backup/backup>.tar.gz
```

The following command creates a backup of your MySQL database split into multiple tar files.

```
xtrabackup --backup --user=<myuser> --password=<password> --stream=tar \  
--target-dir=</on-premises/s3-restore/backup> | split -d --bytes=500MB \  
- </on-premises/s3-restore/backup/backup>.tar
```

The following command creates a backup of your MySQL database split into multiple xbstream files.

```
xtrabackup --backup --user=<myuser> --password=<password> --stream=xbstream \  
--target-dir=</on-premises/s3-restore/backup> | split -d --bytes=500MB \  
- </on-premises/s3-restore/backup/backup>.xbstream
```

Once you have backed up your MySQL database using the Percona XtraBackup utility, you can copy your backup directories and files to an Amazon S3 bucket.

For information on creating and uploading a file to an Amazon S3 bucket, see [Getting started with Amazon Simple Storage Service](#) in the [Amazon S3 Getting Started Guide](#).

Using incremental backups with Percona XtraBackup

Amazon Aurora MySQL supports both full and incremental backups created using Percona XtraBackup. If you already use Percona XtraBackup to perform full and incremental backups of your MySQL database files, you don't need to create a full backup and upload the backup files to Amazon S3. Instead, you can save a significant amount of time by copying your existing backup directories and files for your full and incremental backups to an Amazon S3 bucket. For more information about creating incremental backups using Percona XtraBackup, see [Incremental backup](#).

When copying your existing full and incremental backup files to an Amazon S3 bucket, you must recursively copy the contents of the base directory. Those contents include the full backup and also all incremental backup directories and files. This copy must preserve the directory structure in the Amazon S3 bucket. Aurora iterates through all files and directories. Aurora uses the `xtrabackup-checkpoints` file included with each incremental backup to identify the base directory and to order incremental backups by log sequence number (LSN) range.

For information on creating and uploading a file to an Amazon S3 bucket, see [Getting started with Amazon Simple Storage Service](#) in the [Amazon S3 Getting Started Guide](#).

Backup considerations

When you upload a file to an Amazon S3 bucket, you can use server-side encryption to encrypt the data. You can then restore an Amazon Aurora MySQL DB cluster from those encrypted files. Amazon Aurora MySQL can restore a DB cluster with files encrypted using the following types of server-side encryption:

- Server-side encryption with Amazon S3-managed keys (SSE-S3) – Each object is encrypted with a unique key employing strong multifactor encryption.
- Server-side encryption with AWS KMS-managed keys (SSE-KMS) – Similar to SSE-S3, but you have the option to create and manage encryption keys yourself, and also other differences.

For information about using server-side encryption when uploading files to an Amazon S3 bucket, see [Protecting data using server-side encryption](#) in the *Amazon S3 Developer Guide*.

Amazon S3 limits the size of a file uploaded to an Amazon S3 bucket to 5 TB. If the backup data for your database exceeds 5 TB, use the `split` command to split the backup files into multiple files that are each less than 5 TB.

Aurora limits the number of source files uploaded to an Amazon S3 bucket to 1 million files. In some cases, backup data for your database, including all full and incremental backups, can come to a large number of files. In these cases, use a tarball (.tar.gz) file to store full and incremental backup files in the Amazon S3 bucket.

Aurora consumes your backup files based on the file name. Be sure to name your backup files with the appropriate file extension based on the file format—for example, `.xbstream` for files stored using the Percona xbstream format.

Aurora consumes your backup files in alphabetical order and also in natural number order. Always use the `split` option when you issue the `xtrabackup` command to ensure that your backup files are written and named in the proper order.

Aurora doesn't support partial backups created using Percona XtraBackup. You can't use the following options to create a partial backup when you back up the source files for your database: `--tables`, `--tables-exclude`, `--tables-file`, `--databases`, `--databases-exclude`, or `--databases-file`.

For more information about backing up your database with Percona XtraBackup, see [Percona XtraBackup - documentation](#) and [The xtrabackup binary](#) on the Percona website.

Aurora supports incremental backups created using Percona XtraBackup. For more information about creating incremental backups using Percona XtraBackup, see [Incremental backup](#).

Restoring an Amazon Aurora MySQL DB cluster from an Amazon S3 bucket

You can restore your backup files from your Amazon S3 bucket to create a new Amazon Aurora MySQL DB cluster by using the Amazon RDS console.

To restore an Amazon Aurora MySQL DB cluster from files on an Amazon S3 bucket

1. Sign in to the AWS Management Console and open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. In the top right corner of the Amazon RDS console, choose the AWS Region in which to create your DB cluster. Choose the same AWS Region as the Amazon S3 bucket that contains your database backup.
3. In the navigation pane, choose **Databases**, and then choose **Restore from S3**.
4. Choose **Restore from S3**.

The **Create database by restoring from S3** page appears.

Create database by restoring from S3

S3 destination



Write audit logs to S3

Enter a destination in Amazon S3 where your audit logs will be stored. Amazon S3 is object storage build to store and retrieve any amount of data from anywhere

S3 bucket

test-eu1-bucket



S3 prefix (optional) [Info](#)

Engine options

Engine type [Info](#)

Amazon Aurora



MySQL



Edition

Amazon Aurora with MySQL compatibility

Version [Info](#)

Aurora (MySQL 5.7) 2.07.2



IAM role



IAM role

Choose or create an IAM role to grant write access to your S3 bucket.

Choose an option



Settings

DB cluster identifier [Info](#)

Type a name for your DB cluster. The name must be unique across all DB clusters owned by your AWS account in the current AWS Region.

database-1

The DB cluster identifier is case-insensitive, but is stored as all lowercase (as in "mydbcluster"). Constraints: 1 to 60 alphanumeric characters or hyphens. First character must be a letter. Can't contain two consecutive hyphens. Can't end with a hyphen.

[▼ Credentials Settings](#)

789

Master username [Info](#)

Type a login ID for the master user of your DB instance.

admin

5. Under **S3 destination**:

- a. Choose the **S3 bucket** that contains the backup files.
- b. (Optional) For **S3 folder path prefix**, enter a file path prefix for the files stored in your Amazon S3 bucket.

If you don't specify a prefix, then RDS creates your DB instance using all of the files and folders in the root folder of the S3 bucket. If you do specify a prefix, then RDS creates your DB instance using the files and folders in the S3 bucket where the path for the file begins with the specified prefix.

For example, suppose that you store your backup files on S3 in a subfolder named backups, and you have multiple sets of backup files, each in its own directory (gzip_backup1, gzip_backup2, and so on). In this case, you specify a prefix of backups/gzip_backup1 to restore from the files in the gzip_backup1 folder.

6. Under **Engine options**:

- a. For **Engine type**, choose **Amazon Aurora**.
 - b. For **Version**, choose the Aurora MySQL engine version for your restored DB instance.
7. For **IAM role**, you can choose an existing IAM role.
8. (Optional) You can also have a new IAM role created for you by choosing **Create a new role**. If so:

- a. Enter the **IAM role name**.
- b. Choose whether to **Allow access to KMS key**:
 - If you didn't encrypt the backup files, choose **No**.
 - If you encrypted the backup files with AES-256 (SSE-S3) when you uploaded them to Amazon S3, choose **No**. In this case, the data is decrypted automatically.
 - If you encrypted the backup files with AWS KMS (SSE-KMS) server-side encryption when you uploaded them to Amazon S3, choose **Yes**. Next, choose the correct KMS key for **AWS KMS key**.

The AWS Management Console creates an IAM policy that enables Aurora to decrypt the data.

For more information, see [Protecting data using server-side encryption](#) in the *Amazon S3 Developer Guide*.

9. Choose settings for your DB cluster, such as the DB cluster identifier and the login credentials. For information about each setting, see [Settings for Aurora DB clusters \(p. 137\)](#).
10. Customize additional settings for your Aurora MySQL DB cluster as needed.
11. Choose **Create database** to launch your Aurora DB instance.

On the Amazon RDS console, the new DB instance appears in the list of DB instances. The DB instance has a status of **creating** until the DB instance is created and ready for use. When the state changes to **available**, you can connect to the primary instance for your DB cluster. Depending on the DB instance class and store allocated, it can take several minutes for the new instance to be available.

To view the newly created cluster, choose the **Databases** view in the Amazon RDS console and choose the DB cluster. For more information, see [Viewing an Amazon Aurora DB cluster \(p. 547\)](#).

The screenshot shows the AWS RDS console for a database named 'database-1'. The 'Connectivity & security' tab is active. It lists two endpoints:

Endpoint name	Status
database-1.cluster-ro-... .us-east-2.rds.amazonaws.com	Available
database-1.cluster-... .us-east-2.rds.amazonaws.com	Available

Note the port and the writer endpoint of the DB cluster. Use the writer endpoint and port of the DB cluster in your JDBC and ODBC connection strings for any application that performs write or read operations.

Synchronizing the Amazon Aurora MySQL DB cluster with the MySQL database using replication

To achieve little or no downtime during the migration, you can replicate transactions that were committed on your MySQL database to your Aurora MySQL DB cluster. Replication enables the DB cluster to catch up with the transactions on the MySQL database that happened during the migration. When the DB cluster is completely caught up, you can stop the replication and finish the migration to Aurora MySQL.

Topics

- [Configuring your external MySQL database and your Aurora MySQL DB cluster for encrypted replication \(p. 792\)](#)
- [Synchronizing the Amazon Aurora MySQL DB cluster with the external MySQL database \(p. 793\)](#)

Configuring your external MySQL database and your Aurora MySQL DB cluster for encrypted replication

To replicate data securely, you can use encrypted replication.

Note

If you don't need to use encrypted replication, you can skip these steps and move on to the instructions in [Synchronizing the Amazon Aurora MySQL DB cluster with the external MySQL database \(p. 793\)](#).

The following are prerequisites for using encrypted replication:

- Secure Sockets Layer (SSL) must be enabled on the external MySQL primary database.
- A client key and client certificate must be prepared for the Aurora MySQL DB cluster.

During encrypted replication, the Aurora MySQL DB cluster acts a client to the MySQL database server. The certificates and keys for the Aurora MySQL client are in files in .pem format.

To configure your external MySQL database and your Aurora MySQL DB cluster for encrypted replication

1. Ensure that you are prepared for encrypted replication:
 - If you don't have SSL enabled on the external MySQL primary database and don't have a client key and client certificate prepared, enable SSL on the MySQL database server and generate the required client key and client certificate.
 - If SSL is enabled on the external primary, supply a client key and certificate for the Aurora MySQL DB cluster. If you don't have these, generate a new key and certificate for the Aurora MySQL DB cluster. To sign the client certificate, you must have the certificate authority key that you used to configure SSL on the external MySQL primary database.

For more information, see [Creating SSL certificates and keys using openssl](#) in the MySQL documentation.

You need the certificate authority certificate, the client key, and the client certificate.

2. Connect to the Aurora MySQL DB cluster as the primary user using SSL.

For information about connecting to an Aurora MySQL DB cluster with SSL, see [Using SSL/TLS with Aurora MySQL DB clusters \(p. 775\)](#).

3. Run the `mysql.rds_import_binlog_ssl_material` stored procedure to import the SSL information into the Aurora MySQL DB cluster.

For the `ssl_material_value` parameter, insert the information from the .pem format files for the Aurora MySQL DB cluster in the correct JSON payload.

The following example imports SSL information into an Aurora MySQL DB cluster. In .pem format files, the body code typically is longer than the body code shown in the example.

```
call mysql.rds_import_binlog_ssl_material(
  '{"ssl_ca": "-----BEGIN CERTIFICATE-----"
```

```
AAAAAB3NzaC1yc2EAAAQABAAQClKsfkNkuSevGj3eYhCe53pcjqP3maAhDFcvBS706V
hz2ItxCih+PnDSUaw+WNQn/mZphTk/a/gu8jEzoOWbkM4yxyb/wB96xbiFveSFJuOp/d6RJhJOI0iBXr
lsLnBItntckiJ7FbtJMxLvvwJryDUilBMTjYtwB+QhYXUMOzce5Pjz5/i8SeJtjnV3iAoG/cQk+0Fzz
qaeJAAHco+CY/5WrUBkrHmFJr6HcXkvJdWPkYQS3xqC0+FmUzofz221Cbt5IMucxXPkX4rWi+z7wB3Rb
BQoQzd8v7yeb70z1PnWoyN0qFU0XA246RA8QFYiCNYwI3f05p6KLxEXAMPLE
-----END CERTIFICATE-----\n","ssl_cert":"-----BEGIN CERTIFICATE-----
AAAAAB3NzaC1yc2EAAAQABAAQClKsfkNkuSevGj3eYhCe53pcjqP3maAhDFcvBS706V
hz2ItxCih+PnDSUaw+WNQn/mZphTk/a/gu8jEzoOWbkM4yxyb/wB96xbiFveSFJuOp/d6RJhJOI0iBXr
lsLnBItntckiJ7FbtJMxLvvwJryDUilBMTjYtwB+QhYXUMOzce5Pjz5/i8SeJtjnV3iAoG/cQk+0Fzz
qaeJAAHco+CY/5WrUBkrHmFJr6HcXkvJdWPkYQS3xqC0+FmUzofz221Cbt5IMucxXPkX4rWi+z7wB3Rb
BQoQzd8v7yeb70z1PnWoyN0qFU0XA246RA8QFYiCNYwI3f05p6KLxEXAMPLE
-----END CERTIFICATE-----\n","ssl_key":"-----BEGIN RSA PRIVATE KEY-----
AAAAAB3NzaC1yc2EAAAQABAAQClKsfkNkuSevGj3eYhCe53pcjqP3maAhDFcvBS706V
hz2ItxCih+PnDSUaw+WNQn/mZphTk/a/gu8jEzoOWbkM4yxyb/wB96xbiFveSFJuOp/d6RJhJOI0iBXr
lsLnBItntckiJ7FbtJMxLvvwJryDUilBMTjYtwB+QhYXUMOzce5Pjz5/i8SeJtjnV3iAoG/cQk+0Fzz
qaeJAAHco+CY/5WrUBkrHmFJr6HcXkvJdWPkYQS3xqC0+FmUzofz221Cbt5IMucxXPkX4rWi+z7wB3Rb
BQoQzd8v7yeb70z1PnWoyN0qFU0XA246RA8QFYiCNYwI3f05p6KLxEXAMPLE
-----END RSA PRIVATE KEY-----\n"}');
```

For more information, see [mysql_rds_import_binlog_ssl_material Using SSL/TLS with Aurora MySQL DB clusters \(p. 775\)](#).

Note

After running the procedure, the secrets are stored in files. To erase the files later, you can run the [mysql_rds_remove_binlog_ssl_material](#) stored procedure.

Synchronizing the Amazon Aurora MySQL DB cluster with the external MySQL database

You can synchronize your Amazon Aurora MySQL DB cluster with the MySQL database using replication.

To synchronize your Aurora MySQL DB cluster with the MySQL database using replication

1. Ensure that the /etc/my.cnf file for the external MySQL database has the relevant entries.

If encrypted replication is not required, ensure that the external MySQL database is started with binary logs (binlogs) enabled and SSL disabled. The following are the relevant entries in the /etc/my.cnf file for unencrypted data.

```
log-bin=mysql-bin
server-id=2133421
innodb_flush_log_at_trx_commit=1
sync_binlog=1
```

If encrypted replication is required, ensure that the external MySQL database is started with SSL and binlogs enabled. The entries in the /etc/my.cnf file include the .pem file locations for the MySQL database server.

```
log-bin=mysql-bin
server-id=2133421
innodb_flush_log_at_trx_commit=1
sync_binlog=1

# Setup SSL.
ssl-ca=/home/sslcerts/ca.pem
ssl-cert=/home/sslcerts/server-cert.pem
ssl-key=/home/sslcerts/server-key.pem
```

You can verify that SSL is enabled with the following command.

```
mysql> show variables like 'have_ssl';
```

Your output should be similar the following.

```
+-----+-----+
| Variable_name | Value |
+-----+-----+
| have_ssl     | YES   |
+-----+-----+
1 row in set (0.00 sec)
```

2. Determine the starting binary log position for replication. You specify the position to start replication in a later step.

Using the AWS Management Console

- Sign in to the AWS Management Console and open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
- In the navigation pane, choose **Events**.
- In the **Events** list, note the position in the **Recovered from Binary log filename** event.

Events (3)		
Identifier	Type	Event
testingest	Instances	DB instance created
testingest	Instances	Binlog position from crash recovery is OFF.000001 532
testingest-cluster	DB clusters	Recovered from Binary log filename 'mysqld-bin.00001'

Using the AWS CLI

You can also get the binlog file name and position by calling the `describe-events` command from the AWS CLI. The following shows an example `describe-events` command.

```
PROMPT> aws rds describe-events
```

In the output, identify the event that shows the binlog position.

3. While connected to the external MySQL database, create a user to be used for replication. This account is used solely for replication and must be restricted to your domain to improve security. The following is an example.

```
mysql> CREATE USER '<user_name>'@'<domain_name>' IDENTIFIED BY '<password>;'
```

The user requires the `REPLICATION CLIENT` and `REPLICATION SLAVE` privileges. Grant these privileges to the user.

```
GRANT REPLICATION CLIENT, REPLICATION SLAVE ON *.* TO '<user_name>'@'<domain_name>';
```

If you need to use encrypted replication, require SSL connections for the replication user. For example, you can use the following statement to require SSL connections on the user account `<user_name>`.

```
GRANT USAGE ON *.* TO '<user_name>'@'<domain_name>' REQUIRE SSL;
```

Note

If `REQUIRE SSL` is not included, the replication connection might silently fall back to an unencrypted connection.

4. In the Amazon RDS console, add the IP address of the server that hosts the external MySQL database to the VPC security group for the Aurora MySQL DB cluster. For more information on modifying a VPC security group, see [Security groups for your VPC](#) in the *Amazon Virtual Private Cloud User Guide*.

You might also need to configure your local network to permit connections from the IP address of your Aurora MySQL DB cluster, so that it can communicate with your external MySQL database. To find the IP address of the Aurora MySQL DB cluster, use the `host` command.

```
host <db_cluster_endpoint>
```

The host name is the DNS name from the Aurora MySQL DB cluster endpoint.

5. Enable binary log replication by running the `mysql.rds_set_external_master` (Aurora MySQL version 1 and 2) or `mysql.rds_set_external_source` (Aurora MySQL version 3 and higher) stored procedure. This stored procedure has the following syntax.

```
CALL mysql.rds_set_external_master (
    host_name
    , host_port
    , replication_user_name
    , replication_user_password
    , mysql_binary_log_file_name
    , mysql_binary_log_file_location
    , ssl_encryption
);
CALL mysql.rds_set_external_source (
    host_name
    , host_port
    , replication_user_name
    , replication_user_password
    , mysql_binary_log_file_name
    , mysql_binary_log_file_location
    , ssl_encryption
);
```

For information about the parameters, see [mysql_rds_set_external_master](#).

For `mysql_binary_log_file_name` and `mysql_binary_log_file_location`, use the position in the **Recovered from Binary log filename** event you noted earlier.

If the data in the Aurora MySQL DB cluster is not encrypted, the `ssl_encryption` parameter must be set to 0. If the data is encrypted, the `ssl_encryption` parameter must be set to 1.

The following example runs the procedure for an Aurora MySQL DB cluster that has encrypted data.

```
CALL mysql.rds_set_external_master(
    'Externaldb.some.com',
    3306,
    'repl_user'@'mydomain.com',
    'password',
    'mysql-bin.000010',
    120,
    1);

CALL mysql.rds_set_external_source(
    'Externaldb.some.com',
    3306,
    'repl_user'@'mydomain.com',
    'password',
    'mysql-bin.000010',
    120,
    1);
```

This stored procedure sets the parameters that the Aurora MySQL DB cluster uses for connecting to the external MySQL database and reading its binary log. If the data is encrypted, it also downloads the SSL certificate authority certificate, client certificate, and client key to the local disk.

6. Start binary log replication by running the `mysql.rds_start_replication` stored procedure.

```
CALL mysql.rds_start_replication;
```

7. Monitor how far the Aurora MySQL DB cluster is behind the MySQL replication primary database. To do so, connect to the Aurora MySQL DB cluster and run the following command.

```
Aurora MySQL version 1 and 2:
SHOW SLAVE STATUS;

Aurora MySQL version 3:
SHOW REPLICA STATUS;
```

In the command output, the `Seconds_Behind_Master` field shows how far the Aurora MySQL DB cluster is behind the MySQL primary. When this value is 0 (zero), the Aurora MySQL DB cluster has caught up to the primary, and you can move on to the next step to stop replication.

8. Connect to the MySQL replication primary database and stop replication. To do so, run the following command.

```
CALL mysql.rds_stop_replication;
```

Migrating from MySQL to Amazon Aurora by using mysqldump

Because Amazon Aurora MySQL is a MySQL-compatible database, you can use the `mysqldump` utility to copy data from your MySQL or MariaDB database to an existing Aurora MySQL DB cluster.

For a discussion of how to do so with MySQL databases that are very large, see [Importing data to a MySQL or MariaDB DB instance with reduced downtime](#). For MySQL databases that have smaller amounts of data, see [Importing data from a MySQL or MariaDB DB to a MySQL or MariaDB DB instance](#).

Migrating data from a MySQL DB instance to an Amazon Aurora MySQL DB cluster by using a DB snapshot

You can migrate (copy) data to an Amazon Aurora MySQL DB cluster from an RDS for MySQL DB snapshot, as described following.

Topics

- [Migrating an RDS for MySQL snapshot to Aurora \(p. 797\)](#)
- [Migrating data from a MySQL DB instance to an Amazon Aurora MySQL DB cluster by using an Aurora read replica \(p. 803\)](#)

Note

Because Amazon Aurora MySQL is compatible with MySQL, you can migrate data from your MySQL database by setting up replication between your MySQL database and an Amazon Aurora MySQL DB cluster. If you want to use replication to migrate data from your MySQL database, we recommend that your MySQL database run MySQL version 5.5 or later. For more information, see [Replication with Amazon Aurora \(p. 70\)](#).

Migrating an RDS for MySQL snapshot to Aurora

You can migrate a DB snapshot of an RDS for MySQL DB instance to create an Aurora MySQL DB cluster. The new Aurora MySQL DB cluster is populated with the data from the original RDS for MySQL DB instance. The DB snapshot must have been made from an Amazon RDS DB instance running MySQL version 5.6 or 5.7.

You can migrate either a manual or automated DB snapshot. After the DB cluster is created, you can then create optional Aurora Replicas.

When the MySQL DB instance and the Aurora DB cluster are running the same version of MySQL, you can restore the MySQL snapshot directly to the Aurora DB cluster. For example, you can restore a MySQL version 5.6 snapshot directly to Aurora MySQL version 5.6, but you can't restore a MySQL version 5.6 snapshot directly to Aurora MySQL version 5.7.

If you want to migrate a MySQL version 5.6 snapshot to Aurora MySQL version 5.7, you can perform the migration in one of the following ways:

- Migrate the MySQL version 5.6 snapshot to Aurora MySQL version 5.6, take a snapshot of the Aurora MySQL version 5.6 DB cluster, and then restore the Aurora MySQL version 5.6 snapshot to Aurora MySQL version 5.7.
- Upgrade the MySQL version 5.6 snapshot to MySQL version 5.7, take a snapshot of the MySQL version 5.7 DB instance, and then restore the MySQL version 5.7 snapshot to Aurora MySQL version 5.7.

Note

You can also migrate a MySQL DB instance to an Aurora MySQL DB cluster by creating an Aurora read replica of your source MySQL DB instance. For more information, see [Migrating data from a MySQL DB instance to an Amazon Aurora MySQL DB cluster by using an Aurora read replica \(p. 803\)](#).

You can't migrate a MySQL version 5.7 snapshot to Aurora MySQL version 5.6.

The general steps you must take are as follows:

1. Determine the amount of space to provision for your Aurora MySQL DB cluster. For more information, see [How much space do I need? \(p. 798\)](#)
2. Use the console to create the snapshot in the AWS Region where the Amazon RDS MySQL instance is located. For information about creating a DB snapshot, see [Creating a DB snapshot](#).
3. If the DB snapshot is not in the same AWS Region as your DB cluster, use the Amazon RDS console to copy the DB snapshot to that AWS Region. For information about copying a DB snapshot, see [Copying a DB snapshot](#).
4. Use the console to migrate the DB snapshot and create an Aurora MySQL DB cluster with the same databases as the original MySQL DB instance.

Warning

Amazon RDS limits each AWS account to one snapshot copy into each AWS Region at a time.

How much space do I need?

When you migrate a snapshot of a MySQL DB instance into an Aurora MySQL DB cluster, Aurora uses an Amazon Elastic Block Store (Amazon EBS) volume to format the data from the snapshot before migrating it. In some cases, additional space is needed to format the data for migration.

Tables that are not MyISAM tables and are not compressed can be up to 16 TB in size. If you have MyISAM tables, then Aurora must use additional space in the volume to convert the tables to be compatible with Aurora MySQL. If you have compressed tables, then Aurora must use additional space in the volume to expand these tables before storing them on the Aurora cluster volume. Because of this additional space requirement, you should ensure that none of the MyISAM and compressed tables being migrated from your MySQL DB instance exceeds 8 TB in size.

Reducing the amount of space required to migrate data into Amazon Aurora MySQL

You might want to modify your database schema prior to migrating it into Amazon Aurora. Such modification can be helpful in the following cases:

- You want to speed up the migration process.
- You are unsure of how much space you need to provision.
- You have attempted to migrate your data and the migration has failed due to a lack of provisioned space.

You can make the following changes to improve the process of migrating a database into Amazon Aurora.

Important

Be sure to perform these updates on a new DB instance restored from a snapshot of a production database, rather than on a production instance. You can then migrate the data from the snapshot of your new DB instance into your Aurora DB cluster to avoid any service interruptions on your production database.

Table type	Limitation or guideline
MyISAM tables	Aurora MySQL supports InnoDB tables only. If you have MyISAM tables in your database, then those tables must be converted before being migrated into Aurora MySQL. The conversion process requires additional space for the MyISAM to InnoDB conversion during the migration procedure.

Table type	Limitation or guideline
	<p>To reduce your chances of running out of space or to speed up the migration process, convert all of your MyISAM tables to InnoDB tables before migrating them. The size of the resulting InnoDB table is equivalent to the size required by Aurora MySQL for that table. To convert a MyISAM table to InnoDB, run the following command:</p> <pre style="margin-left: 40px;">alter table <schema>.〈table_name〉 engine=innodb, algorithm=copy;</pre>
Compressed tables	<p>Aurora MySQL doesn't support compressed tables (that is, tables created with <code>ROW_FORMAT=COMPRESSED</code>).</p> <p>To reduce your chances of running out of space or to speed up the migration process, expand your compressed tables by setting <code>ROW_FORMAT</code> to <code>DEFAULT</code>, <code>COMPACT</code>, <code>DYNAMIC</code>, or <code>REDUNDANT</code>. For more information, see https://dev.mysql.com/doc/refman/5.6/en/innodb-row-format.html.</p>

You can use the following SQL script on your existing MySQL DB instance to list the tables in your database that are MyISAM tables or compressed tables.

```
-- This script examines a MySQL database for conditions that block
-- migrating the database into Amazon Aurora.
-- It needs to be run from an account that has read permission for the
-- INFORMATION_SCHEMA database.

-- Verify that this is a supported version of MySQL.

select msg as `==> Checking current version of MySQL.`
from
(
select
    'This script should be run on MySQL version 5.6. ' +
    'Earlier versions are not supported.' as msg,
    cast(substring_index(version(), '.', 1) as unsigned) * 100 +
    cast(substring_index(substring_index(version(), '.', 2), '.', -1)
        as unsigned)
as major_minor
) as T
where major_minor <> 506;

-- List MyISAM and compressed tables. Include the table size.

select concat(TABLE_SCHEMA, '.', TABLE_NAME) as `==> MyISAM or Compressed Tables`,
round(((data_length + index_length) / 1024 / 1024), 2) "Approx size (MB)"
from INFORMATION_SCHEMA.TABLES
where
    ENGINE <> 'InnoDB'
and
(
    -- User tables
    TABLE_SCHEMA not in ('mysql', 'performance_schema',
                          'information_schema')
or
    -- Non-standard system tables
(
    TABLE_SCHEMA = 'mysql' and TABLE_NAME not in
    (
        'columns_priv', 'db', 'event', 'func', 'general_log',
        'procs_priv', 'tables_priv', 'time_zone_leap_second',
        'time_zone_name', 'time_zone_transition',
        'time_zone_transition_time'
    )
)
)
```

```
'help_category', 'help_keyword', 'help_relation',
'help_topic', 'host', 'ndb_binlog_index', 'plugin',
'proc', 'procs_priv', 'proxies_priv', 'servers', 'slow_log',
'tables_priv', 'time_zone', 'time_zone_leap_second',
'time_zone_name', 'time_zone_transition',
'time_zone_transition_type', 'user'
)
)
)
or
(
-- Compressed tables
ROW_FORMAT = 'Compressed'
);
```

The script produces output similar to the output in the following example. The example shows two tables that must be converted from MyISAM to InnoDB. The output also includes the approximate size of each table in megabytes (MB).

==> MyISAM or Compressed Tables	Approx size (MB)
test.name_table	2102.25
test.my_table	65.25

2 rows in set (0.01 sec)

Console

You can migrate a DB snapshot of an RDS for MySQL DB instance to create an Aurora MySQL DB cluster. The new Aurora MySQL DB cluster is populated with the data from the original RDS for MySQL DB instance. The DB snapshot must have been made from an Amazon RDS DB instance running MySQL version 5.6 or 5.7. For information about creating a DB snapshot, see [Creating a DB snapshot](#).

If the DB snapshot is not in the AWS Region where you want to locate your data, use the Amazon RDS console to copy the DB snapshot to that AWS Region. For information about copying a DB snapshot, see [Copying a DB snapshot](#).

When you migrate the DB snapshot by using the AWS Management Console, the console takes the actions necessary to create both the DB cluster and the primary instance.

You can also choose for your new Aurora MySQL DB cluster to be encrypted at rest using an AWS KMS key.

To migrate a MySQL DB snapshot by using the AWS Management Console

1. Sign in to the AWS Management Console and open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. Either start the migration from the MySQL DB instance or from the snapshot:

To start the migration from the DB instance:

1. In the navigation pane, choose **Databases**, and then select the MySQL DB instance.
2. For **Actions**, choose **Migrate latest snapshot**.

To start the migration from the snapshot:

1. Choose **Snapshots**.

2. On the **Snapshots** page, choose the snapshot that you want to migrate into an Aurora MySQL DB cluster.
3. Choose **Snapshot Actions**, and then choose **Migrate Snapshot**.

The **Migrate Database** page appears.

3. Set the following values on the **Migrate Database** page:
 - **Migrate to DB Engine:** Select **aurora**.
 - **DB Engine Version:** Select the DB engine version for the Aurora MySQL DB cluster.
 - **DB Instance Class:** Select a DB instance class that has the required storage and capacity for your database, for example `db.r3.large`. Aurora cluster volumes automatically grow as the amount of data in your database increases. An Aurora cluster volume can grow to a maximum size of 128 tebibytes (TiB). So you only need to select a DB instance class that meets your current storage requirements. For more information, see [Overview of Aurora storage \(p. 64\)](#).
 - **DB Instance Identifier:** Type a name for the DB cluster that is unique for your account in the AWS Region you selected. This identifier is used in the endpoint addresses for the instances in your DB cluster. You might choose to add some intelligence to the name, such as including the AWS Region and DB engine you selected, for example **aurora-cluster1**.

The DB instance identifier has the following constraints:

- It must contain from 1 to 63 alphanumeric characters or hyphens.
- Its first character must be a letter.
- It cannot end with a hyphen or contain two consecutive hyphens.
- It must be unique for all DB instances per AWS account, per AWS Region.
- **Virtual Private Cloud (VPC):** If you have an existing VPC, then you can use that VPC with your Aurora MySQL DB cluster by selecting your VPC identifier, for example `vpc-a464d1c1`. For information on using an existing VPC, see [How to create a VPC for use with Amazon Aurora \(p. 1793\)](#).

Otherwise, you can choose to have Aurora create a VPC for you by selecting **Create a new VPC**.

- **Subnet group:** If you have an existing subnet group, then you can use that subnet group with your Aurora MySQL DB cluster by selecting your subnet group identifier, for example `gs-subnet-group1`.

Otherwise, you can choose to have Aurora create a subnet group for you by selecting **Create a new subnet group**.

- **Public accessibility:** Select **No** to specify that instances in your DB cluster can only be accessed by resources inside of your VPC. Select **Yes** to specify that instances in your DB cluster can be accessed by resources on the public network. The default is **Yes**.

Note

Your production DB cluster might not need to be in a public subnet, because only your application servers require access to your DB cluster. If your DB cluster doesn't need to be in a public subnet, set **Publicly Accessible** to **No**.

- **Availability Zone:** Select the Availability Zone to host the primary instance for your Aurora MySQL DB cluster. To have Aurora select an Availability Zone for you, select **No Preference**.
- **Database Port:** Type the default port to be used when connecting to instances in the Aurora MySQL DB cluster. The default is `3306`.

Note

You might be behind a corporate firewall that doesn't allow access to default ports such as the MySQL default port, `3306`. In this case, provide a port value that your corporate firewall allows. Remember that port value later when you connect to the Aurora MySQL DB cluster.

- **Encryption:** Choose **Enable Encryption** for your new Aurora MySQL DB cluster to be encrypted at rest. If you choose **Enable Encryption**, you must choose a KMS key as the **AWS KMS key** value.

If your DB snapshot isn't encrypted, specify an encryption key to have your DB cluster encrypted at rest.

If your DB snapshot is encrypted, specify an encryption key to have your DB cluster encrypted at rest using the specified encryption key. You can specify the encryption key used by the DB snapshot or a different key. You can't create an unencrypted DB cluster from an encrypted DB snapshot.

- **Auto Minor Version Upgrade:** This setting doesn't apply to Aurora MySQL DB clusters.

For more information about engine updates for Aurora MySQL, see [Database engine updates for Amazon Aurora MySQL \(p. 1082\)](#).

4. Choose **Migrate** to migrate your DB snapshot.
5. Choose **Instances**, and then choose the arrow icon to show the DB cluster details and monitor the progress of the migration. On the details page, you can find the cluster endpoint used to connect to the primary instance of the DB cluster. For more information on connecting to an Aurora MySQL DB cluster, see [Connecting to an Amazon Aurora DB cluster \(p. 281\)](#).

AWS CLI

You can migrate a DB snapshot of an RDS for MySQL DB instance to create an Aurora DB cluster. The new DB cluster is then populated with the data from the DB snapshot. The DB snapshot must come from an Amazon RDS DB instance running MySQL version 5.6 or 5.7. For more information, see [Creating a DB snapshot](#).

If the DB snapshot is not in the AWS Region where you want to locate your data, copy the DB snapshot to that AWS Region. For more information, see [Copying a DB snapshot](#).

You can create an Aurora DB cluster from a DB snapshot of an RDS for MySQL DB instance by using the `restore-db-cluster-from-snapshot` command with the following parameters:

- `--db-cluster-identifier`

The name of the DB cluster to create.

- Either `--engine aurora-mysql` for a MySQL 5.7-compatible or 8.0-compatible DB cluster, or `--engine aurora` for a MySQL 5.6-compatible DB cluster
- `--kms-key-id`

The AWS KMS key to optionally encrypt the DB cluster with, depending on whether your DB snapshot is encrypted.

- If your DB snapshot isn't encrypted, specify an encryption key to have your DB cluster encrypted at rest. Otherwise, your DB cluster isn't encrypted.
- If your DB snapshot is encrypted, specify an encryption key to have your DB cluster encrypted at rest using the specified encryption key. Otherwise, your DB cluster is encrypted at rest using the encryption key for the DB snapshot.

Note

You can't create an unencrypted DB cluster from an encrypted DB snapshot.

- `--snapshot-identifier`

The Amazon Resource Name (ARN) of the DB snapshot to migrate. For more information about Amazon RDS ARNs, see [Amazon Relational Database Service \(Amazon RDS\)](#).

When you migrate the DB snapshot by using the `RestoreDBClusterFromSnapshot` command, the command creates both the DB cluster and the primary instance.

In this example, you create a MySQL 5.7-compatible DB cluster named `mydbcluster` from a DB snapshot with an ARN set to `mydbsnapshotARN`.

For Linux, macOS, or Unix:

```
aws rds restore-db-cluster-from-snapshot \
--db-cluster-identifier mydbcluster \
--snapshot-identifier mydbsnapshotARN \
--engine aurora-mysql
```

For Windows:

```
aws rds restore-db-cluster-from-snapshot ^
--db-cluster-identifier mydbcluster ^
--snapshot-identifier mydbsnapshotARN ^
--engine aurora-mysql
```

In this example, you create a MySQL 5.6-compatible DB cluster named `mydbcluster` from a DB snapshot with an ARN set to `mydbsnapshotARN`.

For Linux, macOS, or Unix:

```
aws rds restore-db-cluster-from-snapshot \
--db-cluster-identifier mydbcluster \
--snapshot-identifier mydbsnapshotARN \
--engine aurora
```

For Windows:

```
aws rds restore-db-cluster-from-snapshot ^
--db-cluster-identifier mydbcluster ^
--snapshot-identifier mydbsnapshotARN ^
--engine aurora
```

Migrating data from a MySQL DB instance to an Amazon Aurora MySQL DB cluster by using an Aurora read replica

Aurora uses the MySQL DB engines' binary log replication functionality to create a special type of DB cluster called an Aurora read replica for a source MySQL DB instance. Updates made to the source MySQL DB instance are asynchronously replicated to the Aurora read replica.

We recommend using this functionality to migrate from a MySQL DB instance to an Aurora MySQL DB cluster by creating an Aurora read replica of your source MySQL DB instance. When the replica lag between the MySQL DB instance and the Aurora read replica is 0, you can direct your client applications to the Aurora read replica and then stop replication to make the Aurora read replica a standalone Aurora MySQL DB cluster. Be prepared for migration to take a while, roughly several hours per terabyte (TiB) of data.

For a list of regions where Aurora is available, see [Amazon Aurora](#) in the *AWS General Reference*.

When you create an Aurora read replica of a MySQL DB instance, Amazon RDS creates a DB snapshot of your source MySQL DB instance (private to Amazon RDS, and incurring no charges). Amazon RDS then migrates the data from the DB snapshot to the Aurora read replica. After the data from the DB snapshot has been migrated to the new Aurora MySQL DB cluster, Amazon RDS starts replication between your MySQL DB instance and the Aurora MySQL DB cluster. If your MySQL DB instance contains tables that

use storage engines other than InnoDB, or that use compressed row format, you can speed up the process of creating an Aurora read replica by altering those tables to use the InnoDB storage engine and dynamic row format before you create your Aurora read replica. For more information about the process of copying a MySQL DB snapshot to an Aurora MySQL DB cluster, see [Migrating data from a MySQL DB instance to an Amazon Aurora MySQL DB cluster by using a DB snapshot \(p. 797\)](#).

You can only have one Aurora read replica for a MySQL DB instance.

Note

Replication issues can arise due to feature differences between Amazon Aurora MySQL and the MySQL database engine version of your RDS for MySQL DB instance that is the replication primary. If you encounter an error, you can find help in the [Amazon RDS community forum](#) or by contacting AWS Support.

For more information on MySQL read replicas, see [Working with read replicas of MariaDB, MySQL, and PostgreSQL DB instances](#).

Creating an Aurora read replica

You can create an Aurora read replica for a MySQL DB instance by using the console or the AWS CLI.

Console

To create an Aurora read replica from a source MySQL DB instance

1. Sign in to the AWS Management Console and open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. In the navigation pane, choose **Databases**.
3. Choose the MySQL DB instance that you want to use as the source for your Aurora read replica.
4. For **Actions**, choose **Create Aurora read replica**.
5. Choose the DB cluster specifications you want to use for the Aurora read replica, as described in the following table.

Option	Description
DB instance class	Choose a DB instance class that defines the processing and memory requirements for the primary instance in the DB cluster. For more information about DB instance class options, see Aurora DB instance classes (p. 54) .
Multi-AZ deployment	Choose Create Replica in Different Zone to create a standby replica of the new DB cluster in another Availability Zone in the target AWS Region for failover support. For more information about multiple Availability Zones, see Regions and Availability Zones (p. 11) .
DB instance identifier	Type a name for the primary instance in your Aurora read replica DB cluster. This identifier is used in the endpoint address for the primary instance of the new DB cluster. The DB instance identifier has the following constraints: <ul style="list-style-type: none">• It must contain from 1 to 63 alphanumeric characters or hyphens.• Its first character must be a letter.• It cannot end with a hyphen or contain two consecutive hyphens.

Option	Description
	<ul style="list-style-type: none"> It must be unique for all DB instances for each AWS account, for each AWS Region. <p>Because the Aurora read replica DB cluster is created from a snapshot of the source DB instance, the master user name and master password for the Aurora read replica are the same as the master user name and master password for the source DB instance.</p>
Virtual Private Cloud (VPC)	Select the VPC to host the DB cluster. Select Create new VPC to have Aurora create a VPC for you. For more information, see DB cluster prerequisites (p. 125) .
Subnet group	Select the DB subnet group to use for the DB cluster. Select Create new DB subnet group to have Aurora create a DB subnet group for you. For more information, see DB cluster prerequisites (p. 125) .
Public accessibility	Select Yes to give the DB cluster a public IP address; otherwise, select No. The instances in your DB cluster can be a mix of both public and private DB instances. For more information about hiding instances from public access, see Hiding a DB instance in a VPC from the internet (p. 1789) .
Availability zone	Determine if you want to specify a particular Availability Zone. For more information about Availability Zones, see Regions and Availability Zones (p. 11) .
VPC security groups	Select Create new VPC security group to have Aurora create a VPC security group for you. Select Select existing VPC security groups to specify one or more VPC security groups to secure network access to the DB cluster. For more information, see DB cluster prerequisites (p. 125) .
Database port	Specify the port for applications and utilities to use to access the database. Aurora MySQL DB clusters default to the default MySQL port, 3306. Firewalls at some companies block connections to this port. If your company firewall blocks the default port, choose another port for the new DB cluster.
DB parameter group	Select a DB parameter group for the Aurora MySQL DB cluster. Aurora has a default DB parameter group you can use, or you can create your own DB parameter group. For more information about DB parameter groups, see Working with DB parameter groups and DB cluster parameter groups (p. 339) .
DB cluster parameter group	Select a DB cluster parameter group for the Aurora MySQL DB cluster. Aurora has a default DB cluster parameter group you can use, or you can create your own DB cluster parameter group. For more information about DB cluster parameter groups, see Working with DB parameter groups and DB cluster parameter groups (p. 339) .

Option	Description
Encryption	<p>Choose Disable encryption if you don't want your new Aurora DB cluster to be encrypted. Choose Enable encryption for your new Aurora DB cluster to be encrypted at rest. If you choose Enable encryption, you must choose a KMS key as the AWS KMS key value.</p> <p>If your MySQL DB instance isn't encrypted, specify an encryption key to have your DB cluster encrypted at rest.</p> <p>If your MySQL DB instance is encrypted, specify an encryption key to have your DB cluster encrypted at rest using the specified encryption key. You can specify the encryption key used by the MySQL DB instance or a different key. You can't create an unencrypted DB cluster from an encrypted MySQL DB instance.</p>
Priority	Choose a failover priority for the DB cluster. If you don't select a value, the default is tier-1 . This priority determines the order in which Aurora Replicas are promoted when recovering from a primary instance failure. For more information, see Fault tolerance for an Aurora DB cluster (p. 69) .
Backup retention period	Select the length of time, from 1 to 35 days, that Aurora retains backup copies of the database. Backup copies can be used for point-in-time restores (PITR) of your database down to the second.
Enhanced Monitoring	Choose Enable enhanced monitoring to enable gathering metrics in real time for the operating system that your DB cluster runs on. For more information, see Monitoring OS metrics with Enhanced Monitoring (p. 626) .
Monitoring Role	Only available if Enhanced Monitoring is set to Enable enhanced monitoring . Choose the IAM role that you created to permit Aurora to communicate with Amazon CloudWatch Logs for you, or choose Default to have Aurora create a role for you named <code>rds-monitoring-role</code> . For more information, see Monitoring OS metrics with Enhanced Monitoring (p. 626) .
Granularity	Only available if Enhanced Monitoring is set to Enable enhanced monitoring . Set the interval, in seconds, between when metrics are collected for your DB cluster.
Auto minor version upgrade	<p>This setting doesn't apply to Aurora MySQL DB clusters.</p> <p>For more information about engine updates for Aurora MySQL, see Database engine updates for Amazon Aurora MySQL (p. 1082).</p>
Maintenance window	Select Select window and specify the weekly time range during which system maintenance can occur. Or, select No preference for Aurora to assign a period randomly.

6. Choose **Create read replica**.

AWS CLI

To create an Aurora read replica from a source MySQL DB instance, use the [create-db-cluster](#) and [create-db-instance](#) AWS CLI commands to create a new Aurora MySQL DB cluster. When you call the `create-db-cluster` command, include the `--replication-source-identifier` parameter to identify the Amazon Resource Name (ARN) for the source MySQL DB instance. For more information about Amazon RDS ARNs, see [Amazon Relational Database Service \(Amazon RDS\)](#).

Don't specify the master username, master password, or database name as the Aurora read replica uses the same master username, master password, and database name as the source MySQL DB instance.

For Linux, macOS, or Unix:

```
aws rds create-db-cluster --db-cluster-identifier sample-replica-cluster --engine aurora \
    --db-subnet-group-name mysubnetgroup --vpc-security-group-ids sg-c7e5b0d2 \
    --replication-source-identifier arn:aws:rds:us-west-2:123456789012:db:primary-mysql-
instance
```

For Windows:

```
aws rds create-db-cluster --db-cluster-identifier sample-replica-cluster --engine aurora ^
    --db-subnet-group-name mysubnetgroup --vpc-security-group-ids sg-c7e5b0d2 ^
    --replication-source-identifier arn:aws:rds:us-west-2:123456789012:db:primary-mysql-
instance
```

If you use the console to create an Aurora read replica, then Aurora automatically creates the primary instance for your DB cluster Aurora read replica. If you use the AWS CLI to create an Aurora read replica, you must explicitly create the primary instance for your DB cluster. The primary instance is the first instance that is created in a DB cluster.

You can create a primary instance for your DB cluster by using the [create-db-instance](#) AWS CLI command with the following parameters.

- `--db-cluster-identifier`
The name of your DB cluster.
- `--db-instance-class`
The name of the DB instance class to use for your primary instance.
- `--db-instance-identifier`
The name of your primary instance.
- `--engine aurora`

In this example, you create a primary instance named `myreadreplicainstance` for the DB cluster named `myreadreplicacluster`, using the DB instance class specified in `myinstanceclass`.

Example

For Linux, macOS, or Unix:

```
aws rds create-db-instance \
    --db-cluster-identifier myreadreplicacluster \
    --db-instance-class myinstanceclass \
    --db-instance-identifier myreadreplicainstance \
```

```
--engine aurora
```

For Windows:

```
aws rds create-db-instance ^
--db-cluster-identifier myreadreplicacluster ^
--db-instance-class myinstanceclass ^
--db-instance-identifier myreadreplicainstance ^
--engine aurora
```

RDS API

To create an Aurora read replica from a source MySQL DB instance, use the [CreateDBCluster](#) and [CreateDBInstance](#) Amazon RDS API commands to create a new Aurora DB cluster and primary instance. Do not specify the master username, master password, or database name as the Aurora read replica uses the same master username, master password, and database name as the source MySQL DB instance.

You can create a new Aurora DB cluster for an Aurora read replica from a source MySQL DB instance by using the [CreateDBCluster](#) Amazon RDS API command with the following parameters:

- **DBClusterIdentifier**

The name of the DB cluster to create.

- **DBSubnetGroupName**

The name of the DB subnet group to associate with this DB cluster.

- **Engine=aurora**

- **KmsKeyId**

The AWS KMS key to optionally encrypt the DB cluster with, depending on whether your MySQL DB instance is encrypted.

- If your MySQL DB instance isn't encrypted, specify an encryption key to have your DB cluster encrypted at rest. Otherwise, your DB cluster is encrypted at rest using the default encryption key for your account.
- If your MySQL DB instance is encrypted, specify an encryption key to have your DB cluster encrypted at rest using the specified encryption key. Otherwise, your DB cluster is encrypted at rest using the encryption key for the MySQL DB instance.

Note

You can't create an unencrypted DB cluster from an encrypted MySQL DB instance.

- **ReplicationSourceIdentifier**

The Amazon Resource Name (ARN) for the source MySQL DB instance. For more information about Amazon RDS ARNs, see [Amazon Relational Database Service \(Amazon RDS\)](#).

- **VpcSecurityGroupIds**

The list of EC2 VPC security groups to associate with this DB cluster.

In this example, you create a DB cluster named *myreadreplicacluster* from a source MySQL DB instance with an ARN set to *mysqlprimaryARN*, associated with a DB subnet group named *mysubnetgroup* and a VPC security group named *mysecuritygroup*.

Example

```
https://rds.us-east-1.amazonaws.com/
```

```
?Action=CreateDBCluster
&DBClusterIdentifier=myreadreplicacluster
&DBSubnetGroupName=mysubnetgroup
&Engine=aurora
&ReplicationSourceIdentifier=mysqlprimaryARN
&SignatureMethod=HmacSHA256
&SignatureVersion=4
&Version=2014-10-31
&VpcSecurityGroupIds=mysecuritygroup
&X-Amz-Algorithm=AWS4-HMAC-SHA256
&X-Amz-Credential=AKIADQKE4SARGYLE/20150927/us-east-1/rds/aws4_request
&X-Amz-Date=20150927T164851Z
&X-Amz-SignedHeaders=content-type;host;user-agent;x-amz-content-sha256;x-amz-date
&X-Amz-Signature=6a8f4bd6a98f649c75ea04a6b3929ecc75ac09739588391cd7250f5280e716db
```

If you use the console to create an Aurora read replica, then Aurora automatically creates the primary instance for your DB cluster Aurora read replica. If you use the AWS CLI to create an Aurora read replica, you must explicitly create the primary instance for your DB cluster. The primary instance is the first instance that is created in a DB cluster.

You can create a primary instance for your DB cluster by using the [CreateDBInstance](#) Amazon RDS API command with the following parameters:

- **DBClusterIdentifier**
The name of your DB cluster.
- **DBInstanceClass**
The name of the DB instance class to use for your primary instance.
- **DBInstanceIdentifier**
The name of your primary instance.
- **Engine=aurora**

In this example, you create a primary instance named *myreadreplicainstance* for the DB cluster named *myreadreplicacluster*, using the DB instance class specified in *myinstanceclass*.

Example

```
https://rds.us-east-1.amazonaws.com/
?Action=CreateDBInstance
&DBClusterIdentifier=myreadreplicacluster
&DBInstanceClass=myinstanceclass
&DBInstanceIdentifier=myreadreplicainstance
&Engine=aurora
&SignatureMethod=HmacSHA256
&SignatureVersion=4
&Version=2014-09-01
&X-Amz-Algorithm=AWS4-HMAC-SHA256
&X-Amz-Credential=AKIADQKE4SARGYLE/20140424/us-east-1/rds/aws4_request
&X-Amz-Date=20140424T194844Z
&X-Amz-SignedHeaders=content-type;host;user-agent;x-amz-content-sha256;x-amz-date
&X-Amz-Signature=bee4aabc750bf7dad0cd9e22b952bd6089d91e2a16592c2293e532eeaab8bc77
```

Viewing an Aurora read replica

You can view the MySQL to Aurora MySQL replication relationships for your Aurora MySQL DB clusters by using the AWS Management Console or the AWS CLI.

Console

To view the primary MySQL DB instance for an Aurora read replica

1. Sign in to the AWS Management Console and open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. In the navigation pane, choose **Databases**.
3. Choose the DB cluster for the Aurora read replica to display its details. The primary MySQL DB instance information is in the **Replication source** field.

The screenshot shows the 'Details' page for an Aurora MySQL DB cluster named 'aurora-mysql-db-cluster'. The page displays various configuration details:

- ARN:** arn:aws:rds:...:aurora-mysql-db-cluster
- DB cluster:** aurora-mysql-db-cluster (available)
- DB cluster role:** Replica
- Replication source:** arn:aws:rds:...:mydbinstance3 (This field is circled in red.)
- Cluster endpoint:** aurora-mysql-db-cluster. rds.amazonaws.com
- Reader endpoint:** aurora-mysql-db-cluster. rds.amazonaws.com
- Port:** 3306

AWS CLI

To view the MySQL to Aurora MySQL replication relationships for your Aurora MySQL DB clusters by using the AWS CLI, use the `describe-db-clusters` and `describe-db-instances` commands.

To determine which MySQL DB instance is the primary, use the `describe-db-clusters` and specify the cluster identifier of the Aurora read replica for the `--db-cluster-identifier` option. Refer to

the `ReplicationSourceIdentifier` element in the output for the ARN of the DB instance that is the replication primary.

To determine which DB cluster is the Aurora read replica, use the `describe-db-instances` and specify the instance identifier of the MySQL DB instance for the `--db-instance-identifier` option. Refer to the `ReadReplicaDBClusterIdentifiers` element in the output for the DB cluster identifier of the Aurora read replica.

Example

For Linux, macOS, or Unix:

```
aws rds describe-db-clusters \
--db-cluster-identifier myreadreplicacluster
```

```
aws rds describe-db-instances \
--db-instance-identifier mysqlprimary
```

For Windows:

```
aws rds describe-db-clusters ^
--db-cluster-identifier myreadreplicacluster
```

```
aws rds describe-db-instances ^
--db-instance-identifier mysqlprimary
```

Promoting an Aurora read replica

After migration completes, you can promote the Aurora read replica to a stand-alone DB cluster and direct your client applications to the endpoint for the Aurora read replica. For more information on the Aurora endpoints, see [Amazon Aurora connection management \(p. 32\)](#). Promotion should complete fairly quickly, and you can read from and write to the Aurora read replica during promotion. However, you can't delete the primary MySQL DB instance or unlink the DB Instance and the Aurora read replica during this time.

Before you promote your Aurora read replica, stop any transactions from being written to the source MySQL DB instance, and then wait for the replica lag on the Aurora read replica to reach 0. You can view the replica lag for an Aurora read replica by calling the `SHOW SLAVE STATUS` (Aurora MySQL version 1 and 2) or `SHOW REPLICAS STATUS` (Aurora MySQL version 3) command on your Aurora read replica. Check the **Seconds behind master** value.

You can start writing to the Aurora read replica after write transactions to the primary have stopped and replica lag is 0. If you write to the Aurora read replica before this and you modify tables that are also being modified on the MySQL primary, you risk breaking replication to Aurora. If this happens, you must delete and recreate your Aurora read replica.

Console

To promote an Aurora read replica to an Aurora DB cluster

1. Sign in to the AWS Management Console and open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. In the navigation pane, choose **Databases**.
3. Choose the DB cluster for the Aurora read replica.
4. For **Actions**, choose **Promote**.
5. Choose **Promote read replica**.

After you promote, confirm that the promotion has completed by using the following procedure.

To confirm that the Aurora read replica was promoted

1. Sign in to the AWS Management Console and open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. In the navigation pane, choose **Events**.
3. On the **Events** page, verify that there is a **Promoted Read Replica cluster to a stand-alone database cluster** event for the cluster that you promoted.

After promotion is complete, the primary MySQL DB instance and the Aurora read replica are unlinked, and you can safely delete the DB instance if you want.

AWS CLI

To promote an Aurora read replica to a stand-alone DB cluster, use the `promote-read-replica-db-cluster` AWS CLI command.

Example

For Linux, macOS, or Unix:

```
aws rds promote-read-replica-db-cluster \
--db-cluster-identifier myreadreplicacluster
```

For Windows:

```
aws rds promote-read-replica-db-cluster ^
--db-cluster-identifier myreadreplicacluster
```

Managing Amazon Aurora MySQL

The following sections discuss managing an Amazon Aurora MySQL DB cluster.

Topics

- [Managing performance and scaling for Amazon Aurora MySQL \(p. 812\)](#)
- [Backtracking an Aurora DB cluster \(p. 816\)](#)
- [Testing Amazon Aurora using fault injection queries \(p. 829\)](#)
- [Altering tables in Amazon Aurora using fast DDL \(p. 832\)](#)
- [Displaying volume status for an Aurora MySQL DB cluster \(p. 837\)](#)

Managing performance and scaling for Amazon Aurora MySQL

Scaling Aurora MySQL DB instances

You can scale Aurora MySQL DB instances in two ways, instance scaling and read scaling. For more information about read scaling, see [Read scaling \(p. 400\)](#).

You can scale your Aurora MySQL DB cluster by modifying the DB instance class for each DB instance in the DB cluster. Aurora MySQL supports several DB instance classes optimized for Aurora. Don't use db.t2

or db.t3 instance classes for larger Aurora clusters of size greater than 40 TB. For the specifications of the DB instance classes supported by Aurora MySQL, see [Aurora DB instance classes \(p. 54\)](#).

Maximum connections to an Aurora MySQL DB instance

The maximum number of connections allowed to an Aurora MySQL DB instance is determined by the `max_connections` parameter in the instance-level parameter group for the DB instance.

The following table lists the resulting default value of `max_connections` for each DB instance class available to Aurora MySQL. You can increase the maximum number of connections to your Aurora MySQL DB instance by scaling the instance up to a DB instance class with more memory, or by setting a larger value for the `max_connections` parameter in the DB parameter group for your instance, up to 16,000.

Instance class	<code>max_connections</code> default value
db.t2.small	45
db.t2.medium	90
db.t3.small	45
db.t3.medium	90
db.t3.large	135
db.t4g.medium	90
db.t4g.large	135
db.r3.large	1000
db.r3.xlarge	2000
db.r3.2xlarge	3000
db.r3.4xlarge	4000
db.r3.8xlarge	5000
db.r4.large	1000
db.r4.xlarge	2000
db.r4.2xlarge	3000
db.r4.4xlarge	4000
db.r4.8xlarge	5000
db.r4.16xlarge	6000
db.r5.large	1000
db.r5.xlarge	2000
db.r5.2xlarge	3000
db.r5.4xlarge	4000

Instance class	max_connections default value		
db.r5.8xlarge	5000		
db.r5.12xlarge	6000		
db.r5.16xlarge	6000		
db.r5.24xlarge	7000		
db.r6g.large	1000		
db.r6g.xlarge	2000		
db.r6g.2xlarge	3000		
db.r6g.4xlarge	4000		
db.r6g.8xlarge	5000		
db.r6g.12xlarge	6000		
db.r6g.16xlarge	6000		
db.x2g.large	2000		
db.x2g.xlarge	3000		
db.x2g.2xlarge	4000		
db.x2g.4xlarge	5000		
db.x2g.8xlarge	6000		
db.x2g.12xlarge	7000		
db.x2g.16xlarge	7000		

If you create a new parameter group to customize your own default for the connection limit, you'll see that the default connection limit is derived using a formula based on the `DBInstanceClassMemory` value. As shown in the preceding table, the formula produces connection limits that increase by 1000 as the memory doubles between progressively larger R3, R4, and R5 instances, and by 45 for different memory sizes of T2 and T3 instances.

The `DBInstanceClassMemory` value represents the memory capacity, in bytes, available for the DB instance. It's a number that Aurora computes internally and isn't directly available for you to query. Aurora reserves some memory in each DB instance for the Aurora management components. This adjustment to the available memory produces a lower `max_connections` value than if the formula used the full memory for the associated DB instance class.

Aurora MySQL and RDS for MySQL DB instances have different amounts of memory overhead. Therefore, the `max_connections` value can be different for Aurora MySQL and RDS for MySQL DB instances that use the same instance class. The values in the table only apply to Aurora MySQL DB instances.

The much lower connectivity limits for T2 and T3 instances are because with Aurora, those instance classes are intended only for development and test scenarios, not for production workloads.

The default connection limits are tuned for systems that use the default values for other major memory consumers, such as the buffer pool and query cache. If you change those other settings for your cluster,

consider adjusting the connection limit to account for the increase or decrease in available memory on the DB instances.

Temporary storage limits for Aurora MySQL

Aurora MySQL stores tables and indexes in the Aurora storage subsystem. Aurora MySQL uses separate temporary storage for non-persistent temporary files. This includes files that are used for such purposes as sorting large datasets during query processing or for index build operations. For more about storage, see [Amazon Aurora storage and reliability \(p. 64\)](#).

The following table shows the maximum amount of temporary storage available for each Aurora MySQL DB instance class.

DB instance class	Maximum temporary storage available (GiB)
db.x2g.16xlarge	1280
db.x2g.12xlarge	960
db.x2g.8xlarge	640
db.x2g.4xlarge	320
db.x2g.2xlarge	160
db.x2g.xlarge	80
db.x2g.large	40
db.r6g.16xlarge	1280
db.r6g.12xlarge	960
db.r6g.8xlarge	640
db.r6g.4xlarge	320
db.r6g.2xlarge	160
db.r6g.xlarge	80
db.r6g.large	32
db.r5.24xlarge	1920
db.r5.16xlarge	1280
db.r5.12xlarge	960
db.r5.8xlarge	640
db.r5.4xlarge	320
db.r5.2xlarge	160
db.r5.xlarge	80
db.r5.large	32
db.r4.16xlarge	1280
db.r4.8xlarge	640

DB instance class	Maximum temporary storage available (GiB)
db.r4.4xlarge	320
db.r4.2xlarge	160
db.r4.xlarge	80
db.r4.large	32
db.t4g.large	32
db.t4g.medium	32
db.t3.large	32
db.t3.medium	32
db.t3.small	32
db.t2.medium	32
db.t2.small	32

Important

These values represent the theoretical maximum amount of free storage on each DB instance. The actual local storage available to you might be lower. Aurora uses some local storage for its management processes, and the DB instance uses some local storage even before you load any data. You can monitor the temporary storage available for a specific DB instance with the [FreeLocalStorage CloudWatch metric](#), described in [Amazon CloudWatch metrics for Amazon Aurora \(p. 633\)](#). You can check the amount of free storage at the present time. You can also chart the amount of free storage over time. Monitoring the free storage over time helps you to determine whether the value is increasing or decreasing, or to find the minimum, maximum, or average values.

Backtracking an Aurora DB cluster

With Amazon Aurora MySQL-Compatible Edition, you can backtrack a DB cluster to a specific time, without restoring data from a backup.

Overview of backtracking

Backtracking "rewinds" the DB cluster to the time you specify. Backtracking is not a replacement for backing up your DB cluster so that you can restore it to a point in time. However, backtracking provides the following advantages over traditional backup and restore:

- You can easily undo mistakes. If you mistakenly perform a destructive action, such as a `DELETE` without a `WHERE` clause, you can backtrack the DB cluster to a time before the destructive action with minimal interruption of service.
- You can backtrack a DB cluster quickly. Restoring a DB cluster to a point in time launches a new DB cluster and restores it from backup data or a DB cluster snapshot, which can take hours. Backtracking a DB cluster doesn't require a new DB cluster and rewinds the DB cluster in minutes.
- You can explore earlier data changes. You can repeatedly backtrack a DB cluster back and forth in time to help determine when a particular data change occurred. For example, you can backtrack a DB cluster three hours and then backtrack forward in time one hour. In this case, the backtrack time is two hours before the original time.

Note

For information about restoring a DB cluster to a point in time, see [Overview of backing up and restoring an Aurora DB cluster \(p. 491\)](#).

Backtrack window

With backtracking, there is a target backtrack window and an actual backtrack window:

- The *target backtrack window* is the amount of time you want to be able to backtrack your DB cluster. When you enable backtracking, you specify a *target backtrack window*. For example, you might specify a target backtrack window of 24 hours if you want to be able to backtrack the DB cluster one day.
- The *actual backtrack window* is the actual amount of time you can backtrack your DB cluster, which can be smaller than the target backtrack window. The actual backtrack window is based on your workload and the storage available for storing information about database changes, called *change records*.

As you make updates to your Aurora DB cluster with backtracking enabled, you generate change records. Aurora retains change records for the target backtrack window, and you pay an hourly rate for storing them. Both the target backtrack window and the workload on your DB cluster determine the number of change records you store. The workload is the number of changes you make to your DB cluster in a given amount of time. If your workload is heavy, you store more change records in your backtrack window than you do if your workload is light.

You can think of your target backtrack window as the goal for the maximum amount of time you want to be able to backtrack your DB cluster. In most cases, you can backtrack the maximum amount of time that you specified. However, in some cases, the DB cluster can't store enough change records to backtrack the maximum amount of time, and your actual backtrack window is smaller than your target. Typically, the actual backtrack window is smaller than the target when you have extremely heavy workload on your DB cluster. When your actual backtrack window is smaller than your target, we send you a notification.

When backtracking is enabled for a DB cluster, and you delete a table stored in the DB cluster, Aurora keeps that table in the backtrack change records. It does this so that you can revert back to a time before you deleted the table. If you don't have enough space in your backtrack window to store the table, the table might be removed from the backtrack change records eventually.

Backtracking time

Aurora always backtracks to a time that is consistent for the DB cluster. Doing so eliminates the possibility of uncommitted transactions when the backtrack is complete. When you specify a time for a backtrack, Aurora automatically chooses the nearest possible consistent time. This approach means that the completed backtrack might not exactly match the time you specify, but you can determine the exact time for a backtrack by using the [describe-db-cluster-backtracks](#) AWS CLI command. For more information, see [Retrieving existing backtracks \(p. 828\)](#).

Backtracking limitations

The following limitations apply to backtracking:

- Backtracking an Aurora DB cluster is available in certain AWS Regions and for specific Aurora MySQL versions only. For more information, see [Backtracking in Aurora \(p. 19\)](#).
- Backtracking is only available for DB clusters that were created with the Backtrack feature enabled. You can enable the Backtrack feature when you create a new DB cluster or restore a snapshot of a DB cluster. For DB clusters that were created with the Backtrack feature enabled, you can create a clone DB cluster with the Backtrack feature enabled. Currently, you can't perform backtracking on DB clusters that were created with the Backtrack feature disabled.
- The limit for a backtrack window is 72 hours.
- Backtracking affects the entire DB cluster. For example, you can't selectively backtrack a single table or a single data update.

- Backtracking isn't supported with binary log (binlog) replication. Cross-Region replication must be disabled before you can configure or use backtracking.
- You can't backtrack a database clone to a time before that database clone was created. However, you can use the original database to backtrack to a time before the clone was created. For more information about database cloning, see [Cloning a volume for an Aurora DB cluster \(p. 402\)](#).
- Backtracking causes a brief DB instance disruption. You must stop or pause your applications before starting a backtrack operation to ensure that there are no new read or write requests. During the backtrack operation, Aurora pauses the database, closes any open connections, and drops any uncommitted reads and writes. It then waits for the backtrack operation to complete.
- Backtracking isn't supported for the following AWS Regions:
 - Africa (Cape Town)
 - China (Ningxia)
 - Asia Pacific (Hong Kong)
 - Europe (Milan)
 - Europe (Stockholm)
 - Middle East (Bahrain)
 - South America (São Paulo)
- You can't restore a cross-Region snapshot of a backtrack-enabled cluster in an AWS Region that doesn't support backtracking.
- You can't use Backtrack with Aurora multi-master clusters.
- If you perform an in-place upgrade for a backtrack-enabled cluster from Aurora MySQL version 1 to version 2, you can't backtrack to a point in time before the upgrade happened.

Upgrade considerations for backtrack-enabled clusters

Backtracking is available for Aurora MySQL 1.*, which is compatible with MySQL 5.6. It's also available for Aurora MySQL 2.06 and higher, which is compatible with MySQL 5.7. Because of the Aurora MySQL 2.* version requirement, if you created the Aurora MySQL 1.* cluster with the Backtrack setting enabled, you can only upgrade to a Backtrack-compatible version of Aurora MySQL 2.*. This requirement affects the following types of upgrade paths:

- You can only restore a snapshot of the Aurora MySQL 1.* DB cluster to a Backtrack-compatible version of Aurora MySQL 2.*.
- You can only perform point-in-time recovery on the Aurora MySQL 1.* DB cluster to restore it to a Backtrack-compatible version of Aurora MySQL 2.*.

These upgrade requirements still apply even if you turn off Backtrack for the Aurora MySQL 1.* cluster.

Configuring backtracking

To use the Backtrack feature, you must enable backtracking and specify a target backtrack window. Otherwise, backtracking is disabled.

For the target backtrack window, specify the amount of time that you want to be able to rewind your database using Backtrack. Aurora tries to retain enough change records to support that window of time.

Console

You can use the console to configure backtracking when you create a new DB cluster. You can also modify a DB cluster to change the backtrack window for a backtrack-enabled cluster. If you turn off backtracking entirely for a cluster by setting the backtrack window to 0, you can't enable backtrack again for that cluster.

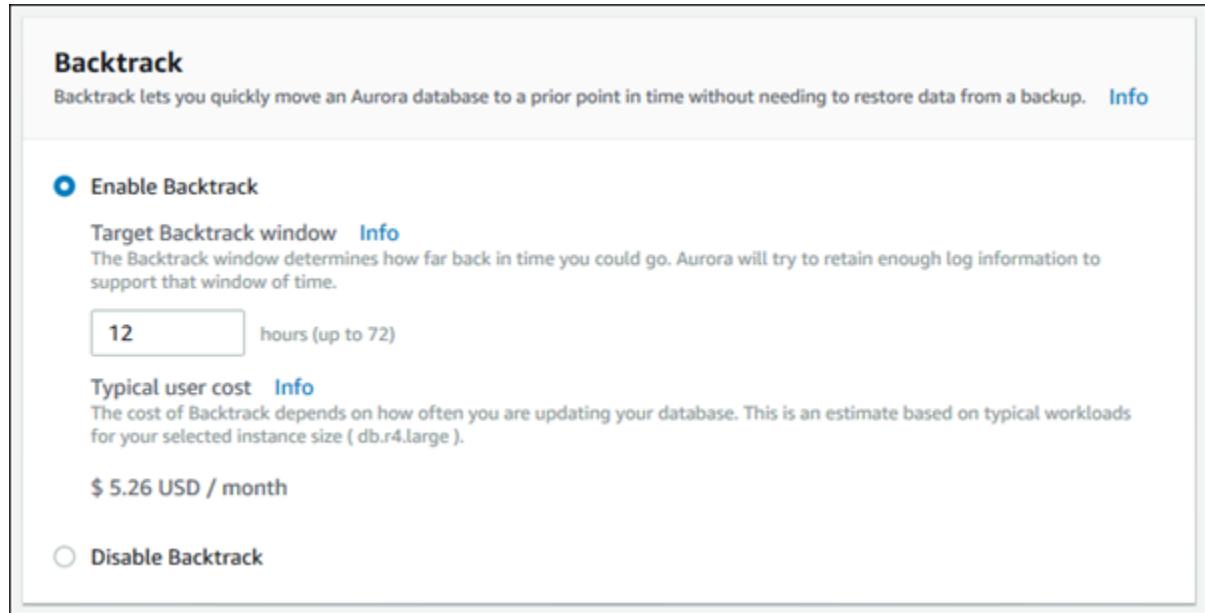
Topics

- [Configuring backtracking with the console when creating a DB cluster \(p. 819\)](#)
- [Configuring backtrack with the console when modifying a DB cluster \(p. 819\)](#)

Configuring backtracking with the console when creating a DB cluster

When you create a new Aurora MySQL DB cluster, backtracking is configured when you choose **Enable Backtrack** and specify a **Target Backtrack window** value that is greater than zero in the **Backtrack** section.

To create a DB cluster, follow the instructions in [Creating an Amazon Aurora DB cluster \(p. 125\)](#). The following image shows the **Backtrack** section.



When you create a new DB cluster, Aurora has no data for the DB cluster's workload. So it can't estimate a cost specifically for the new DB cluster. Instead, the console presents a typical user cost for the specified target backtrack window based on a typical workload. The typical cost is meant to provide a general reference for the cost of the Backtrack feature.

Important

Your actual cost might not match the typical cost, because your actual cost is based on your DB cluster's workload.

Configuring backtrack with the console when modifying a DB cluster

You can modify backtracking for a DB cluster using the console.

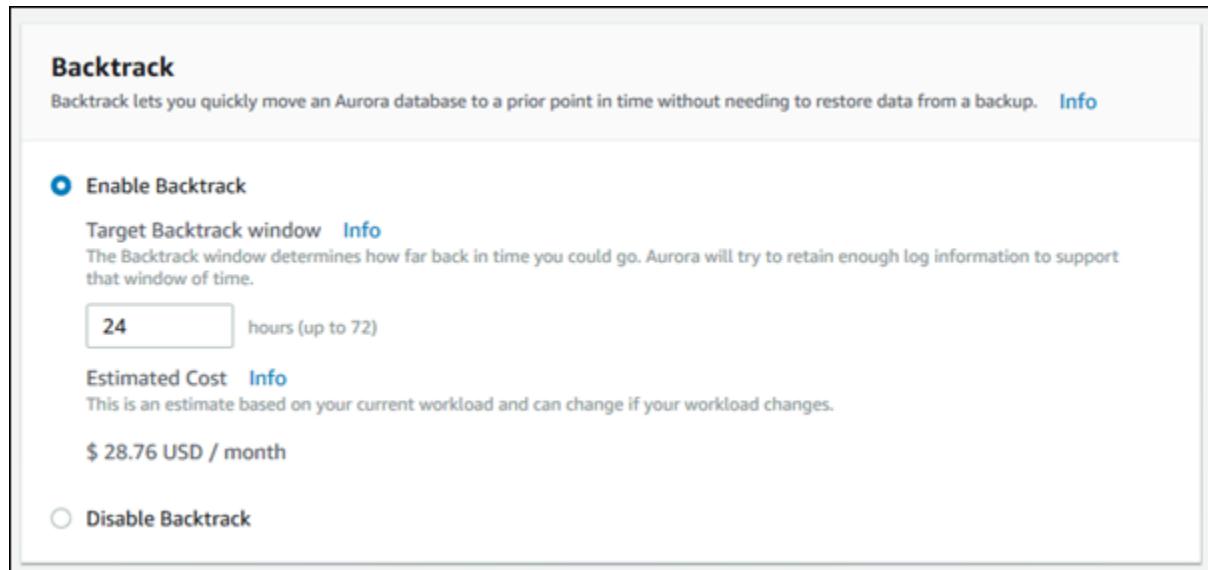
Note

Currently, you can modify backtracking only for a DB cluster that has the Backtrack feature enabled. The **Backtrack** section doesn't appear for a DB cluster that was created with the Backtrack feature disabled or if the Backtrack feature has been disabled for the DB cluster.

To modify backtracking for a DB cluster using the console

1. Sign in to the AWS Management Console and open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. Choose **Databases**.

3. Choose the cluster that you want to modify, and choose **Modify**.
4. For **Target Backtrack window**, modify the amount of time that you want to be able to backtrack. The limit is 72 hours.



The console shows the estimated cost for the amount of time you specified based on the DB cluster's past workload:

- If backtracking was disabled on the DB cluster, the cost estimate is based on the `VolumeWriteIOPS` metric for the DB cluster in Amazon CloudWatch.
 - If backtracking was enabled previously on the DB cluster, the cost estimate is based on the `BacktrackChangeRecordsCreationRate` metric for the DB cluster in Amazon CloudWatch.
5. Choose **Continue**.
 6. For **Scheduling of Modifications**, choose one of the following:
 - **Apply during the next scheduled maintenance window** – Wait to apply the **Target Backtrack window** modification until the next maintenance window.
 - **Apply immediately** – Apply the **Target Backtrack window** modification as soon as possible.
 7. Choose **Modify cluster**.

AWS CLI

When you create a new Aurora MySQL DB cluster using the `create-db-cluster` AWS CLI command, backtracking is configured when you specify a `--backtrack-window` value that is greater than zero. The `--backtrack-window` value specifies the target backtrack window. For more information, see [Creating an Amazon Aurora DB cluster \(p. 125\)](#).

You can also specify the `--backtrack-window` value using the following AWS CLI commands:

- `modify-db-cluster`
- `restore-db-cluster-from-s3`
- `restore-db-cluster-from-snapshot`
- `restore-db-cluster-to-point-in-time`

The following procedure describes how to modify the target backtrack window for a DB cluster using the AWS CLI.

To modify the target backtrack window for a DB cluster using the AWS CLI

- Call the [modify-db-cluster](#) AWS CLI command and supply the following values:
 - `--db-cluster-identifier` – The name of the DB cluster.
 - `--backtrack-window` – The maximum number of seconds that you want to be able to backtrack the DB cluster.

The following example sets the target backtrack window for `sample-cluster` to one day (86,400 seconds).

For Linux, macOS, or Unix:

```
aws rds modify-db-cluster \
--db-cluster-identifier sample-cluster \
--backtrack-window 86400
```

For Windows:

```
aws rds modify-db-cluster ^
--db-cluster-identifier sample-cluster ^
--backtrack-window 86400
```

Note

Currently, you can enable backtracking only for a DB cluster that was created with the Backtrack feature enabled.

RDS API

When you create a new Aurora MySQL DB cluster using the [CreateDBCluster](#) Amazon RDS API operation, backtracking is configured when you specify a `BacktrackWindow` value that is greater than zero. The `BacktrackWindow` value specifies the target backtrack window for the DB cluster specified in the `DBClusterIdentifier` value. For more information, see [Creating an Amazon Aurora DB cluster \(p. 125\)](#).

You can also specify the `BacktrackWindow` value using the following API operations:

- [ModifyDBCluster](#)
- [RestoreDBClusterFromS3](#)
- [RestoreDBClusterFromSnapshot](#)
- [RestoreDBClusterToPointInTime](#)

Note

Currently, you can enable backtracking only for a DB cluster that was created with the Backtrack feature enabled.

Performing a backtrack

You can backtrack a DB cluster to a specified backtrack time stamp. If the backtrack time stamp isn't earlier than the earliest possible backtrack time, and isn't in the future, the DB cluster is backtracked to that time stamp.

Otherwise, an error typically occurs. Also, if you try to backtrack a DB cluster for which binary logging is enabled, an error typically occurs unless you've chosen to force the backtrack to occur. Forcing a backtrack to occur can interfere with other operations that use binary logging.

Important

Backtracking doesn't generate binlog entries for the changes that it makes. If you have binary logging enabled for the DB cluster, backtracking might not be compatible with your binlog implementation.

Note

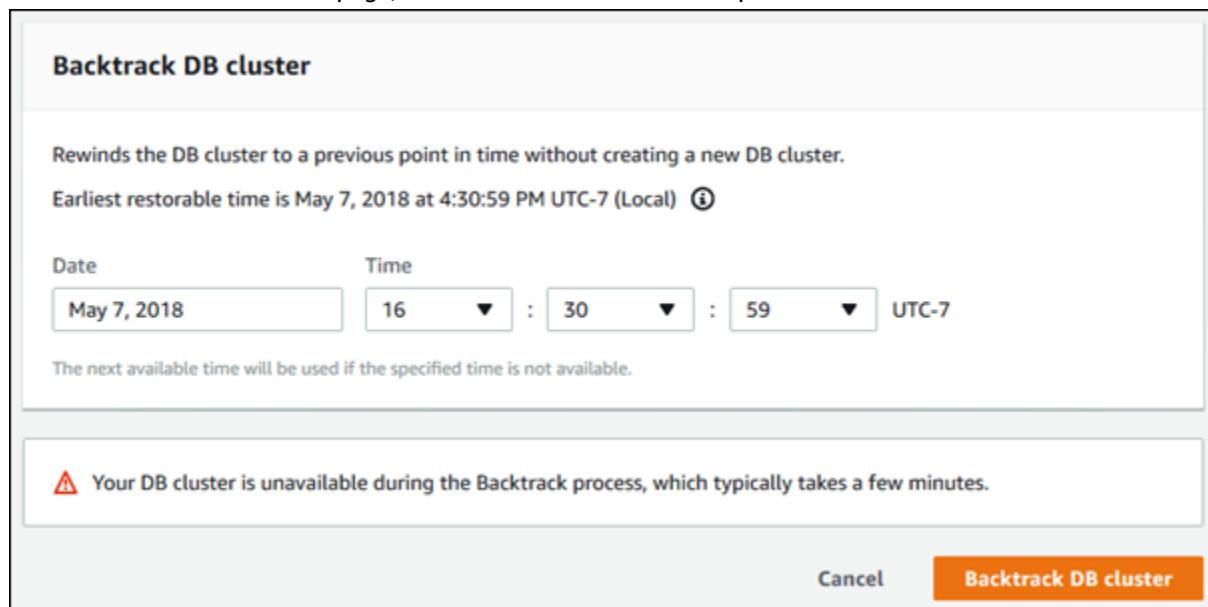
For database clones, you can't backtrack the DB cluster earlier than the date and time when the clone was created. For more information about database cloning, see [Cloning a volume for an Aurora DB cluster \(p. 402\)](#).

[Console](#)

The following procedure describes how to perform a backtrack operation for a DB cluster using the console.

To perform a backtrack operation using the console

1. Sign in to the AWS Management Console and open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. In the navigation pane, choose **Instances**.
3. Choose the primary instance for the DB cluster that you want to backtrack.
4. For **Actions**, choose **Backtrack DB cluster**.
5. On the **Backtrack DB cluster** page, enter the backtrack time stamp to backtrack the DB cluster to.



6. Choose **Backtrack DB cluster**.

[AWS CLI](#)

The following procedure describes how to backtrack a DB cluster using the AWS CLI.

To backtrack a DB cluster using the AWS CLI

- Call the `backtrack-db-cluster` AWS CLI command and supply the following values:

- `--db-cluster-identifier` – The name of the DB cluster.
- `--backtrack-to` – The backtrack time stamp to backtrack the DB cluster to, specified in ISO 8601 format.

The following example backtracks the DB cluster `sample-cluster` to March 19, 2018, at 10 a.m.

For Linux, macOS, or Unix:

```
aws rds backtrack-db-cluster \
--db-cluster-identifier sample-cluster \
--backtrack-to 2018-03-19T10:00:00+00:00
```

For Windows:

```
aws rds backtrack-db-cluster ^
--db-cluster-identifier sample-cluster ^
--backtrack-to 2018-03-19T10:00:00+00:00
```

RDS API

To backtrack a DB cluster using the Amazon RDS API, use the `BacktrackDBCluster` operation. This operation backtracks the DB cluster specified in the `DBClusterIdentifier` value to the specified time.

Monitoring backtracking

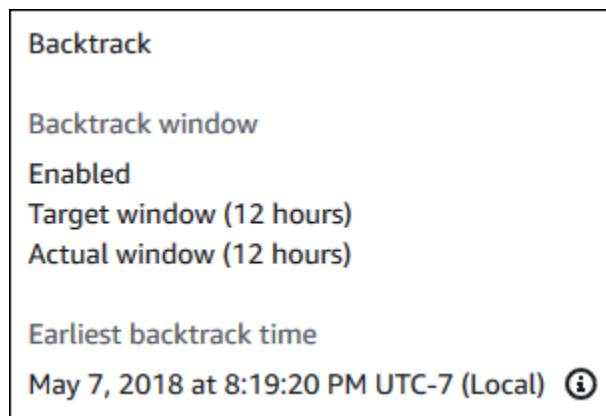
You can view backtracking information and monitor backtracking metrics for a DB cluster.

Console

To view backtracking information and monitor backtracking metrics using the console

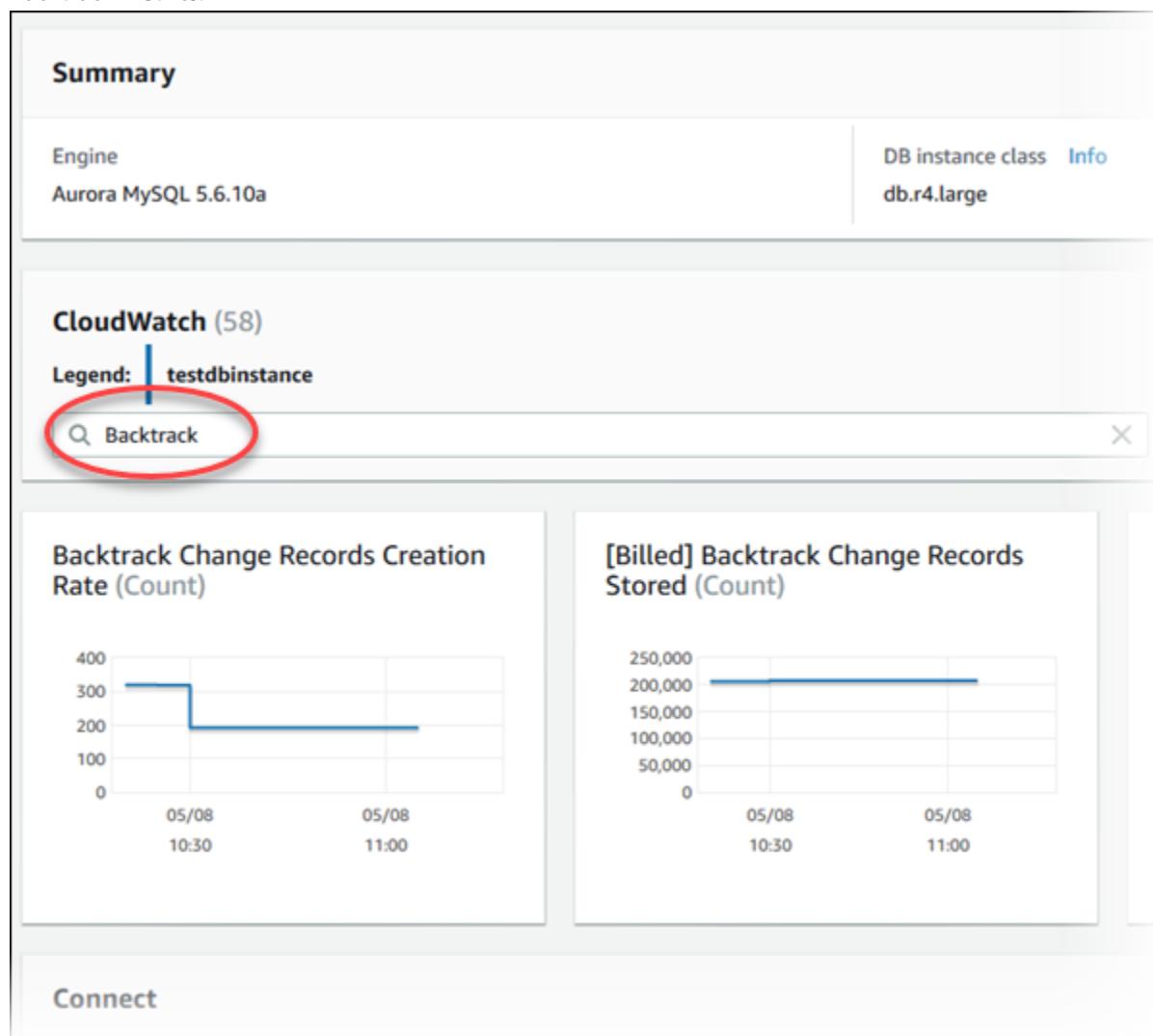
1. Sign in to the AWS Management Console and open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. Choose **Databases**.
3. Choose the DB cluster name to open information about it.

The backtrack information is in the **Backtrack** section.



When backtracking is enabled, the following information is available:

- **Target window** – The current amount of time specified for the target backtrack window. The target is the maximum amount of time that you can backtrack if there is sufficient storage.
 - **Actual window** – The actual amount of time you can backtrack, which can be smaller than the target backtrack window. The actual backtrack window is based on your workload and the storage available for retaining backtrack change records.
 - **Earliest backtrack time** – The earliest possible backtrack time for the DB cluster. You can't backtrack the DB cluster to a time before the displayed time.
4. Do the following to view backtracking metrics for the DB cluster:
- a. In the navigation pane, choose **Instances**.
 - b. Choose the name of the primary instance for the DB cluster to display its details.
 - c. In the **CloudWatch** section, type **Backtrack** into the **CloudWatch** box to show only the Backtrack metrics.



The following metrics are displayed:

- **Backtrack Change Records Creation Rate (Count)** – This metric shows the number of backtrack change records created over five minutes for your DB cluster. You can use this metric to estimate the backtrack cost for your target backtrack window.
- **[Billed] Backtrack Change Records Stored (Count)** – This metric shows the actual number of backtrack change records used by your DB cluster.
- **Backtrack Window Actual (Minutes)** – This metric shows whether there is a difference between the target backtrack window and the actual backtrack window. For example, if your target backtrack window is 2 hours (120 minutes), and this metric shows that the actual backtrack window is 100 minutes, then the actual backtrack window is smaller than the target.
- **Backtrack Window Alert (Count)** – This metric shows how often the actual backtrack window is smaller than the target backtrack window for a given period of time.

Note

The following metrics might lag behind the current time:

- **Backtrack Change Records Creation Rate (Count)**
- **[Billed] Backtrack Change Records Stored (Count)**

AWS CLI

The following procedure describes how to view backtrack information for a DB cluster using the AWS CLI.

To view backtrack information for a DB cluster using the AWS CLI

- Call the [describe-db-clusters](#) AWS CLI command and supply the following values:
 - `--db-cluster-identifier` – The name of the DB cluster.

The following example lists backtrack information for `sample-cluster`.

For Linux, macOS, or Unix:

```
aws rds describe-db-clusters \
    --db-cluster-identifier sample-cluster
```

For Windows:

```
aws rds describe-db-clusters ^
    --db-cluster-identifier sample-cluster
```

RDS API

To view backtrack information for a DB cluster using the Amazon RDS API, use the [DescribeDBClusters](#) operation. This operation returns backtrack information for the DB cluster specified in the `DBClusterIdentifier` value.

Subscribing to a backtrack event with the console

The following procedure describes how to subscribe to a backtrack event using the console. The event sends you an email or text notification when your actual backtrack window is smaller than your target backtrack window.

To view backtrack information using the console

1. Sign in to the AWS Management Console and open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. Choose **Event subscriptions**.
3. Choose **Create event subscription**.
4. In the **Name** box, type a name for the event subscription, and ensure that **Yes** is selected for **Enabled**.
5. In the **Target** section, choose **New email topic**.
6. For **Topic name**, type a name for the topic, and for **With these recipients**, enter the email addresses or phone numbers to receive the notifications.
7. In the **Source** section, choose **Instances** for **Source type**.
8. For **Instances to include**, choose **Select specific instances**, and choose your DB instance.
9. For **Event categories to include**, choose **Select specific event categories**, and choose **backtrack**.

Your page should look similar to the following page.

Create event subscription

Details

Name

Name of the Subscription.

Enabled

 Yes No

Target

Send notifications to

- ARN
- New email topic
- New SMS topic

Topic name

Name of the topic.

With these recipients

Email addresses or phone numbers of SMS enabled devices to send the notifications to

e.g. user@domain.com

Source

Source type

Source type of resource this subscription will consume event from



Instances to include

Instances that this subscription will consume events from

- All instances
- Select specific instances

Specific instances



Event categories to include

Event categories that this subscription will consume events from

- All event categories
- Select specific event categories

827

Specific event



10. Choose **Create**.

Retrieving existing backtracks

You can retrieve information about existing backtracks for a DB cluster. This information includes the unique identifier of the backtrack, the date and time backtracked to and from, the date and time the backtrack was requested, and the current status of the backtrack.

Note

Currently, you can't retrieve existing backtracks using the console.

AWS CLI

The following procedure describes how to retrieve existing backtracks for a DB cluster using the AWS CLI.

To retrieve existing backtracks using the AWS CLI

- Call the [describe-db-cluster-backtracks](#) AWS CLI command and supply the following values:
 - `--db-cluster-identifier` – The name of the DB cluster.

The following example retrieves existing backtracks for `sample-cluster`.

For Linux, macOS, or Unix:

```
aws rds describe-db-cluster-backtracks \
  --db-cluster-identifier sample-cluster
```

For Windows:

```
aws rds describe-db-cluster-backtracks ^
  --db-cluster-identifier sample-cluster
```

RDS API

To retrieve information about the backtracks for a DB cluster using the Amazon RDS API, use the [DescribeDBClusterBacktracks](#) operation. This operation returns information about backtracks for the DB cluster specified in the `DBClusterIdentifier` value.

Disabling backtracking for a DB cluster

You can disable the Backtrack feature for a DB cluster.

Console

You can disable backtracking for a DB cluster using the console. After you turn off backtracking entirely for a cluster, you can't enable it again for that cluster.

To disable the Backtrack feature for a DB cluster using the console

1. Sign in to the AWS Management Console and open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.

2. Choose **Databases**.
3. Choose the cluster you want to modify, and choose **Modify**.
4. In the **Backtrack** section, choose **Disable Backtrack**.
5. Choose **Continue**.
6. For **Scheduling of Modifications**, choose one of the following:
 - **Apply during the next scheduled maintenance window** – Wait to apply the modification until the next maintenance window.
 - **Apply immediately** – Apply the modification as soon as possible.
7. Choose **Modify Cluster**.

AWS CLI

You can disable the Backtrack feature for a DB cluster using the AWS CLI by setting the target backtrack window to 0 (zero). After you turn off backtracking entirely for a cluster, you can't enable it again for that cluster.

To modify the target backtrack window for a DB cluster using the AWS CLI

- Call the [modify-db-cluster](#) AWS CLI command and supply the following values:
 - **--db-cluster-identifier** – The name of the DB cluster.
 - **--backtrack-window** – specify 0 to turn off backtracking.

The following example disables the Backtrack feature for the `sample-cluster` by setting `--backtrack-window` to 0.

For Linux, macOS, or Unix:

```
aws rds modify-db-cluster \
--db-cluster-identifier sample-cluster \
--backtrack-window 0
```

For Windows:

```
aws rds modify-db-cluster ^
--db-cluster-identifier sample-cluster ^
--backtrack-window 0
```

RDS API

To disable the Backtrack feature for a DB cluster using the Amazon RDS API, use the [ModifyDBCluster](#) operation. Set the `BacktrackWindow` value to 0 (zero), and specify the DB cluster in the `DBClusterIdentifier` value. After you turn off backtracking entirely for a cluster, you can't enable it again for that cluster.

Testing Amazon Aurora using fault injection queries

You can test the fault tolerance of your Amazon Aurora DB cluster by using fault injection queries. Fault injection queries are issued as SQL commands to an Amazon Aurora instance and they enable you to schedule a simulated occurrence of one of the following events:

- A crash of a writer or reader DB instance
- A failure of an Aurora Replica
- A disk failure
- Disk congestion

When a fault injection query specifies a crash, it forces a crash of the Aurora DB instance. The other fault injection queries result in simulations of failure events, but don't cause the event to occur. When you submit a fault injection query, you also specify an amount of time for the failure event simulation to occur for.

You can submit a fault injection query to one of your Aurora Replica instances by connecting to the endpoint for the Aurora Replica. For more information, see [Amazon Aurora connection management \(p. 32\)](#).

Testing an instance crash

You can force a crash of an Amazon Aurora instance using the `ALTER SYSTEM CRASH` fault injection query.

For this fault injection query, a failover will not occur. If you want to test a failover, then you can choose the `Failover` instance action for your DB cluster in the RDS console, or use the `failover-db-cluster` AWS CLI command or the `FailoverDBCluster` RDS API operation.

Syntax

```
ALTER SYSTEM CRASH [ INSTANCE | DISPATCHER | NODE ];
```

Options

This fault injection query takes one of the following crash types:

- **INSTANCE** — A crash of the MySQL-compatible database for the Amazon Aurora instance is simulated.
- **DISPATCHER** — A crash of the dispatcher on the writer instance for the Aurora DB cluster is simulated. The *dispatcher* writes updates to the cluster volume for an Amazon Aurora DB cluster.
- **NODE** — A crash of both the MySQL-compatible database and the dispatcher for the Amazon Aurora instance is simulated. For this fault injection simulation, the cache is also deleted.

The default crash type is `INSTANCE`.

Testing an Aurora replica failure

You can simulate the failure of an Aurora Replica using the `ALTER SYSTEM SIMULATE READ REPLICA FAILURE` fault injection query.

An Aurora Replica failure will block all requests to an Aurora Replica or all Aurora Replicas in the DB cluster for a specified time interval. When the time interval completes, the affected Aurora Replicas will be automatically synced up with master instance.

Syntax

```
ALTER SYSTEM SIMULATE percentage_of_failure PERCENT READ REPLICA FAILURE
```

```
[ TO ALL | TO "replica name" ]
FOR INTERVAL quantity { YEAR | QUARTER | MONTH | WEEK | DAY | HOUR | MINUTE | SECOND };
```

Options

This fault injection query takes the following parameters:

- **percentage_of_failure** — The percentage of requests to block during the failure event. This value can be a double between 0 and 100. If you specify 0, then no requests are blocked. If you specify 100, then all requests are blocked.
- **Failure type** — The type of failure to simulate. Specify `TO ALL` to simulate failures for all Aurora Replicas in the DB cluster. Specify `TO` and the name of the Aurora Replica to simulate a failure of a single Aurora Replica. The default failure type is `TO ALL`.
- **quantity** — The amount of time for which to simulate the Aurora Replica failure. The interval is an amount followed by a time unit. The simulation will occur for that amount of the specified unit. For example, `20 MINUTE` will result in the simulation running for 20 minutes.

Note

Take care when specifying the time interval for your Aurora Replica failure event. If you specify too long of a time interval, and your writer instance writes a large amount of data during the failure event, then your Aurora DB cluster might assume that your Aurora Replica has crashed and replace it.

Testing a disk failure

You can simulate a disk failure for an Aurora DB cluster using the `ALTER SYSTEM SIMULATE DISK FAILURE` fault injection query.

During a disk failure simulation, the Aurora DB cluster randomly marks disk segments as faulting. Requests to those segments will be blocked for the duration of the simulation.

Syntax

```
ALTER SYSTEM SIMULATE percentage_of_failure PERCENT DISK FAILURE
[ IN DISK index | NODE index ]
FOR INTERVAL quantity { YEAR | QUARTER | MONTH | WEEK | DAY | HOUR | MINUTE | SECOND };
```

Options

This fault injection query takes the following parameters:

- **percentage_of_failure** — The percentage of the disk to mark as faulting during the failure event. This value can be a double between 0 and 100. If you specify 0, then none of the disk is marked as faulting. If you specify 100, then the entire disk is marked as faulting.
- **DISK index** — A specific logical block of data to simulate the failure event for. If you exceed the range of available logical blocks of data, you will receive an error that tells you the maximum index value that you can specify. For more information, see [Displaying volume status for an Aurora MySQL DB cluster \(p. 837\)](#).
- **NODE index** — A specific storage node to simulate the failure event for. If you exceed the range of available storage nodes, you will receive an error that tells you the maximum index value that you can specify. For more information, see [Displaying volume status for an Aurora MySQL DB cluster \(p. 837\)](#).
- **quantity** — The amount of time for which to simulate the disk failure. The interval is an amount followed by a time unit. The simulation will occur for that amount of the specified unit. For example, `20 MINUTE` will result in the simulation running for 20 minutes.

Testing disk congestion

You can simulate a disk failure for an Aurora DB cluster using the `ALTER SYSTEM SIMULATE DISK CONGESTION` fault injection query.

During a disk congestion simulation, the Aurora DB cluster randomly marks disk segments as congested. Requests to those segments will be delayed between the specified minimum and maximum delay time for the duration of the simulation.

Syntax

```
ALTER SYSTEM SIMULATE percentage_of_failure PERCENT DISK CONGESTION
  BETWEEN minimum AND maximum MILLISECONDS
  [ IN DISK index | NODE index ]
  FOR INTERVAL quantity { YEAR | QUARTER | MONTH | WEEK | DAY | HOUR | MINUTE | SECOND };
```

Options

This fault injection query takes the following parameters:

- **`percentage_of_failure`** — The percentage of the disk to mark as congested during the failure event. This value can be a double between 0 and 100. If you specify 0, then none of the disk is marked as congested. If you specify 100, then the entire disk is marked as congested.
- **`DISK index Or NODE index`** — A specific disk or node to simulate the failure event for. If you exceed the range of indexes for the disk or node, you will receive an error that tells you the maximum index value that you can specify.
- **`minimum And maximum`** — The minimum and maximum amount of congestion delay, in milliseconds. Disk segments marked as congested will be delayed for a random amount of time within the range of the minimum and maximum amount of milliseconds for the duration of the simulation.
- **`quantity`** — The amount of time for which to simulate the disk congestion. The interval is an amount followed by a time unit. The simulation will occur for that amount of the specified time unit. For example, `20 MINUTE` will result in the simulation running for 20 minutes.

Altering tables in Amazon Aurora using fast DDL

Amazon Aurora includes optimizations to run an `ALTER TABLE` operation in place, nearly instantaneously. The operation completes without requiring the table to be copied and without having a material impact on other DML statements. Because the operation doesn't consume temporary storage for a table copy, it makes DDL statements practical even for large tables on small instance classes.

Aurora MySQL version 3 is compatible with the MySQL 8.0 feature called instant DDL. Aurora MySQL versions 1 and 2 use a different implementation called fast DDL.

Topics

- [Instant DDL \(Aurora MySQL version 3\) \(p. 832\)](#)
- [Fast DDL \(Aurora MySQL version 1 and 2\) \(p. 834\)](#)

Instant DDL (Aurora MySQL version 3)

The optimization performed by Aurora MySQL version 3 to improve the efficiency of some DDL operations is called instant DDL.

Aurora MySQL version 3 is compatible with the instant DDL from community MySQL 8.0. You perform an instant DDL operation by using the clause ALGORITHM=INSTANT with the ALTER TABLE statement. For syntax and usage details about instant DDL, see [ALTER TABLE](#) and [Online DDL Operations](#) in the MySQL documentation.

The following examples demonstrate the instant DDL feature. The ALTER TABLE statements create and drop indexes, add columns, and change default column values. The examples include both regular and virtual columns, and both regular and partitioned tables. At each step, you can see the results by issuing SHOW CREATE TABLE and DESCRIBE statements.

```
mysql> CREATE TABLE t1 (a INT, b INT, KEY(b)) PARTITION BY KEY(b) PARTITIONS 6;
Query OK, 0 rows affected (0.02 sec)

mysql> ALTER TABLE t1 DROP KEY b, ADD KEY b(b) USING BTREE, ALGORITHM = INSTANT;
Query OK, 0 rows affected (0.01 sec)

mysql> ALTER TABLE t1 RENAME TO t2, ALGORITHM = INSTANT;
Query OK, 0 rows affected (0.01 sec)

mysql> ALTER TABLE t2 ALTER COLUMN b SET DEFAULT 100, ALGORITHM = INSTANT;
Query OK, 0 rows affected (0.00 sec)

mysql> ALTER TABLE t2 ALTER COLUMN b DROP DEFAULT, ALGORITHM = INSTANT;
Query OK, 0 rows affected (0.01 sec)

mysql> ALTER TABLE t2 ADD COLUMN c ENUM('a', 'b', 'c'), ALGORITHM = INSTANT;
Query OK, 0 rows affected (0.01 sec)

mysql> ALTER TABLE t2 MODIFY COLUMN c ENUM('a', 'b', 'c', 'd', 'e'), ALGORITHM = INSTANT;
Query OK, 0 rows affected (0.01 sec)

mysql> ALTER TABLE t2 ADD COLUMN (d INT GENERATED ALWAYS AS (a + 1) VIRTUAL), ALGORITHM =
INSTANT;
Query OK, 0 rows affected (0.02 sec)

mysql> ALTER TABLE t2 ALTER COLUMN a SET DEFAULT 20,
->      ALTER COLUMN b SET DEFAULT 200, ALGORITHM = INSTANT;
Query OK, 0 rows affected (0.01 sec)

mysql> CREATE TABLE t2 (a INT, b INT) PARTITION BY LIST(a)(
->      PARTITION mypart1 VALUES IN (1,3,5),
->      PARTITION MyPart2 VALUES IN (2,4,6)
-> );
Query OK, 0 rows affected (0.03 sec)

mysql> ALTER TABLE t3 ALTER COLUMN a SET DEFAULT 20, ALTER COLUMN b SET DEFAULT 200,
ALGORITHM = INSTANT;
Query OK, 0 rows affected (0.01 sec)

mysql> CREATE TABLE t4 (a INT, b INT) PARTITION BY RANGE(a)
->      (PARTITION p0 VALUES LESS THAN(100), PARTITION p1 VALUES LESS THAN(1000),
->      PARTITION p2 VALUES LESS THAN MAXVALUE);
Query OK, 0 rows affected (0.05 sec)

mysql> ALTER TABLE t4 ALTER COLUMN a SET DEFAULT 20,
->      ALTER COLUMN b SET DEFAULT 200, ALGORITHM = INSTANT;
Query OK, 0 rows affected (0.01 sec)

/* Sub-partitioning example */
mysql> CREATE TABLE ts (id INT, purchased DATE, a INT, b INT)
->      PARTITION BY RANGE( YEAR(purchased) )
->          SUBPARTITION BY HASH( TO_DAYS(purchased) )
->              SUBPARTITIONS 2 (
->                  PARTITION p0 VALUES LESS THAN (1990),
```

```
-->      PARTITION p1 VALUES LESS THAN (2000),
-->      PARTITION p2 VALUES LESS THAN MAXVALUE
-->    );
Query OK, 0 rows affected (0.10 sec)

mysql> ALTER TABLE ts ALTER COLUMN a SET DEFAULT 20,
-->      ALTER COLUMN b SET DEFAULT 200, ALGORITHM = INSTANT;
Query OK, 0 rows affected (0.01 sec)
```

Fast DDL (Aurora MySQL version 1 and 2)

In MySQL, many data definition language (DDL) operations have a significant performance impact.

For example, suppose that you use an `ALTER TABLE` operation to add a column to a table. Depending on the algorithm specified for the operation, this operation can involve the following:

- Creating a full copy of the table
- Creating a temporary table to process concurrent data manipulation language (DML) operations
- Rebuilding all indexes for the table
- Applying table locks while applying concurrent DML changes
- Slowing concurrent DML throughput

The optimization performed by Aurora MySQL version 1 and 2 to improve the efficiency of some DDL operations is called fast DDL.

In Aurora MySQL version 3, Aurora uses the MySQL 8.0 feature called instant DDL. Aurora MySQL versions 1 and 2 use a different implementation called fast DDL.

Important

Currently, Aurora lab mode must be enabled to use fast DDL for Aurora MySQL. We don't recommend using fast DDL for production DB clusters. For information about enabling Aurora lab mode, see [Amazon Aurora MySQL lab mode \(p. 1032\)](#).

Fast DDL limitations

Currently, fast DDL has the following limitations:

- Fast DDL only supports adding nullable columns, without default values, to the end of an existing table.
- Fast DDL doesn't work for partitioned tables.
- Fast DDL doesn't work for InnoDB tables that use the REDUNDANT row format.
- Fast DDL doesn't work for tables with full-text search indexes.
- If the maximum possible record size for the DDL operation is too large, fast DDL is not used. A record size is too large if it is greater than half the page size. The maximum size of a record is computed by adding the maximum sizes of all columns. For variable sized columns, according to InnoDB standards, extern bytes are not included for computation.

Note

The maximum record size check was added in Aurora 1.15.

Fast DDL syntax

```
ALTER TABLE tbl_name ADD COLUMN col_name column_definition
```

This statement takes the following options:

- **tbl_name** — The name of the table to be modified.
- **col_name** — The name of the column to be added.
- **col_definition** — The definition of the column to be added.

Note

You must specify a nullable column definition without a default value. Otherwise, fast DDL isn't used.

Fast DDL examples

The following examples demonstrate the speedup from fast DDL operations. The first SQL example runs `ALTER TABLE` statements on a large table without using fast DDL. This operation takes substantial time. A CLI example shows how to enable fast DDL for the cluster. Then another SQL example runs the same `ALTER TABLE` statements on an identical table. With fast DDL enabled, the operation is very fast.

This example uses the `ORDERS` table from the TPC-H benchmark, containing 150 million rows. This cluster intentionally uses a relatively small instance class, to demonstrate how long `ALTER TABLE` statements can take when you can't use fast DDL. The example creates a clone of the original table containing identical data. Checking the `aurora_lab_mode` setting confirms that the cluster can't use fast DDL, because lab mode isn't enabled. Then `ALTER TABLE ADD COLUMN` statements take substantial time to add new columns at the end of the table.

```
mysql> create table orders_regular_ddl like orders;
Query OK, 0 rows affected (0.06 sec)

mysql> insert into orders_regular_ddl select * from orders;
Query OK, 150000000 rows affected (1 hour 1 min 25.46 sec)

mysql> select @@aurora_lab_mode;
+-----+
| @@aurora_lab_mode |
+-----+
|          0 |
+-----+

mysql> ALTER TABLE orders_regular_ddl ADD COLUMN o_refunded boolean;
Query OK, 0 rows affected (40 min 31.41 sec)

mysql> ALTER TABLE orders_regular_ddl ADD COLUMN o_coverletter varchar(512);
Query OK, 0 rows affected (40 min 44.45 sec)
```

This example does the same preparation of a large table as the previous example. However, you can't simply enable lab mode within an interactive SQL session. That setting must be enabled in a custom parameter group. Doing so requires switching out of the `mysql` session and running some AWS CLI commands or using the AWS Management Console.

```
mysql> create table orders_fast_ddl like orders;
Query OK, 0 rows affected (0.02 sec)

mysql> insert into orders_fast_ddl select * from orders;
Query OK, 150000000 rows affected (58 min 3.25 sec)

mysql> set aurora_lab_mode=1;
ERROR 1238 (HY000): Variable 'aurora_lab_mode' is a read only variable
```

Enabling lab mode for the cluster requires some work with a parameter group. This AWS CLI example uses a cluster parameter group, to ensure that all DB instances in the cluster use the same value for the lab mode setting.

```
$ aws rds create-db-cluster-parameter-group \
--db-parameter-group-family aurora5.6 \
--db-cluster-parameter-group-name lab-mode-enabled-56 --description 'TBD'
$ aws rds describe-db-cluster-parameters \
--db-cluster-parameter-group-name lab-mode-enabled-56 \
--query '*[].[ParameterName,ParameterValue]' \
--output text | grep aurora_lab_mode
aurora_lab_mode 0
$ aws rds modify-db-cluster-parameter-group \
--db-cluster-parameter-group-name lab-mode-enabled-56 \
--parameters ParameterName=aurora_lab_mode,ParameterValue=1,ApplyMethod=pending-reboot
{
    "DBClusterParameterGroupName": "lab-mode-enabled-56"
}

# Assign the custom parameter group to the cluster that's going to use fast DDL.
$ aws rds modify-db-cluster --db-cluster-identifier tpch100g \
--db-cluster-parameter-group-name lab-mode-enabled-56
{
    "DBClusterIdentifier": "tpch100g",
    "DBClusterParameterGroup": "lab-mode-enabled-56",
    "Engine": "aurora",
    "EngineVersion": "5.6.mysql_aurora.1.22.2",
    "Status": "available"
}

# Reboot the primary instance for the cluster tpch100g:
$ aws rds reboot-db-instance --db-instance-identifier instance-2020-12-22-5208
{
    "DBInstanceIdentifier": "instance-2020-12-22-5208",
    "DBInstanceState": "rebooting"
}

$ aws rds describe-db-clusters --db-cluster-identifier tpch100g \
--query '*[].[DBClusterParameterGroup]' --output text
lab-mode-enabled-56

$ aws rds describe-db-cluster-parameters \
--db-cluster-parameter-group-name lab-mode-enabled-56 \
--query '*[].[ParameterName:ParameterName,ParameterValue:ParameterValue]' \
--output text | grep aurora_lab_mode
aurora_lab_mode 1
```

The following example shows the remaining steps after the parameter group change takes effect. It tests the `aurora_lab_mode` setting to make sure that the cluster can use fast DDL. Then it runs `ALTER TABLE` statements to add columns to the end of another large table. This time, the statements finish very quickly.

```
mysql> select @@aurora_lab_mode;
+-----+
| @@aurora_lab_mode |
+-----+
|          1         |
+-----+

mysql> ALTER TABLE orders_fast_ddl ADD COLUMN o_refunded boolean;
Query OK, 0 rows affected (1.51 sec)

mysql> ALTER TABLE orders_fast_ddl ADD COLUMN o_coverletter varchar(512);
Query OK, 0 rows affected (0.40 sec)
```

Displaying volume status for an Aurora MySQL DB cluster

In Amazon Aurora, a DB cluster volume consists of a collection of logical blocks. Each of these represents 10 gigabytes of allocated storage. These blocks are called *protection groups*.

The data in each protection group is replicated across six physical storage devices, called *storage nodes*. These storage nodes are allocated across three Availability Zones (AZs) in the AWS Region where the DB cluster resides. In turn, each storage node contains one or more logical blocks of data for the DB cluster volume. For more information about protection groups and storage nodes, see [Introducing the Aurora storage engine](#) on the AWS Database Blog.

You can simulate the failure of an entire storage node, or a single logical block of data within a storage node. To do so, you use the `ALTER SYSTEM SIMULATE DISK FAILURE` fault injection statement. For the statement, you specify the index value of a specific logical block of data or storage node. However, if you specify an index value greater than the number of logical blocks of data or storage nodes used by the DB cluster volume, the statement returns an error. For more information about fault injection queries, see [Testing Amazon Aurora using fault injection queries \(p. 829\)](#).

You can avoid that error by using the `SHOW VOLUME STATUS` statement. The statement returns two server status variables, `Disks` and `Nodes`. These variables represent the total number of logical blocks of data and storage nodes, respectively, for the DB cluster volume.

Note

The `SHOW VOLUME STATUS` statement is available for Aurora version 1.12 and later. For more information about Aurora versions, see [Database engine updates for Amazon Aurora MySQL \(p. 1082\)](#).

Syntax

```
SHOW VOLUME STATUS
```

Example

The following example illustrates a typical `SHOW VOLUME STATUS` result.

```
mysql> SHOW VOLUME STATUS;
+-----+-----+
| Variable_name | Value |
+-----+-----+
| Disks         | 96    |
| Nodes         | 74    |
+-----+-----+
```

Tuning Aurora MySQL with wait events and thread states

Wait events and thread states are an important tuning tool for Aurora MySQL. If you can find out why sessions are waiting for resources and what they are doing, you are better able to reduce bottlenecks. You can use the information in this section to find possible causes and corrective actions.

Important

The wait events and thread states in this section are specific to Aurora MySQL. Use the information in this section to tune only Amazon Aurora, not Amazon RDS for MySQL.

Some wait events in this section have no analogs in the open source versions of these database engines. Other wait events have the same names as events in open source engines, but behave differently. For example, Amazon Aurora storage works different from open source storage, so storage-related wait events indicate different resource conditions.

Topics

- [Essential concepts for Aurora MySQL tuning \(p. 838\)](#)
- [Tuning Aurora MySQL with wait events \(p. 840\)](#)
- [Tuning Aurora MySQL with thread states \(p. 876\)](#)

Essential concepts for Aurora MySQL tuning

Before you tune your Aurora MySQL database, make sure to learn what wait events and thread states are and why they occur. Also review the basic memory and disk architecture of Aurora MySQL when using the InnoDB storage engine. For a helpful architecture diagram, see the [MySQL Reference Manual](#).

Topics

- [Aurora MySQL wait events \(p. 838\)](#)
- [Aurora MySQL thread states \(p. 839\)](#)
- [Aurora MySQL memory \(p. 839\)](#)
- [Aurora MySQL processes \(p. 839\)](#)

Aurora MySQL wait events

A *wait event* indicates a resource for which a session is waiting. For example, the wait event `io/socket/sql/client_connection` indicates that a thread is in the process of handling a new connection. Typical resources that a session waits for include the following:

- Single-threaded access to a buffer, for example, when a session is attempting to modify a buffer
- A row that is currently locked by another session
- A data file read
- A log file write

For example, to satisfy a query, the session might perform a full table scan. If the data isn't already in memory, the session waits for the disk I/O to complete. When the buffers are read into memory, the session might need to wait because other sessions are accessing the same buffers. The database records the waits by using a predefined wait event. These events are grouped into categories.

A wait event doesn't by itself show a performance problem. For example, if requested data isn't in memory, reading data from disk is necessary. If one session locks a row for an update, another session waits for the row to be unlocked so that it can update it. A commit requires waiting for the write to a log file to complete. Waits are integral to the normal functioning of a database.

Large numbers of wait events typically show a performance problem. In such cases, you can use wait event data to determine where sessions are spending time. For example, if a report that typically runs in minutes now runs for hours, you can identify the wait events that contribute the most to total wait time. If you can determine the causes of the top wait events, you can sometimes make changes that improve performance. For example, if your session is waiting on a row that has been locked by another session, you can end the locking session.

Aurora MySQL thread states

A *general thread state* is a `State` value that is associated with general query processing. For example, the `sending_data` indicates that a thread is reading and filtering rows for a query to determine the correct result set.

You can use thread states to tune Aurora MySQL in a similar fashion to how you use wait events. For example, frequent occurrences of `sending_data` usually indicate that a query isn't using an index. For more information about thread states, see [General Thread States](#) in the *MySQL Reference Manual*.

When you use Performance Insights, one of the following conditions is true:

- Performance Schema is turned on – Aurora MySQL shows wait events rather than the thread state.
- Performance Schema isn't turned on – Aurora MySQL shows the thread state.

We recommend that you configure the Performance Schema for automatic management. The Performance Schema provides additional insights and better tools to investigate potential performance problems. For more information, see [Enabling the Performance Schema for Performance Insights on Aurora MySQL \(p. 580\)](#).

Aurora MySQL memory

In Aurora MySQL, the most important memory areas are the buffer pool and log buffer.

Topics

- [Buffer pool \(p. 839\)](#)

Buffer pool

The *buffer pool* is the shared memory area where Aurora MySQL caches table and index data. Queries can access frequently used data directly from memory without reading from disk.

The buffer pool is structured as a linked list of pages. A *page* can hold multiple rows. Aurora MySQL uses a least recently used (LRU) algorithm to age pages out of the pool.

For more information, see [Buffer Pool](#) in the *MySQL Reference Manual*.

Aurora MySQL processes

Aurora MySQL uses a process model that is very different from Aurora PostgreSQL.

Topics

- [MySQL server \(mysqld\) \(p. 839\)](#)
- [Threads \(p. 840\)](#)
- [Thread pool \(p. 840\)](#)

MySQL server (mysqld)

The MySQL server is a single operating-system process named mysqld. The MySQL server doesn't spawn additional processes. Thus, an Aurora MySQL database uses mysqld to perform most of its work.

When the MySQL server starts, it listens for network connections from MySQL clients. When a client connects to the database, mysqld opens a thread.

Threads

Connection manager threads associate each client connection with a dedicated thread. This thread manages authentication, runs statements, and returns results to the client. Connection manager creates new threads when necessary.

The *thread cache* is the set of available threads. When a connection ends, MySQL returns the thread to the thread cache if the cache isn't full. The `thread_cache_size` system variable determines the thread cache size.

Thread pool

The *thread pool* consists of a number of thread groups. Each group manages a set of client connections. When a client connects to the database, the thread pool assigns the connections to thread groups in round-robin fashion. The thread pool separates connections and threads. There is no fixed relationship between connections and the threads that run statements received from those connections.

Tuning Aurora MySQL with wait events

The following table summarizes the Aurora MySQL wait events that most commonly indicate performance problems. The following wait events are a subset of the list in [Aurora MySQL wait events \(p. 1063\)](#).

Wait event	Description
cpu (p. 841)	This event occurs when a thread is active in CPU or is waiting for CPU.
io/aurora_redo_log_flush (p. 844)	This event occurs when a session is writing persistent data to Aurora storage.
io/aurora_respond_to_client (p. 847)	This event occurs when a thread is waiting to return a result set to a client.
io/file/innodb/innodb_data_file (p. 849)	This event occurs when there are threads waiting on I/O operations from storage.
io/socket/sql/client_connection (p. 851)	This event occurs when a thread is in the process of handling a new connection.
io/table/sql/handler (p. 853)	This event occurs when work has been delegated to a storage engine.
synch/cond/mysys/my_thread_var::suspend (p. 856)	This event occurs when threads are suspended because they are waiting on a condition.
synch/cond/sql/MDL_context::COND_wait_status (p. 857)	This event occurs when there are threads waiting on a table metadata lock.
synch/mutex/innodb/aurora_lock_thread_slot_futex (p. 864)	This event occurs when one session has locked a row for an update, and another session tries to update the same row.
synch/mutex/innodb/buf_pool_mutex (p. 866)	This event occurs when a thread has acquired a lock on the InnoDB buffer pool to access a page in memory.
synch/mutex/innodb/fil_system_mutex (p. 868)	This event occurs when a session is waiting to access the tablespace memory cache.

Wait event	Description
synch/mutex/innodb/trx_sys_mutex (p. 871)	This event occurs when there is high database activity with a large number of transactions.
synch/rwlock/innodb/hash_table_locks (p. 872)	This event occurs when there is contention on modifying the hash table that maps the buffer cache.
synch/sxlock/innodb/hash_table_locks (p. 874)	This event occurs when pages not found in the buffer pool must be read from a file.

cpu

The `cpu` wait event occurs when a thread is active in CPU or is waiting for CPU.

Topics

- [Supported engine versions \(p. 841\)](#)
- [Context \(p. 841\)](#)
- [Likely causes of increased waits \(p. 842\)](#)
- [Actions \(p. 842\)](#)

Supported engine versions

This wait event information is supported for the following engine versions:

- Aurora MySQL version 2, up to 2.09.2
- Aurora MySQL version 1, up to 1.23.1

Context

For every vCPU, a connection can run work on this CPU. In some situations, the number of active connections that are ready to run is higher than the number of vCPUs. This imbalance results in connections waiting for CPU resources. If the number of active connections stays consistently higher than the number of vCPUs, then your instance experiences CPU contention. The contention causes the `cpu` wait event to occur.

Note

The Performance Insights metric for CPU is `DBLoadCPU`. The value for `DBLoadCPU` can differ from the value for the CloudWatch metric `CPUUtilization`. The latter metric is collected from the HyperVisor for a database instance.

Performance Insights OS metrics provide detailed information about CPU utilization. For example, you can display the following metrics:

- `os.cpuUtilization.nice.avg`
- `os.cpuUtilization.total.avg`
- `os.cpuUtilization.wait.avg`
- `os.cpuUtilization.idle.avg`

Performance Insights reports the CPU usage by the database engine as `os.cpuUtilization.nice.avg`.

Likely causes of increased waits

When this event occurs more than normal, possibly indicating a performance problem, typical causes include the following:

- Analytic queries
- Highly concurrent transactions
- Long-running transactions
- A sudden increase in the number of connections, known as a *login storm*
- An increase in context switching

Actions

If the `cpu` wait event dominates database activity, it doesn't necessarily indicate a performance problem. Respond to this event only when performance degrades.

Depending on the cause of the increase in CPU utilization, consider the following strategies:

- Increase the CPU capacity of the host. This approach typically gives only temporary relief.
- Identify top queries for potential optimization.
- Redirect some read-only workload to reader nodes, if applicable.

Topics

- [Identify the sessions or queries that are causing the problem \(p. 842\)](#)
- [Analyze and optimize the high CPU workload \(p. 843\)](#)

Identify the sessions or queries that are causing the problem

To find the sessions and queries, look at the **Top SQL** table in Performance Insights for the SQL statements that have the highest CPU load. For more information, see [Analyzing metrics with the Performance Insights dashboard \(p. 585\)](#).

Typically, one or two SQL statements consume the majority of CPU cycles. Concentrate your efforts on these statements. Suppose that your DB instance has 2 vCPUs with a DB load of 3.1 average active sessions (AAS), all in the CPU state. In this case, your instance is CPU bound. Consider the following strategies:

- Upgrade to a larger instance class with more vCPUs.
- Tune your queries to have lower CPU load.

In this example, the top SQL queries have a DB load of 1.5 AAS, all in the CPU state. Another SQL statement has a load of 0.1 in the CPU state. In this example, if you stopped the lowest-load SQL statement, you don't significantly reduce database load. However, if you optimize the two high-load queries to be twice as efficient, you eliminate the CPU bottleneck. If you reduce the CPU load of 1.5 AAS by 50 percent, the AAS for each statement decreases to 0.75. The total DB load spent on CPU is now 1.6 AAS. This value is below the maximum vCPU line of 2.0.

For a useful overview of troubleshooting using Performance Insights, see the blog post [Analyze Amazon Aurora MySQL Workloads with Performance Insights](#). Also see the AWS Support article [How can I troubleshoot and resolve high CPU utilization on my Amazon RDS for MySQL instances?](#).

Analyze and optimize the high CPU workload

After you identify the query or queries increasing CPU usage, you can either optimize them or end the connection. The following example shows how to end a connection.

```
CALL mysql.rds_kill(processID);
```

For more information, see [mysql.rds_kill](#) in the *Amazon RDS User Guide*.

If you end a session, the action might trigger a long rollback.

Follow the guidelines for optimizing queries

To optimize queries, consider the following guidelines:

- Run the `EXPLAIN` statement.

This command shows the individual steps involved in running a query. For more information, see [Optimizing Queries with EXPLAIN](#) in the MySQL documentation.

- Run the `SHOW PROFILE` statement.

Use this statement to review profile details that can indicate resource usage for statements that are run during the current session. For more information, see [SHOW PROFILE Statement](#) in the MySQL documentation.

- Run the `ANALYZE TABLE` statement.

Use this statement to refresh the index statistics for the tables accessed by the high-CPU consuming query. By analyzing the statement, you can help the optimizer choose an appropriate execution plan. For more information, see [ANALYZE TABLE Statement](#) in the MySQL documentation.

Follow the guidelines for improving CPU usage

To improve CPU usage in a database instance, follow these guidelines:

- Ensure that all queries are using proper indexes.
- Find out whether you can use Aurora parallel queries. You can use this technique to reduce CPU usage on the head node by pushing down function processing, row filtering, and column projection for the `WHERE` clause.
- Find out whether the number of SQL executions per second meets the expected thresholds.
- Find out whether index maintenance or new index creation takes up CPU cycles needed by your production workload. Schedule maintenance activities outside of peak activity times.
- Find out whether you can use partitioning to help reduce the query data set. For more information, see the blog post [How to plan and optimize Amazon Aurora with MySQL compatibility for consolidated workloads](#).

Check for connection storms

If the `DBLoadCPU` metric is not very high, but the `CPUUtilization` metric is high, the cause of the high CPU utilization lies outside of the database engine. A classic example is a connection storm.

Check whether the following conditions are true:

- There is an increase in both the Performance Insights `CPUUtilization` metric and the Amazon CloudWatch `DatabaseConnections` metric.

- The number of threads in the CPU is greater than the number of vCPUs.

If the preceding conditions are true, consider decreasing the number of database connections. For example, you can use a connection pool such as RDS Proxy. To learn the best practices for effective connection management and scaling, see the whitepaper [Amazon Aurora MySQL DBA Handbook for Connection Management](#).

io/aurora_redo_log_flush

The `io/aurora_redo_log_flush` event occurs when a session is writing persistent data to Amazon Aurora storage.

Topics

- [Supported engine versions \(p. 844\)](#)
- [Context \(p. 844\)](#)
- [Likely causes of increased waits \(p. 844\)](#)
- [Actions \(p. 845\)](#)

Supported engine versions

This wait event information is supported for the following engine versions:

- Aurora MySQL version 2.x up to 2.09.2
- Aurora MySQL version 1.x up to 1.23.1

Context

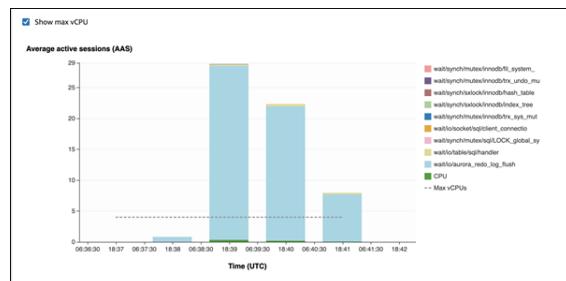
The `io/aurora_redo_log_flush` event is for a write input/output (I/O) operation in Aurora MySQL.

Likely causes of increased waits

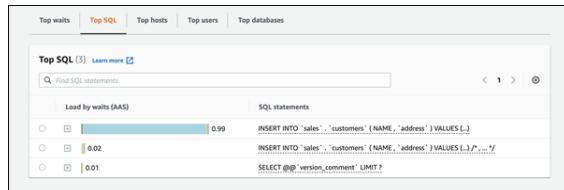
For data persistence, commits require a durable write to stable storage. If the database is doing too many commits, there is a wait event on the write I/O operation, the `io/aurora_redo_log_flush` wait event.

In the following examples, 50,000 records are inserted into an Aurora MySQL DB cluster using the db.r5.xlarge DB instance class:

- In the first example, each session inserts 10,000 records row by row. By default, if a data manipulation language (DML) command isn't within a transaction, Aurora MySQL uses implicit commits. Autocommit is turned on. This means that for each row insertion there is a commit. Performance Insights shows that the connections spend most of their time waiting on the `io/aurora_redo_log_flush` wait event.

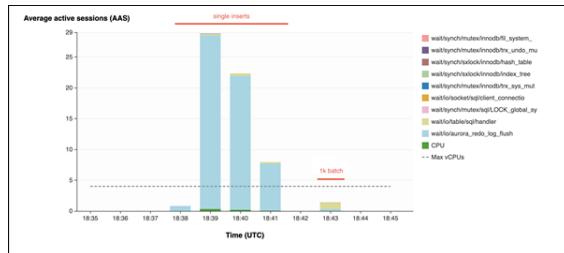


This is caused by the simple insert statements used.



The 50,000 records take 3.5 minutes to be inserted.

- In the second example, inserts are made in 1,000 batches, that is each connection performs 10 commits instead of 10,000. Performance Insights shows that the connections don't spend most of their time on the `io/aurora_redo_log_flush` wait event.



The 50,000 records take 4 seconds to be inserted.

Actions

We recommend different actions depending on the causes of your wait event.

Identify the problematic sessions and queries

If your DB instance is experiencing a bottleneck, your first task is to find the sessions and queries that cause it. For a useful AWS Database Blog post, see [Analyze Amazon Aurora MySQL Workloads with Performance Insights](#).

To identify sessions and queries causing a bottleneck

- Sign in to the AWS Management Console and open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
- In the navigation pane, choose **Performance Insights**.
- Choose your DB instance.
- In **Database load**, choose **Slice by wait**.
- At the bottom of the page, choose **Top SQL**.

The queries at the top of the list are causing the highest load on the database.

Group your write operations

The following examples trigger the `io/aurora_redo_log_flush` wait event. (Autocommit is turned on.)

```
INSERT INTO `sampleDB`.`sampleTable` (sampleCol2, sampleCol3) VALUES ('xxxx','xxxxx');
INSERT INTO `sampleDB`.`sampleTable` (sampleCol2, sampleCol3) VALUES ('xxxx','xxxxx');
INSERT INTO `sampleDB`.`sampleTable` (sampleCol2, sampleCol3) VALUES ('xxxx','xxxxx');
....
```

```

INSERT INTO `sampleDB`.`sampleTable` (sampleCol2, sampleCol3) VALUES ('xxxx', 'xxxxx');

UPDATE `sampleDB`.`sampleTable` SET sampleCol3='xxxxx' WHERE id=xx;
UPDATE `sampleDB`.`sampleTable` SET sampleCol3='xxxxx' WHERE id=xx;
UPDATE `sampleDB`.`sampleTable` SET sampleCol3='xxxxx' WHERE id=xx;
....
UPDATE `sampleDB`.`sampleTable` SET sampleCol3='xxxxx' WHERE id=xx;

DELETE FROM `sampleDB`.`sampleTable` WHERE sampleCol1=xx;
DELETE FROM `sampleDB`.`sampleTable` WHERE sampleCol1=xx;
DELETE FROM `sampleDB`.`sampleTable` WHERE sampleCol1=xx;
....
DELETE FROM `sampleDB`.`sampleTable` WHERE sampleCol1=xx;

```

To reduce the time spent waiting on the `io/aurora_redo_log_flush` wait event, group your write operations logically into a single commit to reduce persistent calls to storage.

Turn off autocommit

Turn off autocommit before making large changes that aren't within a transaction, as shown in the following example.

```

SET SESSION AUTOCOMMIT=OFF;
UPDATE `sampleDB`.`sampleTable` SET sampleCol3='xxxxx' WHERE sampleCol1=xx;
UPDATE `sampleDB`.`sampleTable` SET sampleCol3='xxxxx' WHERE sampleCol1=xx;
UPDATE `sampleDB`.`sampleTable` SET sampleCol3='xxxxx' WHERE sampleCol1=xx;
....
UPDATE `sampleDB`.`sampleTable` SET sampleCol3='xxxxx' WHERE sampleCol1=xx;
-- Other DML statements here
COMMIT;

SET SESSION AUTOCOMMIT=ON;

```

Use transactions

You can use transactions, as shown in the following example.

```

BEGIN
INSERT INTO `sampleDB`.`sampleTable` (sampleCol2, sampleCol3) VALUES ('xxxx', 'xxxxx');
INSERT INTO `sampleDB`.`sampleTable` (sampleCol2, sampleCol3) VALUES ('xxxx', 'xxxxx');
INSERT INTO `sampleDB`.`sampleTable` (sampleCol2, sampleCol3) VALUES ('xxxx', 'xxxxx');
....
INSERT INTO `sampleDB`.`sampleTable` (sampleCol2, sampleCol3) VALUES ('xxxx', 'xxxxx');

DELETE FROM `sampleDB`.`sampleTable` WHERE sampleCol1=xx;
DELETE FROM `sampleDB`.`sampleTable` WHERE sampleCol1=xx;
DELETE FROM `sampleDB`.`sampleTable` WHERE sampleCol1=xx;
....
DELETE FROM `sampleDB`.`sampleTable` WHERE sampleCol1=xx;

-- Other DML statements here
END

```

Use batches

You can make changes in batches, as shown in the following example. However, using batches that are too large can cause performance issues, especially in read replicas or when doing point-in-time recovery (PITR).

```
INSERT INTO `sampleDB`.`sampleTable` (sampleCol2, sampleCol3) VALUES
```

```
('xxxx', 'xxxxx'), ('xxxx', 'xxxxx'), ..., ('xxxx', 'xxxxx'), ('xxxx', 'xxxxx');

UPDATE `sampleDB`.`sampleTable` SET sampleCol3='xxxxx' WHERE sampleCol1 BETWEEN xx AND xxx;

DELETE FROM `sampleDB`.`sampleTable` WHERE sampleCol1<xx;
```

io/aurora_respond_to_client

The `io/aurora_respond_to_client` event occurs when a thread is waiting to return a result set to a client.

Topics

- [Supported engine versions \(p. 847\)](#)
- [Context \(p. 847\)](#)
- [Likely causes of increased waits \(p. 847\)](#)
- [Actions \(p. 848\)](#)

Supported engine versions

This wait event information is supported for the following engine versions:

- For Aurora MySQL version 2, version 2.07.7 and higher 2.07 versions, 2.09.3 and higher 2.09 versions, and 2.10.2 and higher 2.10 versions
- For Aurora MySQL version 1, version 1.22.6 and higher

In versions before version 1.22.6, 2.07.7, 2.09.3, and 2.10.2, this wait event erroneously includes idle time.

Context

The event `io/aurora_respond_to_client` indicates that a thread is waiting to return a result set to a client.

The query processing is complete, and the results are being returned back to the application client. However, because there isn't enough network bandwidth on the DB cluster, a thread is waiting to return the result set.

Likely causes of increased waits

When the `io/aurora_respond_to_client` event appears more than normal, possibly indicating a performance problem, typical causes include the following:

DB instance class insufficient for the workload

The DB instance class used by the DB cluster doesn't have the necessary network bandwidth to process the workload efficiently.

Large result sets

There was an increase in size of the result set being returned, because the query returns higher numbers of rows. The larger result set consumes more network bandwidth.

Increased load on the client

There might be CPU pressure, memory pressure, or network saturation on the client. An increase in load on the client delays the reception of data from the Aurora MySQL DB cluster.

Increased network latency

There might be increased network latency between the Aurora MySQL DB cluster and client. Higher network latency increases the time required for the client to receive the data.

Actions

We recommend different actions depending on the causes of your wait event.

Topics

- [Identify the sessions and queries causing the events \(p. 848\)](#)
- [Scale the DB instance class \(p. 848\)](#)
- [Check workload for unexpected results \(p. 848\)](#)
- [Distribute workload with reader instances \(p. 849\)](#)
- [Use the SQL_BUFFER_RESULT modifier \(p. 849\)](#)

Identify the sessions and queries causing the events

You can use Performance Insights to show queries blocked by the `io/aurora_respond_to_client` wait event. Typically, databases with moderate to significant load have wait events. The wait events might be acceptable if performance is optimal. If performance isn't optimal, then examine where the database is spending the most time. Look at the wait events that contribute to the highest load, and find out whether you can optimize the database and application to reduce those events.

To find SQL queries that are responsible for high load

1. Sign in to the AWS Management Console and open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. In the navigation pane, choose **Performance Insights**.
3. Choose a DB instance. The Performance Insights dashboard is shown for that DB instance.
4. In the **Database load** chart, choose **Slice by wait**.
5. At the bottom of the page, choose **Top SQL**.

The chart lists the SQL queries that are responsible for the load. Those at the top of the list are most responsible. To resolve a bottleneck, focus on these statements.

For a useful overview of troubleshooting using Performance Insights, see the AWS Database Blog post [Analyze Amazon Aurora MySQL Workloads with Performance Insights](#).

Scale the DB instance class

Check for the increase in the value of the Amazon CloudWatch metrics related to network throughput, such as `NetworkReceiveThroughput` and `NetworkTransmitThroughput`. If the DB instance class network bandwidth is being reached, you can scale the DB instance class used by the DB cluster by modifying the DB cluster. A DB instance class with larger network bandwidth returns data to clients more efficiently.

For information about monitoring Amazon CloudWatch metrics, see [Viewing metrics in the Amazon RDS console \(p. 563\)](#). For information about DB instance classes, see [Aurora DB instance classes \(p. 54\)](#). For information about modifying a DB cluster, see [Modifying an Amazon Aurora DB cluster \(p. 372\)](#).

Check workload for unexpected results

Check the workload on the DB cluster and make sure that it isn't producing unexpected results. For example, there might be queries that are returning a higher number of rows than expected. In this case,

you can use Performance Insights counter metrics such as `Innodb_rows_read`. For more information, see [Performance Insights counter metrics \(p. 653\)](#).

Distribute workload with reader instances

You can distribute read-only workload with Aurora replicas. You can scale horizontally by adding more Aurora replicas. Doing so can result in an increase in the throttling limits for network bandwidth. For more information, see [Amazon Aurora DB clusters \(p. 3\)](#).

Use the `SQL_BUFFER_RESULT` modifier

You can add the `SQL_BUFFER_RESULT` modifier to `SELECT` statements to force the result into a temporary table before they are returned to the client. This modifier can help with performance issues when InnoDB locks aren't being freed because queries are in the `io/aurora_respond_to_client` wait state. For more information, see [SELECT Statement](#) in the MySQL documentation.

io/file/innodb/innodb_data_file

The `io/file/innodb/innodb_data_file` event occurs when there are threads waiting on I/O operations from storage.

Topics

- [Supported engine versions \(p. 849\)](#)
- [Context \(p. 849\)](#)
- [Likely causes of increased waits \(p. 849\)](#)
- [Actions \(p. 850\)](#)

Supported engine versions

This wait event information is supported for the following engine versions:

Aurora MySQL version 1, up to 1.23.1

Context

The *InnoDB buffer pool* is the shared memory area where Aurora MySQL caches table and index data. Queries can access frequently used data directly from memory without reading from disk. The event `io/file/innodb/innodb_data_file` indicates that processing the query requires a storage I/O operation because the data isn't available in the buffer pool.

RDS typically generates this event when it performs I/O operations such as reads, writes, or flushes. RDS also generates this event when it runs data definition language (DDL) statements. This happens because these statements involve creating, deleting, opening, closing, or renaming InnoDB data files.

Likely causes of increased waits

When this event appears more than normal, possibly indicating a performance problem, typical causes include the following:

- A spike in an application workload that's I/O intensive can increase the occurrence of this wait event because more queries need to read from storage.

A significant increase in the number of pages being scanned causes least recently used (LRU) pages to be evicted from the buffer pool at a faster rate. Inefficient query plans can contribute to the problem.

Query plans can be inefficient because of outdated states, missing indexes, or inefficiently written queries.

- Storage capacity is sufficient but network throughput exceeds the maximum bandwidth for the instance class, causing I/O throttling. For information about network throughput capacity for different instance classes, see [Hardware specifications for DB instance classes for Aurora \(p. 62\)](#).
- Operations involving DDL statements or transactions that read, insert, or modify a large number of rows. For example, bulk inserts or update or delete statements can specify a wide range of values in the `WHERE` clause.
- `SELECT` queries that scan a large number of rows. For example, queries that use `BETWEEN` or `IN` clauses can specify wide ranges of data.
- A low buffer pool hit ratio because the buffer pool is too small. The smaller the buffer pool, the more frequently LRU pages are flushed out. This increases the likelihood that the requested data is read from disk.

Actions

We recommend different actions depending on the causes of your wait event.

Topics

- [Identify and optimize problem queries \(p. 850\)](#)
- [Scale up your instance \(p. 850\)](#)
- [Make your buffer scan resistant \(p. 851\)](#)

Identify and optimize problem queries

Find the query digest responsible for this wait from Performance Insights. Check the query's statement execution plan to see if the query can be optimized to read fewer pages into the InnoDB buffer pool. Doing so reduces the number of least recently used pages that are evicted from the buffer pool. This increases the cache hit efficiency of the buffer pool, which lessens the load on the I/O subsystem.

To check a query's statement execution plan, run the `EXPLAIN` statement. This command shows the individual steps involved in query execution. For more information, see [Optimizing Queries with EXPLAIN](#) in the MySQL documentation.

Scale up your instance

If your `io/file/innodb/innodb_data_file` wait events are caused by insufficient network or buffer pool capacity, consider scaling up your RDS instance to a higher instance class type.

- Network throughput – Check for an increase in the value of the Amazon CloudWatch metrics `network receive throughput` and `network transmit throughput`. If your instance has reached the network bandwidth limit for your instance class, consider scaling up your RDS instance to a higher instance class type. For more information, see [Hardware specifications for DB instance classes for Aurora \(p. 62\)](#).
- Buffer pool size – Check for a low buffer pool hit ratio. To monitor this value in Performance Insights, check the `db.Cache.innoDB_buffer_pool_hit_rate.avg` metric. To add this metric, choose **Manage metrics**, and choose `innodb_buffer_pool_hit_rate` under **Cache** on the **Database metrics** tab.

If the hit ratio is low, consider scaling up your RDS instance to a higher instance class type.

Note

The DB instance parameter that controls the buffer pool size is `innodb_buffer_pool_size`. You can modify this parameter value, but we recommend that you scale up your instance class instead because the default value is optimized for each instance class.

Make your buffer scan resistant

If you have a mix of reporting and online transaction processing (OLTP) queries, consider making your buffer pool scan resistant. To do this, tune the parameters `innodb_old_blocks_pct` and `innodb_old_blocks_time`. The effects of these parameters can vary based on your instance class hardware, data, and workload type. We highly recommend that you benchmark your system before you set these parameters in your production environment. For more information, see [Making the Buffer Pool Scan Resistant](#) in the MySQL documentation.

io/socket/sql/client_connection

The `io/socket/sql/client_connection` event occurs when a thread is in the process of handling a new connection.

Topics

- [Supported engine versions \(p. 851\)](#)
- [Context \(p. 851\)](#)
- [Likely causes of increased waits \(p. 851\)](#)
- [Actions \(p. 851\)](#)

Supported engine versions

This wait event information is supported for the following engine versions:

- Aurora MySQL version 2, up to 2.09.2
- Aurora MySQL version 1, up to 1.23.1

Context

The event `io/socket/sql/client_connection` indicates that mysqld is busy creating threads to handle incoming new client connections. In this scenario, the processing of servicing new client connection requests slows down while connections wait for the thread to be assigned. For more information, see [MySQL server \(mysqld\) \(p. 839\)](#).

Likely causes of increased waits

When this event appears more than normal, possibly indicating a performance problem, typical causes include the following:

- There is a sudden increase in new user connections from the application to your Amazon RDS instance.
- Your DB instance can't process new connections because the network, CPU, or memory is being throttled.

Actions

If `io/socket/sql/client_connection` dominates database activity, it doesn't necessarily indicate a performance problem. In a database that isn't idle, a wait event is always on top. Act only when performance degrades. We recommend different actions depending on the causes of your wait event.

Topics

- [Identify the problematic sessions and queries \(p. 852\)](#)
- [Follow best practices for connection management \(p. 852\)](#)

- [Scale up your instance if resources are being throttled \(p. 852\)](#)
- [Check the top hosts and top users \(p. 853\)](#)
- [Query the performance_schema tables \(p. 853\)](#)
- [Check the thread states of your queries \(p. 853\)](#)
- [Audit your requests and queries \(p. 853\)](#)
- [Pool your database connections \(p. 853\)](#)

Identify the problematic sessions and queries

If your DB instance is experiencing a bottleneck, your first task is to find the sessions and queries that cause it. For a useful blog post, see [Analyze Amazon Aurora MySQL Workloads with Performance Insights](#).

To identify sessions and queries causing a bottleneck

1. Sign in to the AWS Management Console and open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. In the navigation pane, choose **Performance Insights**.
3. Choose your DB instance.
4. In **Database load**, choose **Slice by wait**.
5. At the bottom of the page, choose **Top SQL**.

The queries at the top of the list are causing the highest load on the database.

Follow best practices for connection management

To manage your connections, consider the following strategies:

- Use connection pooling.

You can gradually increase the number of connections as required. For more information, see the whitepaper [Amazon Aurora MySQL Database Administrator's Handbook](#).

- Use a reader node to redistribute read-only traffic.

For more information, see [Aurora Replicas \(p. 70\)](#) and [Amazon Aurora connection management \(p. 32\)](#).

Scale up your instance if resources are being throttled

Look for examples of throttling in the following resources:

- CPU

Check your Amazon CloudWatch metrics for high CPU usage.

- Network

Check for an increase in the value of the CloudWatch metrics `network_receive_throughput` and `network_transmit_throughput`. If your instance has reached the network bandwidth limit for your instance class, consider scaling up your RDS instance to a higher instance class type. For more information, see [Aurora DB instance classes \(p. 54\)](#).

- Freeable memory

Check for a drop in the CloudWatch metric `FreeableMemory`. Also, consider turning on Enhanced Monitoring. For more information, see [Monitoring OS metrics with Enhanced Monitoring \(p. 626\)](#).

Check the top hosts and top users

Use Performance Insights to check the top hosts and top users. For more information, see [Analyzing metrics with the Performance Insights dashboard \(p. 585\)](#).

Query the performance_schema tables

To get an accurate count of the current and total connections, query the `performance_schema` tables. With this technique, you identify the source user or host that is responsible for creating a high number of connections. For example, query the `performance_schema` tables as follows.

```
SELECT * FROM performance_schema.accounts;
SELECT * FROM performance_schema.users;
SELECT * FROM performance_schema.hosts;
```

Check the thread states of your queries

If your performance issue is ongoing, check the thread states of your queries. In the `mysql` client, issue the following command.

```
show processlist;
```

Audit your requests and queries

To check the nature of the requests and queries from user accounts, use Aurora MySQL Advanced Auditing. To learn how to turn on auditing, see [Using Advanced Auditing with an Amazon Aurora MySQL DB cluster \(p. 914\)](#).

Pool your database connections

Consider using Amazon RDS Proxy for connection management. By using RDS Proxy, you can allow your applications to pool and share database connections to improve their ability to scale. RDS Proxy makes applications more resilient to database failures by automatically connecting to a standby DB instance while preserving application connections. For more information, see [Using Amazon RDS Proxy \(p. 288\)](#).

io/table/sql/handler

The `io/table/sql/handler` event occurs when work has been delegated to a storage engine.

Topics

- [Supported engine versions \(p. 853\)](#)
- [Context \(p. 854\)](#)
- [Likely causes of increased waits \(p. 854\)](#)
- [Actions \(p. 854\)](#)

Supported engine versions

This wait event information is supported for the following engine versions:

- Aurora MySQL version 2, up to 2.09.2
- Aurora MySQL version 1, up to 1.23.1

Context

The event `io/table` indicates a wait for access to a table. This event occurs regardless of whether the data is cached in the buffer pool or accessed on disk. The `io/table/sql/handler` event indicates an increase in workload activity.

A *handler* is a routine specialized in a certain type of data or focused on certain special tasks. For example, an event handler receives and digests events and signals from the operating system or from a user interface. A memory handler performs tasks related to memory. A file input handler is a function that receives file input and performs special tasks on the data, according to context.

Views such as `performance_schema.events_waits_current` often show `io/table/sql/handler` when the actual wait is a nested wait event such as a lock. When the actual wait isn't `io/table/sql/handler`, Performance Insights reports the nested wait event. When Performance Insights reports `io/table/sql/handler`, it represents the actual I/O wait and not a hidden nested wait event. For more information, see [Performance Schema Atom and Molecule Events](#) in the *MySQL Reference Manual*.

The `io/table/sql/handler` event often appears in top wait events with I/O waits such as `io/aurora_redo_log_flush` and `io/file/innodb/innodb_data_file`.

Likely causes of increased waits

In Performance Insights, sudden spikes in the `io/table/sql/handler` event indicate an increase in workload activity. Increased activity means increased I/O.

Performance Insights filters the nesting event IDs and doesn't report a `io/table/sql/handler` wait when the underlying nested event is a lock wait. For example, if the root cause event is `synch/mutex/innodb/aurora_lock_thread_slot_futex`, Performance Insights displays this wait in top wait events and not `io/table/sql/handler`.

In views such as `performance_schema.events_waits_current`, waits for `io/table/sql/handler` often appear when the actual wait is a nested wait event such as a lock. When the actual wait differs from `io/table/sql/handler`, Performance Insights looks up the nested wait and reports the actual wait instead of `io/table/sql/handler`. When Performance Insights reports `io/table/sql/handler`, the real wait is `io/table/sql/handler` and not a hidden nested wait event. For more information, see [Performance Schema Atom and Molecule Events](#) in the *MySQL 5.7 Reference Manual*.

Actions

If this wait event dominates database activity, it doesn't necessarily indicate a performance problem. A wait event is always on top when the database is active. You need to act only when performance degrades.

We recommend different actions depending on the other wait events that you see.

Topics

- [Identify the sessions and queries causing the events \(p. 854\)](#)
- [Check for a correlation with Performance Insights counter metrics \(p. 855\)](#)
- [Check for other correlated wait events \(p. 855\)](#)

[Identify the sessions and queries causing the events](#)

Typically, databases with moderate to significant load have wait events. The wait events might be acceptable if performance is optimal. If performance is isn't optimal, then examine where the database is spending the most time. Look at the wait events that contribute to the highest load, and find out whether you can optimize the database and application to reduce those events.

To find SQL queries that are responsible for high load

1. Sign in to the AWS Management Console and open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. In the navigation pane, choose **Performance Insights**.
3. Choose a DB instance. The Performance Insights dashboard is shown for that DB instance.
4. In the **Database load** chart, choose **Slice by wait**.
5. At the bottom of the page, choose **Top SQL**.

The chart lists the SQL queries that are responsible for the load. Those at the top of the list are most responsible. To resolve a bottleneck, focus on these statements.

For a useful overview of troubleshooting using Performance Insights, see the blog post [Analyze Amazon Aurora MySQL Workloads with Performance Insights](#).

Check for a correlation with Performance Insights counter metrics

Check for Performance Insights counter metrics such as `Innodb_rows_changed`. If counter metrics are correlated with `io/table/sql/handler`, follow these steps:

1. In Performance Insights, look for the SQL statements accounting for the `io/table/sql/handler` top wait event. If possible, optimize this statement so that it returns fewer rows.
2. Retrieve the top tables from the `schema_table_statistics` and `x$schema_table_statistics` views. These views show the amount of time spent per table. For more information, see [The schema_table_statistics and x\\$schema_table_statistics Views](#) in the *MySQL Reference Manual*.

By default, rows are sorted by descending total wait time. Tables with the most contention appear first. The output indicates whether time is spent on reads, writes, fetches, inserts, updates, or deletes. The following example was run on an Aurora MySQL 2.09.1 instance.

```
mysql> select * from sys.schema_table_statistics limit 1\G
***** 1. row *****
    table_schema: read_only_db
      table_name: sbtest41
    total_latency: 54.11 m
      rows_fetched: 6001557
    fetch_latency: 39.14 m
      rows_inserted: 14833
    insert_latency: 5.78 m
      rows_updated: 30470
    update_latency: 5.39 m
      rows_deleted: 14833
    delete_latency: 3.81 m
  io_read_requests: NULL
    io_read: NULL
  io_read_latency: NULL
  io_write_requests: NULL
    io_write: NULL
  io_write_latency: NULL
  io_misc_requests: NULL
  io_misc_latency: NULL
1 row in set (0.11 sec)
```

Check for other correlated wait events

If `synch/sxlock/innodb/btr_search_latch` and `io/table/sql/handler` contribute most to the DB load anomaly together, check whether the `innodb_adaptive_hash_index` variable is turned on. If it is, consider increasing the `innodb_adaptive_hash_index_parts` parameter value.

If the Adaptive Hash Index is turned off, and the situation warrants it, consider turning it on. To learn more about the MySQL Adaptive Hash Index, see the following resources:

- The article [Is Adaptive Hash Index in InnoDB right for my workload?](#) on the Percona website
- [Adaptive Hash Index](#) in the *MySQL Reference Manual*
- The article [Contention in MySQL InnoDB: Useful Info From the Semaphores Section](#) on the Percona website

The Adaptive Hash Index isn't a viable option for Aurora reader nodes. In some cases, performance might be poor on a reader node when `synch/sxlock/innodb/btr_search_latch` and `io/table/sql/handler` are dominant. If so, consider redirecting the workload temporarily to the writer node and turning on the Adaptive Hash Index.

[synch/cond/mysys/my_thread_var::suspend](#)

The `synch/cond/mysys/my_thread_var::suspend` wait event indicates that threads are suspended because they are waiting on a condition.

Topics

- [Supported engine versions \(p. 856\)](#)
- [Context \(p. 856\)](#)
- [Likely causes of increased waits \(p. 856\)](#)
- [Actions \(p. 857\)](#)

[Supported engine versions](#)

This wait event information is supported for the following versions:

- Aurora MySQL version 2 up to 2.09.2
- Aurora MySQL version 1 up to 1.23.1

[Context](#)

The event `synch/cond/mysys/my_thread_var::suspend` indicates that threads are suspended because they are waiting on a condition. For example, this wait event occurs when threads are waiting for a table-level lock. In this case, we recommend that you investigate your workload to determine which threads might be acquiring table locks on your DB instance.

[Likely causes of increased waits](#)

When the `synch/cond/mysys/my_thread_var::suspend` event appears more than normal, possibly indicating a performance problem, typical causes include the following:

Thread waiting on a table-level lock

One or more threads are waiting on a table-level lock. In this case, the thread state is `Waiting for table level lock`.

Data being sent to the mysqldump client

One or more threads are waiting because you are using `mysqldump`, and the result is being sent to the `mysqldump` client. In this case, the thread state is `Writing to net`.

Actions

We recommend different actions depending on the causes of your wait event.

Topics

- [Avoid locking tables \(p. 857\)](#)
- [Make sure that backup tools don't lock tables \(p. 857\)](#)
- [Long-running sessions that lock tables \(p. 857\)](#)
- [Non-InnoDB temporary table \(p. 857\)](#)

Avoid locking tables

Make sure that the application is not explicitly locking the tables using the `LOCK TABLE` statement. You can check the statements run by applications using Advanced Auditing. For more information, see [Using Advanced Auditing with an Amazon Aurora MySQL DB cluster \(p. 914\)](#).

Make sure that backup tools don't lock tables

If you are using a backup tool, make sure that it isn't locking tables. For example, if you are using `mysqldump`, use the `--single-transaction` option so that it doesn't lock tables.

Long-running sessions that lock tables

There might be long-running sessions that have explicitly locked tables. Run the following SQL statement to check for such sessions.

```
SELECT
p.id as session_id, p.user, p.host, p.db, p.command, p.time, p.state,
SUBSTRING(p.info, 1, 50) AS INFO,
t trx_started, unix_timestamp(now()) - unix_timestamp(t trx_started) as trx_age_seconds,
t trx_rows_modified, t trx_isolation_level
FROM information_schema.processlist p
LEFT JOIN information_schema.innodb_trx t
ON p.id = t trx_mysql_thread_id;
```

When you identify the session, your options include the following:

- Contact the application owner or the user.
- If the blocking session is idle, consider ending the blocking session. This action might trigger a long rollback. To learn how to end a session, see [Ending a session or query](#) in the *Amazon RDS User Guide*.

For more information about identifying blocking transactions, see [Using InnoDB Transaction and Locking Information](#) in the MySQL documentation.

Non-InnoDB temporary table

If you are using a non-InnoDB temporary table, then the database doesn't use row-level locking, which can result in table locks. `MyISAM` and `MEMORY` tables are examples of a non-InnoDB temporary table. If you are using a non-InnoDB temporary table, consider switching to an InnoDB memory table.

synch/cond/sql/MDL_context::COND_wait_status

The `synch/cond/sql/MDL_context::COND_wait_status` event occurs when there are threads waiting on a table metadata lock.

Topics

- [Supported engine versions \(p. 858\)](#)
- [Context \(p. 858\)](#)
- [Likely causes of increased waits \(p. 858\)](#)
- [Actions \(p. 859\)](#)

Supported engine versions

This wait event information is supported for the following engine versions:

- Aurora MySQL version 2, up to 2.09.2
- Aurora MySQL version 1, up to 1.23.1

Context

The event `synch/cond/sql/MDL_context::COND_wait_status` indicates that there are threads waiting on a table metadata lock. In some cases, one session holds a metadata lock on a table and another session tries to get the same lock on the same table. In such a case, the second session waits on the `synch/cond/sql/MDL_context::COND_wait_status` wait event.

MySQL uses metadata locking to manage concurrent access to database objects and to ensure data consistency. Metadata locking applies to tables, schemas, scheduled events, tablespaces, and user locks acquired with the `get_lock` function, and stored programs. Stored programs include procedures, functions, and triggers. For more information, see [Metadata locking](#) in the MySQL documentation.

The MySQL process list shows this session in the state `waiting for metadata lock`. In Performance Insights, if `Performance_schema` is turned on, the event `synch/cond/sql/MDL_context::COND_wait_status` appears.

The default timeout for a query waiting on a metadata lock is based on the value of the `lock_wait_timeout` parameter, which defaults to 31,536,000 seconds (365 days).

For more details on different InnoDB locks and the types of locks that can cause conflicts, see [InnoDB Locking](#) in the MySQL documentation.

Likely causes of increased waits

When the `synch/cond/sql/MDL_context::COND_wait_status` event appears more than normal, possibly indicating a performance problem, typical causes include the following:

Long-running transactions

One or more transactions are modifying a large amount of data and holding locks on tables for a very long time.

Idle transactions

One or more transactions remain open for a long time, without being committed or rolled back.

DDL statements on large tables

One or more data definition statements (DDL) statements, such as `ALTER TABLE` commands, were run on very large tables.

Explicit table locks

There are explicit locks on tables that aren't being released in a timely manner. For example, an application might run `LOCK TABLE` statements improperly.

Actions

We recommend different actions depending on the causes of your wait event and on the version of the Aurora MySQL DB cluster.

Topics

- [Identify the sessions and queries causing the events \(p. 859\)](#)
- [Check for past events \(p. 859\)](#)
- [Run queries on Aurora MySQL version 1 \(p. 860\)](#)
- [Run queries on Aurora MySQL version 2 \(p. 862\)](#)
- [Respond to the blocking session \(p. 864\)](#)

Identify the sessions and queries causing the events

You can use Performance Insights to show queries blocked by the `synch/cond/sql/MDI_context::COND_wait_status` wait event. However, to identify the blocking session, query metadata tables from `performance_schema` and `information_schema` on the DB cluster.

Typically, databases with moderate to significant load have wait events. The wait events might be acceptable if performance is optimal. If performance isn't optimal, then examine where the database is spending the most time. Look at the wait events that contribute to the highest load, and find out whether you can optimize the database and application to reduce those events.

To find SQL queries that are responsible for high load

1. Sign in to the AWS Management Console and open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. In the navigation pane, choose **Performance Insights**.
3. Choose a DB instance. The Performance Insights dashboard for that DB instance appears.
4. In the **Database load** chart, choose **Slice by wait**.
5. At the bottom of the page, choose **Top SQL**.

The chart lists the SQL queries that are responsible for the load. Those at the top of the list are most responsible. To resolve a bottleneck, focus on these statements.

For a useful overview of troubleshooting using Performance Insights, see the AWS Database Blog post [Analyze Amazon Aurora MySQL Workloads with Performance Insights](#).

Check for past events

You can gain insight into this wait event to check for past occurrences of it. To do so, complete the following actions:

- Check the data manipulation language (DML) and DDL throughput and latency to see if there were any changes in workload.

You can use Performance Insights to find queries waiting on this event at the time of the issue. Also, you can view the digest of the queries run near the time of issue.

- If audit logs or general logs are turned on for the DB cluster, you can check for all queries run on the objects (schema.table) involved in the waiting transaction. You can also check for the queries that completed running before the transaction.

The information available to troubleshoot past events is limited. Performing these checks doesn't show which object is waiting for information. However, you can identify tables with heavy load at the time of

the event and the set of frequently operated rows causing conflict at the time of issue. You can then use this information to reproduce the issue in a test environment and provide insights about its cause.

Run queries on Aurora MySQL version 1

In Aurora MySQL version 1, you can query tables in `information_schema` and `performance_schema` to identify a blocking session. To run the queries, make sure that the DB cluster is configured with the `performance_schema consumer events_statements_history`. Also, maintain an adequate number of queries in `events_statements_history` table in `performance_schema`. You control the number of queries maintained in that table with the `performance_schema_events_statements_history_size` parameter. If the required data isn't available in `performance_schema`, you can check the audit logs or general logs.

An example can illustrate how to query tables to identify blocking queries and sessions. In this example, every session runs fewer than 10 statements and required consumers are enabled on the DB cluster.

In the following process list output, process ID 59 (running the `TRUNCATE` command) and process ID 53 (running the `INSERT` command) have been waiting on a metadata lock for 33 seconds. Also, both of the threads are running queries on same table named `sbtest.sbtest1`.

```
MySQL [(none)]> select @@version, @@aurora_version;
+-----+-----+
| @@version | @@aurora_version |
+-----+-----+
| 5.6.10   | 1.23.0      |
+-----+-----+
1 row in set (0.00 sec)

MySQL [performance_schema]> select * from setup_consumers where
  name='events_statements_history';
+-----+-----+
| NAME          | ENABLED |
+-----+-----+
| events_statements_history | YES    |
+-----+-----+
1 row in set (0.00 sec)

MySQL [performance_schema]> show global variables like
  'performance_schema_events_statements_history_size';
+-----+-----+
| Variable_name           | Value  |
+-----+-----+
| performance_schema_events_statements_history_size | 10     |
+-----+-----+
1 row in set (0.00 sec)

MySQL [performance_schema]> show processlist;
+-----+-----+-----+-----+-----+-----+-----+
| Id | User          | Host          | db       | Command | Time | State  |
| Info                                     |             |           |          |          |      |        |
+-----+-----+-----+-----+-----+-----+-----+
+
| 11 | rdsadmin     | localhost     | NULL    | Sleep   | 0   |       |
| cleaned up                                |           |          |          |          |      |        |
+-----+-----+-----+-----+-----+-----+-----+
+
| 14 | rdsadmin     | localhost     | NULL    | Sleep   | 1   |       |
| cleaned up                                |           |          |          |          |      |        |
+-----+-----+-----+-----+-----+-----+-----+
```

15 rdsadmin cleaned up	localhost	NULL	Sleep	14	
16 rdsadmin cleaned up	localhost	NULL	Sleep	1	
17 rdsadmin cleaned up	localhost	NULL	Sleep	214	
40 auroramysql156123 172.31.21.51:44876 sbtest123 sleep	select sleep(10000)	Query	1843 User		
41 auroramysql156123 172.31.21.51:44878 performance_schema show processlist		Query	0 init		
48 auroramysql156123 172.31.21.51:44894 sbtest123 delayed commit ok initiated	COMMIT	Execute	0		
49 auroramysql156123 172.31.21.51:44899 sbtest123 delayed commit ok initiated	COMMIT	Execute	0		
50 auroramysql156123 172.31.21.51:44896 sbtest123 delayed commit ok initiated	COMMIT	Execute	0		
51 auroramysql156123 172.31.21.51:44892 sbtest123 delayed commit ok initiated	COMMIT	Execute	0		
52 auroramysql156123 172.31.21.51:44898 sbtest123 delayed commit ok initiated	COMMIT	Execute	0		
53 auroramysql156123 172.31.21.51:44902 sbtest Waiting for table metadata lock INSERT INTO sbtest1 (id, k, c, pad) VALUES (0, 5021, '91560616281-61537173720-56678788409-8805377477		Query	33		
56 auroramysql156123 172.31.21.51:44908 NULL sleep	select sleep(10000)	Query	118 User		
58 auroramysql156123 172.31.21.51:44912 NULL cleaned up	NULL	Sleep	41		
59 auroramysql156123 172.31.21.51:44914 NULL Waiting for table metadata lock truncate table sbtest.sbtest1		Query	33		
+-----+-----+-----+-----+					
+-----+-----+-----+-----+					
+-----+-----+-----+-----+					
+-----+-----+-----+-----+					
16 rows in set (0.00 sec)					

Given this output, run the following query. This query identifies transactions that have been running for longer than 33 seconds with connection ID 59 waiting for a lock on a table for same amount of time.

```
MySQL [performance_schema]> select
    b.id,
    a trx_id,
    a trx_state,
    a trx_started,
    TIMESTAMPDIFF(SECOND,a trx_started, now()) as "Seconds Transaction Has Been Open",
    a trx_rows_modified,
    b.USER,
    b.host,
    b.db,
    b.command,
    b.time,
    b.state
```

```

from information_schema.innodb_trx a,
      information_schema.processlist b
     where a trx_mysql_thread_id=b.id
       and TIMESTAMPDIFF(SECOND,a.trx_started, now()) > 33 order by trx_started;
+-----+-----+-----+-----+
+-----+-----+-----+-----+
+-----+
| id | trx_id | trx_state | trx_started           | Seconds Transaction Has Been Open | |
| trx_rows_modified | USER          | host                | db        | command | time   |
| state             |               |                     |           |          |          |
+-----+-----+-----+-----+
+-----+-----+-----+-----+
| 40 | 1907737 | RUNNING    | 2021-02-02 12:58:16 |                               1955 |
|      0 | auroramysql56123 | 172.31.21.51:44876 | sbtest123 | Query    | 1955 | User
| sleep |           |                     |           |          |          |
| 56 | 3797992 | RUNNING    | 2021-02-02 13:27:01 |                               230 |
|      0 | auroramysql56123 | 172.31.21.51:44908 | NULL      | Query    | 230 | User
| sleep |           |                     |           |          |          |
| 58 | 3895074 | RUNNING    | 2021-02-02 13:28:18 |                               153 |
|      0 | auroramysql56123 | 172.31.21.51:44912 | NULL      | Sleep    | 153 | User
| cleaned up |           |                     |           |          |          |
+-----+-----+-----+-----+
+-----+-----+-----+-----+
+-----+
3 rows in set (0.00 sec)

```

In the output, processes 40, 56, and 58 have been active for long time. Let's identify queries run by these sessions on the `sbtest.sbtest1` table.

```

MySQL [performance_schema]> select
      t.processlist_id,
      t.thread_id,
      sql_text
     from performance_schema.threads t
    join events_statements_history sh
      on t.thread_id=sh.thread_id
   where processlist_id in (40,56,58)
   and SQL_TEXT like '%sbtest1%' order by 1;
+-----+-----+-----+
| processlist_id | thread_id | sql_text          |
+-----+-----+-----+
|      56 |      84 | select * from sbtest123.sbtest10 limit 1 |
|      58 |      86 | select * from sbtest.sbtest1 limit 1      |
+-----+-----+-----+
2 rows in set (0.01 sec)

```

In this output, the session with a `processlist_id` of 58 ran a query on the table and holds an open transaction. That open transaction is blocking the `TRUNCATE` command.

Run queries on Aurora MySQL version 2

In Aurora MySQL version 2, you can identify the blocked session directly by querying `performance_schema` tables or `sys` schema views. An example can illustrate how to query tables to identify blocking queries and sessions.

In the following process list output, the connection ID 89 is waiting on a metadata lock, and it's running a `TRUNCATE TABLE` command. In a query on the `performance_schema` tables or `sys` schema views, the output shows that the blocking session is 76.

```

MySQL [(none)]> select @@version, @@aurora_version;
+-----+-----+

```

```

| @@version | @@aurora_version |
+-----+-----+
| 5.7.12 | 2.09.0 |
+-----+
1 row in set (0.01 sec)

MySQL [(none)]> show processlist;
+----+-----+-----+-----+-----+-----+-----+
| Id | User      | Host     | db    | Command | Time | State
|   | Info       |          |       |          |      |      |
+----+-----+-----+-----+-----+-----+-----+
| 2 | rdsadmin  | localhost | NULL  | Sleep   | 0   | NULL
| 4 | rdsadmin  | localhost | NULL  | Sleep   | 2   | NULL
| 5 | rdsadmin  | localhost | NULL  | Sleep   | 1   | NULL
| 20 | rdsadmin | localhost | NULL  | Sleep   | 0   | NULL
| 21 | rdsadmin | localhost | NULL  | Sleep   | 261 | NULL
| 66 | auroramysql5712 | 172.31.21.51:52154 | sbtest123 | Sleep   | 0   | NULL
| 67 | auroramysql5712 | 172.31.21.51:52158 | sbtest123 | Sleep   | 0   | NULL
| 68 | auroramysql5712 | 172.31.21.51:52150 | sbtest123 | Sleep   | 0   | NULL
| 69 | auroramysql5712 | 172.31.21.51:52162 | sbtest123 | Sleep   | 0   | NULL
| 70 | auroramysql5712 | 172.31.21.51:52160 | sbtest123 | Sleep   | 0   | NULL
| 71 | auroramysql5712 | 172.31.21.51:52152 | sbtest123 | Sleep   | 0   | NULL
| 72 | auroramysql5712 | 172.31.21.51:52156 | sbtest123 | Sleep   | 0   | NULL
| 73 | auroramysql5712 | 172.31.21.51:52164 | sbtest123 | Sleep   | 0   | NULL
| 74 | auroramysql5712 | 172.31.21.51:52166 | sbtest123 | Sleep   | 0   | NULL
| 75 | auroramysql5712 | 172.31.21.51:52168 | sbtest123 | Sleep   | 0   | NULL
| 76 | auroramysql5712 | 172.31.21.51:52170 | NULL    | Query   | 0   | starting
|       | show processlist |
| 88 | auroramysql5712 | 172.31.21.51:52194 | NULL    | Query   | 22  | User sleep
|       | select sleep(10000) |
| 89 | auroramysql5712 | 172.31.21.51:52196 | NULL    | Query   | 5   | Waiting for
|       | table metadata lock | truncate table sbtest.sbtest1 |
+----+-----+-----+-----+-----+-----+-----+
18 rows in set (0.00 sec)

```

Next, a query on the `performance_schema` tables or `sys` schema views shows that the blocking session is 76.

```

MySQL [(none)]> select * from sys.schema_table_lock_waits;
+-----+-----+-----+
+-----+-----+-----+
+-----+-----+-----+
+-----+-----+-----+
+-----+-----+-----+

```

```

| object_schema | object_name | waiting_thread_id | waiting_pid | waiting_account
| waiting_lock_type | waiting_lock_duration | waiting_query
| waiting_query_secs | waiting_query_rows_affected | waiting_query_rows_examined |
blocking_thread_id | blocking_pid | blocking_account | blocking_lock_type |
blocking_lock_duration | sql_kill_blocking_query | sql_kill_blocking_connection |
+-----+-----+-----+
+-----+-----+-----+
+-----+-----+-----+
+-----+-----+-----+
| sbtest      | sbtest1      |           121 |          89 | auroramysql5712@192.0.2.0
| EXCLUSIVE    | TRANSACTION   |           0 |          0 |          108
|           10 |           0 | auroramysql5712@192.0.2.0 | SHARED_READ   | TRANSACTION
| KILL QUERY 76 | KILL 76       |           0 |           0 |
+-----+-----+-----+
+-----+-----+-----+
+-----+-----+-----+
+-----+-----+-----+
1 row in set (0.00 sec)

```

Respond to the blocking session

When you identify the session, your options include the following:

- Contact the application owner or the user.
- If the blocking session is idle, consider ending the blocking session. This action might trigger a long rollback. To learn how to end a session, see [Ending a session or query](#) in the *Amazon RDS User Guide*.

For more information about identifying blocking transactions, see [Using InnoDB Transaction and Locking Information](#) in the MySQL documentation.

[synch/mutex/innodb/aurora_lock_thread_slot_futex](#)

The `synch/mutex/innodb/aurora_lock_thread_slot_futex` event occurs when one session has locked a row for an update, and another session tries to update the same row. For more information, see [InnoDB locking](#) in the *MySQL Reference*.

Supported engine versions

This wait event information is supported for the following engine versions:

- Aurora MySQL version 2, up to 2.09.2
- Aurora MySQL version 1, up to 1.23.1

Likely causes of increased waits

Multiple data manipulation language (DML) statements are accessing the same row or rows simultaneously.

Actions

We recommend different actions depending on the other wait events that you see.

Topics

- [Find and respond to the SQL statements responsible for this wait event \(p. 865\)](#)
- [Find and respond to the blocking session \(p. 865\)](#)

Find and respond to the SQL statements responsible for this wait event

Use Performance Insights to identify the SQL statements responsible for this wait event. Consider the following strategies:

- If row locks are a persistent problem, consider rewriting the application to use optimistic locking.
- Use multirow statements.
- Spread the workload over different database objects. One way to do achieve this is through partitioning.
- Check the value of the `innodb_lock_wait_timeout` parameter. It controls how long transactions wait before generating a timeout error.

For a useful overview of troubleshooting using Performance Insights, see the blog post [Analyze Amazon Aurora MySQL Workloads with Performance Insights](#).

Find and respond to the blocking session

Determine whether the blocking session is idle or active. Also, find out whether the session comes from an application or an active user.

To identify the session holding the lock, you can run `SHOW ENGINE INNODB STATUS`. The following example shows sample output.

```
mysql> SHOW ENGINE INNODB STATUS;
-----TRANSACTION 302631452, ACTIVE 2 sec starting index read
mysql tables in use 1, locked 1
LOCK WAIT 2 lock struct(s), heap size 376, 1 row lock(s)
MySQL thread id 80109, OS thread handle 0x2ae915060700, query id 938819 10.0.4.12 reinvent
    updating
UPDATE sbtest1 SET k=k+1 WHERE id=503
----- TRX HAS BEEN WAITING 2 SEC FOR THIS LOCK TO BE GRANTED:
RECORD LOCKS space id 148 page no 11 n bits 30 index `PRIMARY` of table
`sysbench2`.`sbtest1` trx id 302631452 lock_mode X locks rec but not gap waiting
Record lock, heap no 30 PHYSICAL RECORD: n_fields 6; compact format; info bits 0
```

Or you can use the following query to extract details on current locks.

```
mysql> SELECT p1.id waiting_thread,
        p1.user waiting_user,
        p1.host waiting_host,
        it1.trx_query waiting_query,
        ilw.requesting_trx_id waiting_transaction,
        ilw.blocking_lock_id blocking_lock,
        il.lock_mode blocking_mode,
        il.lock_type blocking_type,
        ilw.blocking_trx_id blocking_transaction,
        CASE it trx_state
            WHEN 'LOCK WAIT'
            THEN it trx_state
            ELSE p.state
        END blocker_state,
        il.lock_table locked_table,
        it.trx_mysql_thread_id blocker_thread,
```

```
        p.user blocker_user,
        p.host blocker_host
    FROM information_schema.innodb_lock_waits ilw
    JOIN information_schema.innodb_locks il
        ON ilw.blocking_lock_id = il.lock_id
        AND ilw.blocking_trx_id = il.lock_trx_id
    JOIN information_schema.innodb_trx it
        ON ilw.blocking_trx_id = it trx_id
    JOIN information_schema.processlist p
        ON it trx_mysql_thread_id = p.id
    JOIN information_schema.innodb_trx it1
        ON ilw.requesting_trx_id = it1 trx_id
    JOIN information_schema.processlist p1
        ON it1 trx_mysql_thread_id = p1.id\G

***** 1. row *****
waiting_thread: 3561959471
waiting_user: reinvent
waiting_host: 123.456.789.012:20485
waiting_query: select id1 from test.t1 where id1=1 for update
waiting_transaction: 312337314
blocking_lock: 312337287:261:3:2
blocking_mode: X
blocking_type: RECORD
blocking_transaction: 312337287
blocker_state: User sleep
locked_table: `test`.`t1`
blocker_thread: 3561223876
blocker_user: reinvent
blocker_host: 123.456.789.012:17746
1 row in set (0.04 sec)
```

When you identify the session, your options include the following:

- Contact the application owner or the user.
- If the blocking session is idle, consider ending the blocking session. This action might trigger a long rollback. To learn how to end a session, see [Ending a session or query](#) in the *Amazon RDS User Guide*.

For more information about identifying blocking transactions, see [Using InnoDB Transaction and Locking Information](#) in the *MySQL Reference Manual*.

synch/mutex/innodb/buf_pool_mutex

The `synch/mutex/innodb/buf_pool_mutex` event occurs when a thread has acquired a lock on the InnoDB buffer pool to access a page in memory.

Topics

- [Relevant engine versions \(p. 866\)](#)
- [Context \(p. 867\)](#)
- [Likely causes of increased waits \(p. 867\)](#)
- [Actions \(p. 867\)](#)

Relevant engine versions

This wait event information is supported for the following engine versions:

- Aurora MySQL version 2.x up to 2.09.2

- Aurora MySQL version 1.x up to 1.23.1

Context

The `buf_pool` mutex is a single mutex that protects the control data structures of the buffer pool.

For more information, see [Monitoring InnoDB Mutex Waits Using Performance Schema](#) in the MySQL documentation.

Likely causes of increased waits

This is a workload-specific wait event. Common causes for `synch/mutex/innodb/buf_pool_mutex` to appear among the top wait events include the following:

- The buffer pool size isn't large enough to hold the working set of data.
- The workload is more specific to certain pages from a specific table in the database, leading to contention in the buffer pool.

Actions

We recommend different actions depending on the causes of your wait event.

Identify the sessions and queries causing the events

Typically, databases with moderate to significant load have wait events. The wait events might be acceptable if performance is optimal. If performance isn't optimal, then examine where the database is spending the most time. Look at the wait events that contribute to the highest load, and find out whether you can optimize the database and application to reduce those events.

To view the Top SQL chart in the AWS Management Console

1. Open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. In the navigation pane, choose **Performance Insights**.
3. Choose a DB instance. The Performance Insights dashboard is shown for that DB instance.
4. In the **Database load** chart, choose **Slice by wait**.
5. Underneath the **Database load** chart, choose **Top SQL**.

The chart lists the SQL queries that are responsible for the load. Those at the top of the list are most responsible. To resolve a bottleneck, focus on these statements.

For a useful overview of troubleshooting using Performance Insights, see the blog post [Analyze Amazon Aurora MySQL Workloads with Performance Insights](#).

Use Performance Insights

This event is related to workload. You can use Performance Insights to do the following:

- Identify when wait events start, and whether there's any change in the workload around that time from the application logs or related sources.
- Identify the SQL statements responsible for this wait event. Examine the execution plan of the queries to make sure that these queries are optimized and using appropriate indexes.

If the top queries responsible for the wait event are related to the same database object or table, then consider partitioning that object or table.

Create Aurora Replicas

You can create Aurora Replicas to serve read-only traffic. You can also use Aurora Auto Scaling to handle surges in read traffic. Make sure to run scheduled read-only tasks and logical backups on Aurora Replicas.

For more information, see [Using Amazon Aurora Auto Scaling with Aurora replicas \(p. 427\)](#).

Examine the buffer pool size

Check whether the buffer pool size is sufficient for the workload by looking at the metric `innodb_buffer_pool_wait_free`. If the value of this metric is high and increasing continuously, that indicates that the size of the buffer pool isn't sufficient to handle the workload. If `innodb_buffer_pool_size` has been set properly, the value of `innodb_buffer_pool_wait_free` should be small. For more information, see [Innodb_buffer_pool_wait_free](#) in the MySQL documentation.

Increase the buffer pool size if the DB instance has enough memory for session buffers and operating-system tasks. If it doesn't, change the DB instance to a larger DB instance class to get additional memory that can be allocated to the buffer pool.

Note

Aurora MySQL automatically adjusts the value of `innodb_buffer_pool_instances` based on the configured `innodb_buffer_pool_size`.

Monitor the global status history

By monitoring the change rates of status variables, you can detect locking or memory issues on your DB instance. Turn on Global Status History (GoSH) if it isn't already turned on. For more information on GoSH, see [Managing the global status history](#).

You can also create custom Amazon CloudWatch metrics to monitor status variables. For more information, see [Publishing custom metrics](#).

synch/mutex/innodb/fil_system_mutex

The `synch/mutex/innodb/fil_system_mutex` event occurs when a session is waiting to access the tablespace memory cache.

Topics

- [Supported engine versions \(p. 868\)](#)
- [Context \(p. 868\)](#)
- [Likely causes of increased waits \(p. 869\)](#)
- [Actions \(p. 869\)](#)

Supported engine versions

This wait event information is supported for the following engine versions:

- Aurora MySQL version 2, up to 2.09.2
- Aurora MySQL version 1, up to 1.23.1

Context

InnoDB uses tablespaces to manage the storage area for tables and log files. The *tablespace memory cache* is a global memory structure that maintains information about tablespaces. MySQL uses `synch`

`mutex/innodb/fil_system_mutex` waits to control concurrent access to the tablespace memory cache.

The event `synch/mutex/innodb/fil_system_mutex` indicates that there is currently more than one operation that needs to retrieve and manipulate information in the tablespace memory cache for the same tablespace.

Likely causes of increased waits

When the `synch/mutex/innodb/fil_system_mutex` event appears more than normal, possibly indicating a performance problem, this typically occurs when all of the following conditions are present:

- An increase in concurrent data manipulation language (DML) operations that update or delete data in the same table.
- The tablespace for this table is very large and has a lot of data pages.
- The fill factor for these data pages is low.

Actions

We recommend different actions depending on the causes of your wait event.

Topics

- [Identify the sessions and queries causing the events \(p. 869\)](#)
- [Reorganize large tables during off-peak hours \(p. 870\)](#)

Identify the sessions and queries causing the events

Typically, databases with moderate to significant load have wait events. The wait events might be acceptable if performance is optimal. If performance isn't optimal, examine where the database is spending the most time. Look at the wait events that contribute to the highest load, and find out whether you can optimize the database and application to reduce those events.

To find SQL queries that are responsible for high load

1. Sign in to the AWS Management Console and open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. In the navigation pane, choose **Performance Insights**.
3. Choose a DB instance. The Performance Insights dashboard appears for that DB instance.
4. In the **Database load** chart, choose **Slice by wait**.
5. At the bottom of the page, choose **Top SQL**.

The chart lists the SQL queries that are responsible for the load. Those at the top of the list are most responsible. To resolve a bottleneck, focus on these statements.

For a useful overview of troubleshooting using Performance Insights, see the blog post [Analyze Amazon Aurora MySQL Workloads with Performance Insights](#).

Another way to find out which queries are causing high numbers of `synch/mutex/innodb/fil_system_mutex` waits is to check `performance_schema`, as in the following example.

```
mysql> select * from performance_schema.events_waits_current where EVENT_NAME='wait/synch/mutex/innodb/fil_system_mutex'\G
```

```
***** 1. row *****
    THREAD_ID: 19
        EVENT_ID: 195057
    END_EVENT_ID: 195057
        EVENT_NAME: wait/synch/mutex/innodb/fil_system_mutex
            SOURCE: filofil.cc:6700
        TIMER_START: 1010146190118400
            TIMER_END: 1010146196524000
            TIMER_WAIT: 6405600
                SPINS: NULL
        OBJECT_SCHEMA: NULL
            OBJECT_NAME: NULL
            INDEX_NAME: NULL
            OBJECT_TYPE: NULL
OBJECT_INSTANCE_BEGIN: 47285552262176
    NESTING_EVENT_ID: NULL
    NESTING_EVENT_TYPE: NULL
        OPERATION: lock
        NUMBER_OF_BYTES: NULL
        FLAGS: NULL
***** 2. row *****
    THREAD_ID: 23
        EVENT_ID: 5480
    END_EVENT_ID: 5480
        EVENT_NAME: wait/synch/mutex/innodb/fil_system_mutex
            SOURCE: filofil.cc:5906
        TIMER_START: 995269979908800
            TIMER_END: 995269980159200
            TIMER_WAIT: 250400
                SPINS: NULL
        OBJECT_SCHEMA: NULL
            OBJECT_NAME: NULL
            INDEX_NAME: NULL
            OBJECT_TYPE: NULL
OBJECT_INSTANCE_BEGIN: 47285552262176
    NESTING_EVENT_ID: NULL
    NESTING_EVENT_TYPE: NULL
        OPERATION: lock
        NUMBER_OF_BYTES: NULL
        FLAGS: NULL
***** 3. row *****
    THREAD_ID: 55
        EVENT_ID: 23233794
    END_EVENT_ID: NULL
        EVENT_NAME: wait/synch/mutex/innodb/fil_system_mutex
            SOURCE: filofil.cc:449
        TIMER_START: 1010492125341600
            TIMER_END: 1010494304900000
            TIMER_WAIT: 2179558400
                SPINS: NULL
        OBJECT_SCHEMA: NULL
            OBJECT_NAME: NULL
            INDEX_NAME: NULL
            OBJECT_TYPE: NULL
OBJECT_INSTANCE_BEGIN: 47285552262176
    NESTING_EVENT_ID: 23233786
    NESTING_EVENT_TYPE: WAIT
        OPERATION: lock
        NUMBER_OF_BYTES: NULL
        FLAGS: NULL
```

Reorganize large tables during off-peak hours

Reorganize large tables that you identify as the source of high numbers of `synch/mutex/innodb/fil_system_mutex` wait events during a maintenance window outside of production hours. Doing so

ensures that the internal tablespaces map cleanup doesn't occur when quick access to the table is critical. For information about reorganizing tables, see [OPTIMIZE TABLE Statement](#) in the *MySQL Reference*.

synch/mutex/innodb/trx_sys_mutex

The `synch/mutex/innodb/trx_sys_mutex` event occurs when there is high database activity with a large number of transactions.

Topics

- [Relevant engine versions \(p. 871\)](#)
- [Context \(p. 871\)](#)
- [Likely causes of increased waits \(p. 871\)](#)
- [Actions \(p. 872\)](#)

Relevant engine versions

This wait event information is supported for the following engine versions:

- Aurora MySQL version 2.x up to 2.09.2
- Aurora MySQL version 1.x up to 1.23.1

Context

Internally, the InnoDB database engine uses the repeatable read isolation level with snapshots to provide read consistency. This gives you a point-in-time view of the database at the time the snapshot was created.

In InnoDB, all changes are applied to the database as soon as they arrive, regardless of whether they're committed. This approach means that without multiversion concurrency control (MVCC), all users connected to the database see all of the changes and the latest rows. Therefore, InnoDB requires a way to track the changes to understand what to roll back when necessary.

To do this, InnoDB uses a transaction system (`trx_sys`) to track snapshots. The transaction system does the following:

- Tracks the transaction ID for each row in the undo logs.
- Uses an internal InnoDB structure called `ReadView` that helps to identify which transaction IDs are visible for a snapshot.

Likely causes of increased waits

Any database operation that requires the consistent and controlled handling (creating, reading, updating, and deleting) of transactions IDs generates a call from `trx_sys` to the mutex.

These calls happen inside three functions:

- `trx_sys_mutex_enter` – Creates the mutex.
- `trx_sys_mutex_exit` – Releases the mutex.
- `trx_sys_mutex_own` – Tests whether the mutex is owned.

The InnoDB Performance Schema instrumentation tracks all `trx_sys` mutex calls. Tracking includes, but isn't limited to, management of `trx_sys` on database startup or shutdown, rollback operations,

undo cleanups, row read access, and buffer pool loads. High database activity with a large number of transactions results in `synch/mutex/innodb/trx_sys_mutex` appearing among the top wait events.

For more information, see [Monitoring InnoDB Mutex Waits Using Performance Schema](#) in the MySQL documentation.

Actions

We recommend different actions depending on the causes of your wait event.

Identify the sessions and queries causing the events

Typically, databases with moderate to significant load have wait events. The wait events might be acceptable if performance is optimal. If performance isn't optimal, then examine where the database is spending the most time. Look at the wait events that contribute to the highest load. Find out whether you can optimize the database and application to reduce those events.

To view the Top SQL chart in the AWS Management Console

1. Open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. In the navigation pane, choose **Performance Insights**.
3. Choose a DB instance. The Performance Insights dashboard is shown for that DB instance.
4. In the **Database load** chart, choose **Slice by wait**.
5. Under the **Database load** chart, choose **Top SQL**.

The chart lists the SQL queries that are responsible for the load. Those at the top of the list are most responsible. To resolve a bottleneck, focus on these statements.

For a useful overview of troubleshooting using Performance Insights, see the blog post [Analyze Amazon Aurora MySQL Workloads with Performance Insights](#).

Examine other wait events

Examine the other wait events associated with the `synch/rwlock/innodb/hash_table_locks` wait event. Doing this can provide more information about the nature of the workload. A large number of transactions might reduce throughput, but the workload might also make this necessary.

For more information on how to optimize transactions, see [Optimizing InnoDB Transaction Management](#) in the MySQL documentation.

`synch/rwlock/innodb/hash_table_locks`

The `synch/rwlock/innodb/hash_table_locks` event occurs when there is contention on modifying the hash table that maps the buffer cache.

Topics

- [Supported engine versions \(p. 872\)](#)
- [Context \(p. 873\)](#)
- [Likely causes of increased waits \(p. 873\)](#)
- [Actions \(p. 873\)](#)

Supported engine versions

This wait event information is supported for Aurora MySQL version 1, up to 1.23.1.

Context

The event `synch/rwlock/innodb/hash_table_locks` indicates that there is contention on modifying the hash table that maps the buffer cache. A *hash table* is a table in memory designed to improve buffer pool access performance. This wait event is invoked when the hash table needs to be modified.

For more information, see [Buffer Pool](#) in the MySQL documentation.

Likely causes of increased waits

When the `synch/rwlock/innodb/hash_table_locks` event appears more than normal, possibly indicating a performance problem, typical causes include the following:

An undersized buffer pool

The size of the buffer pool is too small to keep all of the frequently accessed pages in memory.

Heavy workload

The workload is causing frequent evictions and data pages reloads in the buffer cache.

Actions

We recommend different actions depending on the causes of your wait event.

Topics

- [Increase the size of the buffer pool \(p. 873\)](#)
- [Improve data access patterns \(p. 873\)](#)
- [Find SQL queries responsible for high load \(p. 873\)](#)
- [Reduce or avoid full-table scans \(p. 874\)](#)

Increase the size of the buffer pool

Make sure that the buffer pool is appropriately sized for the workload. To do so, you can check the buffer pool cache hit rate. Typically, if the value drops below 95 percent, consider increasing the buffer pool size. A larger buffer pool can keep frequently accessed pages in memory longer.

To increase the size of the buffer pool, modify the value of the `innodb_buffer_pool_size` parameter. The default value of this parameter is based on the DB instance class size. For more information, see the AWS Database Blog post [Best practices for configuring parameters for Amazon RDS for MySQL, part 1: Parameters related to performance](#).

Improve data access patterns

Check the queries affected by this wait and their execution plans. Consider improving data access patterns. For example, if you are using `mysql_result::fetch_array`, you can try increasing the array fetch size.

You can use Performance Insights to show queries and sessions that might be causing the `synch/rwlock/innodb/hash_table_locks` wait event.

Find SQL queries responsible for high load

Typically, databases with moderate to significant load have wait events. The wait events might be acceptable if performance is optimal. If performance isn't optimal, then examine where the database

is spending the most time. Look at the wait events that contribute to the highest load, and find out whether you can optimize the database and application to reduce those events.

To find SQL queries that are responsible for high load

1. Sign in to the AWS Management Console and open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. In the navigation pane, choose **Performance Insights**.
3. Choose a DB instance. The Performance Insights dashboard is shown for that DB instance.
4. In the **Database load** chart, choose **Slice by wait**.
5. At the bottom of the page, choose **Top SQL**.

The chart lists the SQL queries that are responsible for the load. Those at the top of the list are most responsible. To resolve a bottleneck, focus on these statements.

For a useful overview of troubleshooting using Performance Insights, see the AWS Database Blog post [Analyze Amazon Aurora MySQL Workloads with Performance Insights](#).

Reduce or avoid full-table scans

Monitor your workload to see if it's running full-table scans, and, if it is, reduce or avoid them. For example, you can monitor status variables such as `Handler_read_rnd_next`. For more information, see [Server Status Variables](#) in the MySQL documentation.

[synch/sxlock/innodb/hash_table_locks](#)

The `synch/sxlock/innodb/hash_table_locks` event occurs when pages not found in the buffer pool must be read from a file.

Topics

- [Supported engine versions \(p. 874\)](#)
- [Context \(p. 874\)](#)
- [Likely causes of increased waits \(p. 875\)](#)
- [Actions \(p. 875\)](#)

[Supported engine versions](#)

This wait event information is supported for Aurora MySQL version 2, up to 2.09.2.

[Context](#)

The event `synch/sxlock/innodb/hash_table_locks` indicates that a workload is frequently accessing data that isn't stored in the buffer pool. This wait event is associated with new page additions and old data evictions from the buffer pool. The data stored in the buffer pool aged and new data must be cached, so the aged pages are evicted to allow caching of the new pages. MySQL uses a least recently used (LRU) algorithm to evict pages from the buffer pool. The workload is trying to access data that hasn't been loaded into the buffer pool or data that has been evicted from the buffer pool.

This wait event occurs when the workload must access the data in files on disk or when blocks are freed from or added to the buffer pool's LRU list. These operations wait to obtain a shared excluded lock (SX-lock). This SX-lock is used for the synchronization over the *hash table*, which is a table in memory designed to improve buffer pool access performance.

For more information, see [Buffer Pool](#) in the MySQL documentation.

Likely causes of increased waits

When the `synch/sxlock/innodb/hash_table_locks` wait event appears more than normal, possibly indicating a performance problem, typical causes include the following:

An undersized buffer pool

The size of the buffer pool is too small to keep all of the frequently accessed pages in memory.

Heavy workload

The workload is causing frequent evictions and data pages reloads in the buffer cache.

Errors reading the pages

There are errors reading pages in the buffer pool, which might indicate data corruption.

Actions

We recommend different actions depending on the causes of your wait event.

Topics

- [Increase the size of the buffer pool \(p. 875\)](#)
- [Improve data access patterns \(p. 875\)](#)
- [Reduce or avoid full-table scans \(p. 876\)](#)
- [Check the error logs for page corruption \(p. 876\)](#)

Increase the size of the buffer pool

Make sure that the buffer pool is appropriately sized for the workload. To do so, you can check the buffer pool cache hit rate. Typically, if the value drops below 95 percent, consider increasing the buffer pool size. A larger buffer pool can keep frequently accessed pages in memory longer. To increase the size of the buffer pool, modify the value of the `innodb_buffer_pool_size` parameter. The default value of this parameter is based on the DB instance class size. For more information, see [Best practices for Amazon Aurora MySQL database configuration](#).

Improve data access patterns

Check the queries affected by this wait and their execution plans. Consider improving data access patterns. For example, if you are using `mysqli_result::fetch_array`, you can try increasing the array fetch size.

You can use Performance Insights to show queries and sessions that might be causing the `synch/sxlock/innodb/hash_table_locks` wait event.

To find SQL queries that are responsible for high load

1. Sign in to the AWS Management Console and open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. In the navigation pane, choose **Performance Insights**.
3. Choose a DB instance. The Performance Insights dashboard is shown for that DB instance.
4. In the **Database load** chart, choose **Slice by wait**.

- At the bottom of the page, choose **Top SQL**.

The chart lists the SQL queries that are responsible for the load. Those at the top of the list are most responsible. To resolve a bottleneck, focus on these statements.

For a useful overview of troubleshooting using Performance Insights, see the AWS Database Blog post [Analyze Amazon Aurora MySQL Workloads with Performance Insights](#).

Reduce or avoid full-table scans

Monitor your workload to see if it's running full-table scans, and, if it is, reduce or avoid them. For example, you can monitor status variables such as `Handler_read_rnd_next`. For more information, see [Server Status Variables](#) in the MySQL documentation.

Check the error logs for page corruption

You can check the `mysql-error.log` for corruption-related messages that were detected near the time of the issue. Messages that you can work with to resolve the issue are in the error log. You might need to recreate objects that were reported as corrupted.

Tuning Aurora MySQL with thread states

The following table summarizes the most common general thread states for Aurora MySQL.

General thread state	Description
??? (p. 876)	This thread state indicates that a thread is processing a <code>SELECT</code> statement that requires the use of an internal temporary table to sort the data.
??? (p. 879)	This thread state indicates that a thread is reading and filtering rows for a query to determine the correct result set.

creating sort index

The `creating sort index` thread state indicates that a thread is processing a `SELECT` statement that requires the use of an internal temporary table to sort the data.

Topics

- [Supported engine versions \(p. 876\)](#)
- [Context \(p. 877\)](#)
- [Likely causes of increased waits \(p. 877\)](#)
- [Actions \(p. 877\)](#)

Supported engine versions

This thread state information is supported for the following versions:

- Aurora MySQL version 2 up to 2.09.2
- Aurora MySQL version 1 up to 1.23.1

Context

The `creating sort index` state appears when a query with an `ORDER BY` or `GROUP BY` clause can't use an existing index to perform the operation. In this case, MySQL needs to perform a more expensive `filesort` operation. This operation is typically performed in memory if the result set isn't too large. Otherwise, it involves creating a file on disk.

Likely causes of increased waits

The appearance of `creating sort index` doesn't by itself indicate a problem. If performance is poor, and you see frequent instances of `creating sort index`, the most likely cause is slow queries with `ORDER BY` or `GROUP BY` operators.

Actions

The general guideline is to find queries with `ORDER BY` or `GROUP BY` clauses that are associated with the increases in the `creating sort index` state. Then see whether adding an index or increasing the sort buffer size solves the problem.

Topics

- [Turn on the Performance Schema if it isn't turned on \(p. 877\)](#)
- [Identify the problem queries \(p. 877\)](#)
- [Examine the explain plans for filesort usage \(p. 877\)](#)
- [Increase the sort buffer size \(p. 878\)](#)

Turn on the Performance Schema if it isn't turned on

Performance Insights reports thread states only if Performance Schema instruments aren't turned on. When Performance Schema instruments are turned on, Performance Insights reports wait events instead. Performance Schema instruments provide additional insights and better tools when you investigate potential performance problems. Therefore, we recommend that you turn on the Performance Schema. For more information, see [Enabling the Performance Schema for Performance Insights on Aurora MySQL \(p. 580\)](#).

Identify the problem queries

To identify current queries that are causing increases in the `creating sort index` state, run `show processlist` and see if any of the queries have `ORDER BY` or `GROUP BY`. Optionally, run `explain` for connection `N`, where `N` is the process list ID of the query with `filesort`.

To identify past queries that are causing these increases, turn on the slow query log and find the queries with `ORDER BY`. Run `EXPLAIN` on the slow queries and look for "using filesort." For more information, see [Examine the explain plans for filesort usage \(p. 877\)](#).

Examine the explain plans for filesort usage

Identify the statements with `ORDER BY` or `GROUP BY` clauses that result in the `creating sort index` state.

The following example shows how to run `explain` on a query. The `Extra` column shows that this query uses `filesort`.

```
mysql> explain select * from mytable order by c1 limit 10\G
***** 1. row *****
```

```

      id: 1
select_type: SIMPLE
      table: mytable
    partitions: NULL
        type: ALL
possible_keys: NULL
          key: NULL
     key_len: NULL
        ref: NULL
      rows: 2064548
  filtered: 100.00
    Extra: Using filesort
1 row in set, 1 warning (0.01 sec)

```

The following example shows the result of running `EXPLAIN` on the same query after an index is created on column `c1`.

```
mysql> alter table mytable add index (c1);
```

```

mysql> explain select * from mytable order by c1 limit 10\G
***** 1. row ****
      id: 1
select_type: SIMPLE
      table: mytable
    partitions: NULL
        type: index
possible_keys: NULL
          key: c1
     key_len: 1023
        ref: NULL
      rows: 10
  filtered: 100.00
    Extra: Using index
1 row in set, 1 warning (0.01 sec)

```

For information on using indexes for sort order optimization, see [ORDER BY Optimization](#) in the MySQL documentation.

Increase the sort buffer size

To see whether a specific query required a `filesort` process that created a file on disk, check the `sort_merge_passes` variable value after running the query. The following shows an example.

```

mysql> show session status like 'sort_merge_passes';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| Sort_merge_passes | 0 |
+-----+-----+
1 row in set (0.01 sec)

--- run query
mysql> select * from mytable order by u limit 10;
--- run status again:

mysql> show session status like 'sort_merge_passes';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| Sort_merge_passes | 0 |
+-----+-----+

```

```
1 row in set (0.01 sec)
```

If the value of `sort_merge_passes` is high, consider increasing the sort buffer size. Apply the increase at the session level, because increasing it globally can significantly increase the amount of RAM MySQL uses. The following example shows how to change the sort buffer size before running a query.

```
mysql> set session sort_buffer_size=10*1024*1024;
Query OK, 0 rows affected (0.00 sec)
-- run query
```

sending data

The `sending data` thread state indicates that a thread is reading and filtering rows for a query to determine the correct result set. The name is misleading because it implies the state is transferring data, not collecting and preparing data to be sent later.

Topics

- [Supported engine versions \(p. 879\)](#)
- [Context \(p. 879\)](#)
- [Likely causes of increased waits \(p. 879\)](#)
- [Actions \(p. 880\)](#)

Supported engine versions

This thread state information is supported for the following versions:

- Aurora MySQL version 2 up to 2.09.2
- Aurora MySQL version 1 up to 1.23.1

Context

Many thread states are short-lasting. Operations occurring during `sending data` tend to perform large numbers of disk or cache reads. Therefore, `sending data` is often the longest-running state over the lifetime of a given query. This state appears when Aurora MySQL is doing the following:

- Reading and processing rows for a `SELECT` statement
- Performing a large number of reads from either disk or memory
- Completing a full read of all data from a specific query
- Reading data from a table, an index, or the work of a stored procedure
- Sorting, grouping, or ordering data

After the `sending data` state finishes preparing the data, the thread state `writing to net` indicates the return of data to the client. Typically, `writing to net` is captured only when the result set is very large or severe network latency is slowing the transfer.

Likely causes of increased waits

The appearance of `sending data` doesn't by itself indicate a problem. If performance is poor, and you see frequent instances of `sending data`, the most likely causes are as follows.

Topics

- [Inefficient query \(p. 880\)](#)
- [Suboptimal server configuration \(p. 880\)](#)

Inefficient query

In most cases, what's responsible for this state is a query that isn't using an appropriate index to find the result set of a specific query. For example, consider a query reading a 10 million record table for all orders placed in California, where the state column isn't indexed or is poorly indexed. In the latter case, the index might exist, but the optimizer ignores it because of low cardinality.

Suboptimal server configuration

If several queries appear in the sending data state, the database server might be configured poorly. Specifically, the server might have the following issues:

- The database server doesn't have enough computing capacity: disk I/O, disk type and speed, CPU, or number of CPUs.
- The server is starved for allocated resources, such as the InnoDB buffer pool for InnoDB tables or the key buffer for MyISAM tables.
- Per-thread memory settings such as `sort_buffer`, `read_buffer`, and `join_buffer` consume more RAM than required, starving the physical server for memory resources.

Actions

The general guideline is to find queries that return large numbers of rows by checking the Performance Schema. If logging queries that don't use indexes is turned on, you can also examine the results from the slow logs.

Topics

- [Turn on the Performance Schema if it isn't turned on \(p. 880\)](#)
- [Examine memory settings \(p. 880\)](#)
- [Examine the explain plans for index usage \(p. 881\)](#)
- [Check the volume of data returned \(p. 881\)](#)
- [Check for concurrency issues \(p. 881\)](#)
- [Check the structure of your queries \(p. 881\)](#)

Turn on the Performance Schema if it isn't turned on

Performance Insights reports thread states only if Performance Schema instruments aren't turned on. When Performance Schema instruments are turned on, Performance Insights reports wait events instead. Performance Schema instruments provide additional insights and better tools when you investigate potential performance problems. Therefore, we recommend that you turn on the Performance Schema. For more information, see [Enabling the Performance Schema for Performance Insights on Aurora MySQL \(p. 580\)](#).

Examine memory settings

Examine the memory settings for the primary buffer pools. Make sure that these pools are appropriately sized for the workload. If your database uses multiple buffer pool instances, make sure that they aren't divided into many small buffer pools. Threads can only use one buffer pool at a time.

Make sure that the following memory settings used for each thread are properly sized:

- `read_buffer`
- `read_rnd_buffer`
- `sort_buffer`
- `join_buffer`
- `binlog_cache`

Unless you have a specific reason to modify the settings, use the default values.

Examine the explain plans for index usage

For queries in the sending `data` thread state, examine the plan to determine whether appropriate indexes are used. If a query isn't using a useful index, consider adding hints like `USE INDEX` or `FORCE INDEX`. Hints can greatly increase or decrease the time it takes to run a query, so use care before adding them.

Check the volume of data returned

Check the tables that are being queried and the amount of data that they contain. Can any of this data be archived? In many cases, the cause of poor query execution times isn't the result of the query plan, but the volume of data to be processed. Many developers are very efficient in adding data to a database but seldom consider dataset life cycle in the design and development phases.

Look for queries that perform well in low-volume databases but perform poorly in your current system. Sometimes developers who design specific queries might not realize that these queries are returning 350,000 rows. The developers might have developed the queries in a lower-volume environment with smaller datasets than production environments have.

Check for concurrency issues

Check whether multiple queries of the same type are running at the same time. Some forms of queries run efficiently when they run alone. However, if similar forms of query run together, or in high volume, they can cause concurrency issues. Often, these issues are caused when the database uses temp tables to render results. A restrictive transaction isolation level can also cause concurrency issues.

If tables are read and written to concurrently, the database might be using locks. To help identify periods of poor performance, examine the use of databases through large-scale batch processes. To see recent locks and rollbacks, examine the output of the `SHOW ENGINE INNODB STATUS` command.

Check the structure of your queries

Check whether captured queries from these states use subqueries. This type of query often leads to poor performance because the database compiles the results internally and then substitutes them back into the query to render data. This process is an extra step for the database. In many cases, this step can cause poor performance in a highly concurrent loading condition.

Also check whether your queries use large numbers of `ORDER BY` and `GROUP BY` clauses. In such operations, often the database must first form the entire dataset in memory. Then it must order or group it in a specific manner before returning it to the client.

Working with parallel query for Amazon Aurora MySQL

Following, you can find a description of the parallel query performance optimization for Amazon Aurora MySQL-Compatible Edition. This feature uses a special processing path for certain data-intensive queries, taking advantage of the Aurora shared storage architecture. Parallel query works best with Aurora MySQL DB clusters that have tables with millions of rows and analytic queries that take minutes or hours to complete. For information about Aurora MySQL versions that support parallel query in an AWS Region, see [Aurora parallel queries \(p. 26\)](#).

Contents

- [Overview of parallel query for Aurora MySQL \(p. 883\)](#)
 - [Benefits \(p. 883\)](#)
 - [Architecture \(p. 884\)](#)
 - [Prerequisites \(p. 884\)](#)
 - [Limitations \(p. 885\)](#)
- [Planning for a parallel query cluster \(p. 885\)](#)
 - [Checking Aurora MySQL version compatibility for parallel query \(p. 886\)](#)
- [Creating a DB cluster that works with parallel query \(p. 886\)](#)
 - [Creating a parallel query cluster using the console \(p. 887\)](#)
 - [Creating a parallel query cluster using the CLI \(p. 888\)](#)
- [Turning parallel query on and off \(p. 890\)](#)
 - [Aurora MySQL 1.23 and 2.09 or higher \(p. 890\)](#)
 - [Before Aurora MySQL 1.23 \(p. 891\)](#)
 - [Turning on hash join for parallel query clusters \(p. 892\)](#)
 - [Turning on and turning off parallel query using the console \(p. 892\)](#)
 - [Turning on and turning off parallel query using the CLI \(p. 893\)](#)
- [Upgrade considerations for parallel query \(p. 893\)](#)
 - [Upgrading parallel query clusters to Aurora MySQL version 3 \(p. 893\)](#)
 - [Upgrading to Aurora MySQL 1.23 or 2.09 and higher \(p. 894\)](#)
- [Performance tuning for parallel query \(p. 894\)](#)
- [Creating schema objects to take advantage of parallel query \(p. 895\)](#)
- [Verifying which statements use parallel query \(p. 895\)](#)
- [Monitoring parallel query \(p. 898\)](#)
- [How parallel query works with SQL constructs \(p. 901\)](#)
 - [EXPLAIN statement \(p. 902\)](#)
 - [WHERE clause \(p. 903\)](#)
 - [Data definition language \(DDL\) \(p. 905\)](#)
 - [Column data types \(p. 905\)](#)
 - [Partitioned tables \(p. 905\)](#)
 - [Aggregate functions, GROUP BY clauses, and HAVING clauses \(p. 906\)](#)
 - [Function calls in WHERE clause \(p. 907\)](#)
 - [LIMIT clause \(p. 908\)](#)
 - [Comparison operators \(p. 908\)](#)
 - [Joins \(p. 908\)](#)
 - [Subqueries \(p. 909\)](#)
 - [UNION \(p. 910\)](#)
 - [Views \(p. 910\)](#)
- [Data manipulation language \(DML\) statements \(p. 910\)](#)

- [Transactions and locking \(p. 911\)](#)
- [B-tree indexes \(p. 913\)](#)
- [Full-text search \(FTS\) indexes \(p. 913\)](#)
- [Virtual columns \(p. 913\)](#)
- [Built-in caching mechanisms \(p. 913\)](#)
- [MyISAM temporary tables \(p. 914\)](#)

Overview of parallel query for Aurora MySQL

Aurora MySQL parallel query is an optimization that parallelizes some of the I/O and computation involved in processing data-intensive queries. The work that is parallelized includes retrieving rows from storage, extracting column values, and determining which rows match the conditions in the `WHERE` clause and join clauses. This data-intensive work is delegated (in database optimization terms, *pushed down*) to multiple nodes in the Aurora distributed storage layer. Without parallel query, each query brings all the scanned data to a single node within the Aurora MySQL cluster (the head node) and performs all the query processing there.

Tip

The PostgreSQL database engine also has a feature that's also called "parallel query". That feature is unrelated to Aurora parallel query.

When the parallel query feature is turned on, the Aurora MySQL engine automatically determines when queries can benefit, without requiring SQL changes such as hints or table attributes. In the following sections, you can find an explanation of when parallel query is applied to a query. You can also find how to make sure that parallel query is applied where it provides the most benefit.

Note

The parallel query optimization provides the most benefit for long-running queries that take minutes or hours to complete. Aurora MySQL generally doesn't perform parallel query optimization for inexpensive queries. It also generally doesn't perform parallel query optimization if another optimization technique makes more sense, such as query caching, buffer pool caching, or index lookups. If you find that parallel query isn't being used when you expect it, see [Verifying which statements use parallel query \(p. 895\)](#).

Topics

- [Benefits \(p. 883\)](#)
- [Architecture \(p. 884\)](#)
- [Prerequisites \(p. 884\)](#)
- [Limitations \(p. 885\)](#)

Benefits

With parallel query, you can run data-intensive analytic queries on Aurora MySQL tables. In many cases, you can get an order-of-magnitude performance improvement over the traditional division of labor for query processing.

Benefits of parallel query include the following:

- Improved I/O performance, due to parallelizing physical read requests across multiple storage nodes.
- Reduced network traffic. Aurora doesn't transmit entire data pages from storage nodes to the head node and then filter out unnecessary rows and columns afterward. Instead, Aurora transmits compact tuples containing only the column values needed for the result set.

- Reduced CPU usage on the head node, due to pushing down function processing, row filtering, and column projection for the `WHERE` clause.
- Reduced memory pressure on the buffer pool. The pages processed by the parallel query aren't added to the buffer pool. This approach reduces the chance of a data-intensive scan evicting frequently used data from the buffer pool.
- Potentially reduced data duplication in your extract, transform, load (ETL) pipeline, by making it practical to perform long-running analytic queries on existing data.

Architecture

The parallel query feature uses the major architectural principles of Aurora MySQL: decoupling the database engine from the storage subsystem, and reducing network traffic by streamlining communication protocols. Aurora MySQL uses these techniques to speed up write-intensive operations such as redo log processing. Parallel query applies the same principles to read operations.

Note

The architecture of Aurora MySQL parallel query differs from that of similarly named features in other database systems. Aurora MySQL parallel query doesn't involve symmetric multiprocessing (SMP) and so doesn't depend on the CPU capacity of the database server. The parallel processing happens in the storage layer, independent of the Aurora MySQL server that serves as the query coordinator.

By default, without parallel query, the processing for an Aurora query involves transmitting raw data to a single node within the Aurora cluster (the *head node*). Aurora then performs all further processing for that query in a single thread on that single node. With parallel query, much of this I/O-intensive and CPU-intensive work is delegated to nodes in the storage layer. Only the compact rows of the result set are transmitted back to the head node, with rows already filtered, and column values already extracted and transformed. The performance benefit comes from the reduction in network traffic, reduction in CPU usage on the head node, and parallelizing the I/O across the storage nodes. The amount of parallel I/O, filtering, and projection is independent of the number of DB instances in the Aurora cluster that runs the query.

Prerequisites

To use all features of parallel query requires an Aurora MySQL DB cluster that's running version 1.23 or 2.09 and higher. If you already have a cluster that you want to use with parallel query, you can upgrade it to a compatible version and turn on parallel query afterward. In that case, make sure to follow the upgrade procedure in [Upgrade considerations for parallel query \(p. 893\)](#) because the configuration setting names and default values are different in these newer versions.

You can also use parallel query with certain older Aurora MySQL versions that are compatible with MySQL 5.6: 1.22.2, 1.20.1, 1.19.6, and 5.6.10a. The parallel query support for these older versions is only available in certain AWS Regions. Those older versions have additional limitations, as described following. Using parallel query with an older Aurora MySQL version also requires creating a dedicated DB cluster with a special engine mode parameter that can't be changed later. For those reasons, we recommend using parallel query with Aurora MySQL 1.23 or 2.09 and higher where practical.

The DB instances in your cluster must be using the `db.r*` instance classes.

Make sure that the hash join optimization is turned on for your cluster. The procedure to do so is different depending on whether your cluster is running an Aurora MySQL version higher or lower than 1.23 or 2.09. To learn how, see [Turning on hash join for parallel query clusters \(p. 892\)](#).

To customize parameters such as `aurora_parallel_query` and `aurora_disable_hash_join`, you must have a custom parameter group that you use with your cluster. You can specify these parameters individually for each DB instance by using a DB parameter group. However, we recommend that you

specify them in a DB cluster parameter group. That way, all DB instances in your cluster inherit the same settings for these parameters.

Limitations

The following limitations apply to the parallel query feature:

- You can't use parallel query with the db.t2 or db.t3 instance classes. This limitation applies even if you request parallel query using the `aurora_pq_force` SQL hint.
- Parallel query doesn't apply to tables using the COMPRESSED or REDUNDANT row formats. Use the COMPACT or DYNAMIC row formats for tables you plan to use with parallel query.
- Aurora uses a cost-based algorithm to determine whether to use the parallel query mechanism for each SQL statement. Using certain SQL constructs in a statement can prevent parallel query or make parallel query unlikely for that statement. For information about compatibility of SQL constructs with parallel query, see [How parallel query works with SQL constructs \(p. 901\)](#).
- Each Aurora DB instance can run only a certain number of parallel query sessions at one time. If a query has multiple parts that use parallel query, such as subqueries, joins, or UNION operators, those phases run in sequence. The statement only counts as a single parallel query session at any one time. You can monitor the number of active sessions using the [parallel query status variables \(p. 898\)](#). You can check the limit on concurrent sessions for a given DB instance by querying the status variable `Aurora_pq_max_concurrent_requests`.
- Parallel query is available in all AWS Regions that Aurora supports. For most AWS Regions, the minimum required Aurora MySQL version to use parallel query is 1.23 or 2.09. For more information, see [Aurora parallel queries \(p. 26\)](#).
- Aurora MySQL 1.22.2, 1.20.1, 1.19.6, and 5.6.10a only: Using parallel query with these older versions involves creating a new cluster, or restoring from an existing Aurora MySQL cluster snapshot.
- Aurora MySQL 1.22.2, 1.20.1, 1.19.6, and 5.6.10a only: Parallel query doesn't support AWS Identity and Access Management (IAM) database authentication.

Planning for a parallel query cluster

Planning for a DB cluster that has parallel query turned on requires making some choices. These include performing setup steps (either creating or restoring a full Aurora MySQL cluster) and deciding how broadly to turn on parallel query across your DB cluster.

Consider the following as part of planning:

- Which Aurora MySQL version do you plan to use for the cluster? Depending on your choice, you can use one of these ways to turn on parallel query for the cluster:

If you use Aurora MySQL that's compatible with MySQL 5.7, you must choose Aurora MySQL 2.09 or higher. In this case, you always create a provisioned cluster. Then you turn on parallel query using the `aurora_parallel_query` parameter. We recommend this choice if you are starting with Aurora parallel query for the first time.

If you use Aurora MySQL that's compatible with MySQL 5.6, you can choose version 1.23 or certain lower versions. With version 1.23 or higher, you create a provisioned cluster and then turn on parallel query using the `aurora_parallel_query` DB cluster parameter. With a version lower than 1.23, you choose the `parallelquery` engine mode when creating the cluster. In this case, parallel query is permanently turned on for the cluster. The `parallelquery` engine mode imposes limitations on interoperating with other kinds of Aurora MySQL clusters. If you have a choice, we recommend choosing version 1.23 or higher for Aurora MySQL with MySQL 5.6 compatibility.

If you have an existing Aurora MySQL cluster that's running version 1.23 or higher, or 2.09 or higher, you don't have to create a new cluster to use parallel query. You can associate your cluster, or

specific DB instances in the cluster, with a parameter group that has the `aurora_parallel_query` parameter turned on. By doing so, you can reduce the time and effort to set up the relevant data to use with parallel query.

- Plan for any large tables that you need to reorganize so that you can use parallel query when accessing them. You might need to create new versions of some large tables where parallel query is useful. For example, you might need to remove full-text search indexes. For details, see [Creating schema objects to take advantage of parallel query \(p. 895\)](#).

Checking Aurora MySQL version compatibility for parallel query

To check which Aurora MySQL versions are compatible with parallel query clusters, use the `describe-db-engine-versions` AWS CLI command and check the value of the `SupportsParallelQuery` field. The following code example shows how to check which combinations are available for parallel query clusters in a specified AWS Region. Make sure to specify the full `--query` parameter string on a single line.

```
aws rds describe-db-engine-versions --region us-east-1 --engine aurora --query '*[]|[? SupportsParallelQuery == `true`].[EngineVersion]' --output text

aws rds describe-db-engine-versions --region us-east-1 --engine aurora-mysql --query '*[]|[?SupportsParallelQuery == `true`].[EngineVersion]' --output text
```

The preceding commands produce output similar to the following. The output might vary depending on which Aurora MySQL versions are available in the specified AWS Region.

```
5.6.10a
5.6.mysql_aurora.1.19.0
5.6.mysql_aurora.1.19.1
5.6.mysql_aurora.1.19.2
5.6.mysql_aurora.1.19.3
5.6.mysql_aurora.1.19.3.1
5.6.mysql_aurora.1.19.3.90
5.6.mysql_aurora.1.19.4
5.6.mysql_aurora.1.19.4.1
5.6.mysql_aurora.1.19.4.2
5.6.mysql_aurora.1.19.4.3
5.6.mysql_aurora.1.19.4.4
5.6.mysql_aurora.1.19.4.5
5.6.mysql_aurora.1.19.5
5.6.mysql_aurora.1.19.5.90
5.6.mysql_aurora.1.19.6
5.6.mysql_aurora.1.20.1
5.6.mysql_aurora.1.22.0
5.6.mysql_aurora.1.22.2
5.6.mysql_aurora.1.23.0

5.7.mysql_aurora.2.09.0
```

After you start using parallel query with a cluster, you can monitor performance and remove obstacles to parallel query usage. For those instructions, see [Performance tuning for parallel query \(p. 894\)](#).

Creating a DB cluster that works with parallel query

To create an Aurora MySQL cluster with parallel query, add new instances to it, or perform other administrative operations, you use the same AWS Management Console and AWS CLI techniques that you do with other Aurora MySQL clusters. You can create a new cluster to work with parallel query. You can also create a DB cluster to work with parallel query by restoring from a snapshot of a MySQL-

compatible Aurora DB cluster. If you aren't familiar with the process for creating a new Aurora MySQL cluster, you can find background information and prerequisites in [Creating an Amazon Aurora DB cluster \(p. 125\)](#).

However, certain options are different:

- When you choose an Aurora MySQL engine version, we recommend that you choose the latest engine that is compatible with MySQL 5.7. Currently, Aurora MySQL 2.09 or higher, and certain Aurora MySQL versions that are compatible with MySQL 5.6 support parallel query. You have more flexibility to turn parallel query on and off, or use parallel query with existing clusters, if you use Aurora MySQL 1.23 or 2.09 and higher.
- Only for Aurora MySQL before version 1.23: When you create or restore the DB cluster, make sure to choose the **parallelquery** engine mode.

Whether you create a new cluster or restore from a snapshot, you use the same techniques to add new DB instances that you do with other Aurora MySQL clusters.

Creating a parallel query cluster using the console

You can create a new parallel query cluster with the console as described following.

To create a parallel query cluster with the AWS Management Console

1. Follow the general AWS Management Console procedure in [Creating an Amazon Aurora DB cluster \(p. 125\)](#).
2. On the **Select engine** screen, choose Aurora MySQL.

For **Engine version**, choose Aurora MySQL 2.09 or higher, or Aurora MySQL 1.23 or higher if practical. With those versions, you have the fewest limitations on parallel query usage. Those versions also have the most flexibility to turn parallel query on or off at any time.

If it isn't practical to use a recent Aurora MySQL version for this cluster, choose **Show versions that support the parallel query feature**. Doing so filters the **Version** menu to show only the specific Aurora MySQL versions that are compatible with parallel query.

3. (For older versions only) For **Capacity type**, choose **Provisioned with Aurora parallel query enabled**. The AWS Management Console only displays this choice when you select an Aurora MySQL version lower than 1.23. For Aurora MySQL 1.23 or 2.09 and higher, you don't need to make any special choice to make the cluster compatible with parallel query.
4. (For recent versions only) For **Additional configuration**, choose a parameter group that you created for **DB cluster parameter group**. Using such a custom parameter group is required for Aurora MySQL 1.23 or 2.09 or 3.1 and higher. In your DB cluster parameter group, specify the parameter settings `aurora_parallel_query=ON` and `aurora_disable_hash_join=OFF`. Doing so turns on parallel query for the cluster, and turns on the hash join optimization which works in combination with parallel query.

To verify that a new cluster can use parallel query

1. Create a cluster using the preceding technique.
2. (For Aurora MySQL version 2.09 and higher minor versions, or Aurora MySQL version 3) Check that the `aurora_parallel_query` configuration setting is true.

```
mysql> select @@aurora_parallel_query;
+-----+
| @@aurora_parallel_query |
+-----+
```

```
|           1 |
+-----+
```

3. (For Aurora MySQL version 2.09 and higher minor versions) Check that the `aurora_disable_hash_join` setting is false.

```
mysql> select @@aurora_disable_hash_join;
+-----+
| @@aurora_disable_hash_join |
+-----+
|          0 |
+-----+
```

4. (For older versions only) Check that the `aurora_pq_supported` configuration setting is true.

```
mysql> select @@aurora_pq_supported;
+-----+
| @@aurora_pq_supported |
+-----+
|          1 |
+-----+
```

5. With some large tables and data-intensive queries, check the query plans to confirm that some of your queries are using the parallel query optimization. To do so, follow the procedure in [Verifying which statements use parallel query \(p. 895\)](#).

Creating a parallel query cluster using the CLI

You can create a new parallel query cluster with the CLI as described following.

To create a parallel query cluster with the AWS CLI

1. (Optional) Check which Aurora MySQL versions are compatible with parallel query clusters. To do so, use the `describe-db-engine-versions` command and check the value of the `SupportsParallelQuery` field. For an example, see [Checking Aurora MySQL version compatibility for parallel query \(p. 886\)](#).
2. (Optional) Create a custom DB cluster parameter group with the settings `aurora_parallel_query=ON` and `aurora_disable_hash_join=OFF`. Use commands such as the following.

```
aws rds create-db-cluster-parameter-group --db-parameter-group-family aurora-mysql5.7
--db-cluster-parameter-group-name pq-enabled-57-compatible
aws rds modify-db-cluster-parameter-group --db-cluster-parameter-group-name pq-
enabled-57-compatible \
    --parameters
    ParameterName=aurora_parallel_query,ParameterValue=ON,ApplyMethod=pending-reboot
aws rds modify-db-cluster-parameter-group --db-cluster-parameter-group-name pq-
enabled-57-compatible \
    --parameters
    ParameterName=aurora_disable_hash_join,ParameterValue=OFF,ApplyMethod=pending-reboot
```

If you perform this step, specify the option `--db-cluster-parameter-group-name my_cluster_parameter_group` in the subsequent `create-db-cluster` statement. Substitute the name of your own parameter group. If you omit this step, you create the parameter group and associate it with the cluster later, as described in [Turning parallel query on and off \(p. 890\)](#).

3. Follow the general AWS CLI procedure in [Creating an Amazon Aurora DB cluster \(p. 125\)](#).
4. Specify the following set of options:

- For the `--engine` option, use `aurora` or `aurora-mysql`. These values produce parallel query clusters that are compatible with MySQL 5.6 or MySQL 5.7 respectively.
- The value to use for the `--engine-mode` parameter depends on the engine version that you choose.

For Aurora MySQL 1.23 or higher, or 2.09 or higher, specify `--engine-mode provisioned`. You can also omit the `--engine-mode` parameter, because `provisioned` is the default. In these versions, you can turn parallel query on or off for the default kind of Aurora MySQL clusters, instead of creating clusters dedicated to always using parallel query.

Before Aurora MySQL 1.23, for the `--engine-mode` option, use `parallelquery`. The `--engine-mode` parameter applies to the `create-db-cluster` operation. Then the engine mode of the cluster is used automatically by subsequent `create-db-instance` operations.

- For the `--db-cluster-parameter-group-name` option, specify the name of a DB cluster parameter group that you created and specified the parameter value `aurora_parallel_query=ON`. If you omit this option, you can create the cluster with a default parameter group and later modify it to use such a custom parameter group.
- For the `--engine-version` option, use an Aurora MySQL version that's compatible with parallel query. Use the procedure from [Planning for a parallel query cluster \(p. 885\)](#) to get a list of versions if necessary. If practical, use at least 1.23.0 or 2.09.0. These versions and all higher ones contain substantial enhancements to parallel query.

The following code example shows how. Substitute your own value for each of the environment variables such as `$CLUSTER_ID`.

```
aws rds create-db-cluster --db-cluster-identifier $CLUSTER_ID --engine aurora-mysql --engine-version 5.7.mysql_aurora.2.09.0 \
--master-username $MASTER_USER_ID --master-user-password $MASTER_USER_PW \
--db-cluster-parameter-group-name $CUSTOM_CLUSTER_PARAM_GROUP

aws rds create-db-cluster --db-cluster-identifier $CLUSTER_ID
--engine aurora --engine-version 5.6.mysql_aurora.1.23.0 \
--master-username $MASTER_USER_ID --master-user-password $MASTER_USER_PW \
--db-cluster-parameter-group-name $CUSTOM_CLUSTER_PARAM_GROUP

aws rds create-db-instance --db-instance-identifier ${INSTANCE_ID}-1 \
--engine same_value_as_in_create_cluster_command \
--db-cluster-identifier $CLUSTER_ID --db-instance-class $INSTANCE_CLASS
```

5. Verify that a cluster you created or restored has the parallel query feature available.

For Aurora MySQL 1.23 and 2.09 or higher: Check that the `aurora_parallel_query` configuration setting exists. If this setting has the value 1, parallel query is ready for you to use. If this setting has the value 0, set it to 1 before you can use parallel query. Either way, the cluster is capable of performing parallel queries.

```
mysql> select @@aurora_parallel_query;
+-----+
| @@aurora_parallel_query |
+-----+
| 1 |
+-----+
```

Before Aurora MySQL 1.23: Check that the `aurora_pq_supported` configuration setting is true.

```
mysql> select @@aurora_pq_supported;
+-----+
| @@aurora_pq_supported |
+-----+
```

```
+-----+  
| 1 |  
+-----+
```

To restore a snapshot to a parallel query cluster with the AWS CLI

1. Check which Aurora MySQL versions are compatible with parallel query clusters. To do so, use the `describe-db-engine-versions` command and check the value of the `SupportsParallelQuery` field. For an example, see [Checking Aurora MySQL version compatibility for parallel query \(p. 886\)](#). Decide which version to use for the restored cluster. If practical, choose Aurora MySQL 2.09.0 or higher for a MySQL 5.7-compatible cluster, or 1.23.0 or higher for a MySQL 5.6-compatible cluster.
2. Locate an Aurora MySQL-compatible cluster snapshot.
3. Follow the general AWS CLI procedure in [Restoring from a DB cluster snapshot \(p. 497\)](#).
4. The value to use for the `--engine-mode` parameter depends on the engine version that you choose.

For Aurora MySQL 1.23 or higher, or 2.09 or higher, specify `--engine-mode provisioned`. You can also omit the `--engine-mode` parameter, because `provisioned` is the default. In these versions, you can turn parallel query on or off for your Aurora MySQL clusters, instead of creating clusters dedicated to always using parallel query.

Before Aurora MySQL 1.23, specify `--engine-mode parallelquery`. The `--engine-mode` parameter applies to the `create-db-cluster` operation. Then the engine mode of the cluster is used automatically by subsequent `create-db-instance` operations.

```
aws rds restore-db-cluster-from-snapshot \  
  --db-cluster-identifier mynewdbcluster \  
  --snapshot-identifier mydbclustersnapshot \  
  --engine aurora \  
  --engine-mode parallelquery
```

5. Verify that a cluster you created or restored has the parallel query feature available. Use the same verification procedure as in [Creating a parallel query cluster using the CLI \(p. 888\)](#).

Turning parallel query on and off

Note

When parallel query is turned on, Aurora MySQL determines whether to use it at runtime for each query. In the case of joins, unions, subqueries, and so on, Aurora MySQL determines whether to use parallel query at runtime for each query block. For details, see [Verifying which statements use parallel query \(p. 895\)](#) and [How parallel query works with SQL constructs \(p. 901\)](#).

Aurora MySQL 1.23 and 2.09 or higher

In Aurora MySQL 1.23 and 2.09 or higher, you can turn on and turn off parallel query dynamically at both the global and session level for a DB instance by using the `aurora_parallel_query` option. You can change the `aurora_parallel_query` setting in your DB cluster group to turn parallel query on or off by default.

```
mysql> select @@aurora_parallel_query;  
+-----+  
| @@aurora_parallel_query|  
+-----+
```

```
|           1 |
+-----+
```

To toggle the `aurora_parallel_query` parameter at the session level, use the standard methods to change a client configuration setting. For example, you can do so through the `mysql` command line or within a JDBC or ODBC application. The command on the standard MySQL client is `set session aurora_parallel_query = { 'ON' / 'OFF' }`. You can also add the session-level parameter to the JDBC configuration or within your application code to turn on or turn off parallel query dynamically.

You can permanently change the setting for the `aurora_parallel_query` parameter, either for a specific DB instance or for your whole cluster. If you specify the parameter value in a DB parameter group, that value only applies to specific DB instance in your cluster. If you specify the parameter value in a DB cluster parameter group, all DB instances in the cluster inherit the same setting. To toggle the `aurora_parallel_query` parameter, use the techniques for working with parameter groups, as described in [Working with DB parameter groups and DB cluster parameter groups \(p. 339\)](#). Follow these steps:

1. Create a custom cluster parameter group (recommended) or a custom DB parameter group.
2. In this parameter group, update `parallel_query` to the value that you want.
3. Depending on whether you created a DB cluster parameter group or a DB parameter group, attach the parameter group to your Aurora cluster or to the specific DB instances where you plan to use the parallel query feature.

Tip

Because `aurora_parallel_query` is a dynamic parameter, you don't need to restart your cluster after changing this setting.

You can modify the parallel query parameter by using the [ModifyDBClusterParameterGroup](#) or [ModifyDBParameterGroup](#) API operation or the AWS Management Console.

Before Aurora MySQL 1.23

For these older versions, you can turn on and turn off parallel query dynamically at both the global and session level for a DB instance by using the `aurora_pq` option. On clusters where the parallel query feature is available, the parameter is turned on by default.

```
mysql> select @@aurora_pq;
+-----+
| @@aurora_pq |
+-----+
|          1 |
+-----+
```

To toggle the `aurora_pq` parameter at the session level, for example through the `mysql` command line or within a JDBC or ODBC application, use the standard methods to change a client configuration setting. For example, the command on the standard MySQL client is `set session aurora_pq = { 'ON' / 'OFF' }`. You can also add the session-level parameter to the JDBC configuration or within your application code to turn on or turn off parallel query dynamically.

To toggle the `aurora_pq` parameter permanently, use the techniques for working with parameter groups, as described in [Working with DB parameter groups and DB cluster parameter groups \(p. 339\)](#). Follow these steps:

1. Create a custom cluster parameter group or DB instance parameter group. We recommend using a cluster parameter group, so that all DB instance in the cluster inherit the same settings.
2. In this parameter group, update `aurora_pq` to the value that you want.

3. Associate the custom cluster parameter group with the Aurora cluster where you plan to use the parallel query feature. Or, for a custom DB parameter group, associate it with one or more DB instances in the cluster.
4. Restart all the DB instances of the cluster.

You can modify the parallel query parameter by using the [ModifyDBClusterParameterGroup](#) or [ModifyDBParameterGroup](#) API operation or the AWS Management Console.

Note

When parallel query is turned on, Aurora MySQL determines whether to use it at runtime for each query. In the case of joins, unions, subqueries, and so on, Aurora MySQL determines whether to use parallel query at runtime for each query block. For details, see [Verifying which statements use parallel query \(p. 895\)](#) and [How parallel query works with SQL constructs \(p. 901\)](#).

Turning on hash join for parallel query clusters

Parallel query is typically used for the kinds of resource-intensive queries that benefit from the hash join optimization. Thus, it's helpful to ensure that hash joins are turned on for clusters where you plan to use parallel query.

- In Aurora MySQL version 3, the hash join optimization is turned on by default. You can turn it on and off by using the `block_nested_loop` flag of the `optimizer_switch` configuration setting. The `aurora_disable_hash_join` option isn't used.
- In Aurora MySQL 1.23 or 2.09 and higher minor versions, the parallel query and hash join settings are both turned off by default. When you turn on parallel query for such a cluster, turn on hash joins also. The simplest way to do so is to set the cluster configuration parameter `aurora_disable_hash_join=OFF`.
- For Aurora MySQL 5.6-compatible clusters before version 1.23, hash joins are always available in parallel query clusters. In this case, you don't need to take any action for the hash join feature. If you upgrade such clusters to a higher release of version 1 or version 2, you do need to turn on hash joins at that time.

For information about how to use hash joins effectively, see [Optimizing large Aurora MySQL join queries with hash joins \(p. 1038\)](#).

Turning on and turning off parallel query using the console

You can turn on or turn off parallel query at the DB instance level or the DB cluster level by working with parameter groups.

To turn on or turn off parallel query for a DB cluster with the AWS Management Console

1. Create a custom parameter group, as described in [Working with DB parameter groups and DB cluster parameter groups \(p. 339\)](#).
2. For Aurora MySQL 1.23 and 2.09 or higher: Update `aurora_parallel_query` to `1` (turned on) or `0` (turned off). For clusters where the parallel query feature is available, `aurora_parallel_query` is turned off by default.

For Aurora MySQL before 1.23: Update `aurora_pq` to `1` (turned on) or `0` (turned off). For clusters where the parallel query feature is available, `aurora_pq` is turned on by default.

3. If you use a custom cluster parameter group, attach it to the Aurora DB cluster where you plan to use the parallel query feature. If you use a custom DB parameter group, attach it to one or more DB instances in the cluster. We recommend using a cluster parameter group. Doing so makes sure that all DB instances in the cluster have the same settings for parallel query and associated features such as hash join.

Turning on and turning off parallel query using the CLI

You can modify the parallel query parameter by using the `modify-db-cluster-parameter-group` or `modify-db-parameter-group` command. Choose the appropriate command depending on whether you specify the value of `aurora_parallel_query` through a DB cluster parameter group or a DB parameter group.

To turn on or turn off parallel query for a DB cluster with the CLI

- Modify the parallel query parameter by using the `modify-db-cluster-parameter-group` command. Use a command such as the following. Substitute the appropriate name for your own custom parameter group. Substitute either ON or OFF for the `ParameterValue` portion of the `--parameters` option.

```
# Aurora MySQL 1.23 or 2.09 and higher:  
$ aws rds modify-db-cluster-parameter-group --db-cluster-parameter-group-  
name cluster_param_group_name \  
  --parameters  
    ParameterName=aurora_parallel_query,ParameterValue=ON,ApplyMethod=pending-reboot  
{  
  "DBClusterParameterGroupName": "cluster_param_group_name"  
}  
  
# Before Aurora MySQL 1.23:  
$ aws rds modify-db-cluster-parameter-group --db-cluster-parameter-group-  
name cluster_param_group_name \  
  --parameters ParameterName=aurora_pq,ParameterValue=ON,ApplyMethod=pending-reboot  
{  
  "DBClusterParameterGroupName": "cluster_param_group_name"  
}
```

You can also turn on or turn off parallel query at the session level, for example through the `mysql` command line or within a JDBC or ODBC application. To do so, use the standard methods to change a client configuration setting. For example, the command on the standard MySQL client is `set session aurora_parallel_query = { 'ON' / 'OFF' }` for Aurora MySQL 1.23 or 2.09 and higher. Before Aurora MySQL 1.23, the command is `set session aurora_pq = { 'ON' / 'OFF' }`.

You can also add the session-level parameter to the JDBC configuration or within your application code to turn on or turn off parallel query dynamically.

Upgrade considerations for parallel query

Depending on the original and destination versions when you upgrade a parallel query cluster, you might find enhancements in the types of queries that parallel query can optimize. You might also find that you don't need to specify a special engine mode parameter for parallel query. The following sections explain the considerations when you upgrade a cluster that has parallel query turned on.

Upgrading parallel query clusters to Aurora MySQL version 3

Several SQL statements, clauses, and data types have new or improved parallel query support starting in Aurora MySQL version 3. When you upgrade from a release that's earlier than version 3, check whether additional queries can benefit from parallel query optimizations. For information about these parallel query enhancements, see [Column data types \(p. 905\)](#), [Partitioned tables \(p. 905\)](#), and [Aggregate functions, GROUP BY clauses, and HAVING clauses \(p. 906\)](#).

If you are upgrading a parallel query cluster from Aurora MySQL 2.08 or lower, also learn about changes in how to turn on parallel query. To do so, read [Upgrading to Aurora MySQL 1.23 or 2.09 and higher \(p. 894\)](#).

In Aurora MySQL version 3, the hash join optimization is turned on by default. The `aurora_disable_hash_join` configuration option from earlier versions isn't used.

Upgrading to Aurora MySQL 1.23 or 2.09 and higher

In Aurora MySQL 1.23 or 2.09 and higher, parallel query works for provisioned clusters and doesn't require the `parallelquery` engine mode parameter. Thus, you don't need to create a new cluster or restore from an existing snapshot to use parallel query with these versions. You can use the upgrade procedures described in [Upgrading the minor version or patch level of an Aurora MySQL DB cluster \(p. 1088\)](#) to upgrade your cluster to such a version. You can upgrade an older cluster regardless of whether it was a parallel query cluster or a provisioned cluster. To reduce the number of choices in the **Engine version** menu, you can choose **Show versions that support the parallel query feature** to filter the entries in that menu. Then choose Aurora MySQL 1.23 or 2.09 and higher.

After you upgrade an earlier parallel query cluster to Aurora MySQL 1.23 or 2.09 and higher, you turn on parallel query in the upgraded cluster. Parallel query is turned off by default in these versions, and the procedure for enabling it is different. The hash join optimization is also turned off by default and must be turned on separately. Thus, make sure that you turn on these settings again after the upgrade. For instructions on doing so, see [Turning parallel query on and off \(p. 890\)](#) and [Turning on hash join for parallel query clusters \(p. 892\)](#).

In particular, you turn on parallel query by using the configuration parameters `aurora_parallel_query=ON` and `aurora_disable_hash_join=OFF` instead of `aurora_pq_supported` and `aurora_pq`. The `aurora_pq_supported` and `aurora_pq` parameters are deprecated in the newer Aurora MySQL versions.

In the upgraded cluster, the `EngineMode` attribute has the value `provisioned` instead of `parallelquery`. To check whether parallel query is available for a specified engine version, now you check the value of the `SupportsParallelQuery` field in the output of the `describe-db-engine-versions` AWS CLI command. In earlier Aurora MySQL versions, you checked for the presence of `parallelquery` in the `SupportedEngineModes` list.

After you upgrade to Aurora MySQL 1.23 or 2.09 and higher, you can take advantage of the following features. These features aren't available to parallel query clusters running older Aurora MySQL versions.

- Performance Insights. For more information, see [Monitoring DB load with Performance Insights on Amazon Aurora \(p. 573\)](#).
- Backtracking. For more information, see [Backtracking an Aurora DB cluster \(p. 816\)](#).
- Stopping and starting the cluster. For more information, see [Stopping and starting an Amazon Aurora DB cluster \(p. 368\)](#).

Performance tuning for parallel query

To manage the performance of a workload with parallel query, make sure that parallel query is used for the queries where this optimization helps the most.

To do so, you can do the following:

- Make sure that your biggest tables are compatible with parallel query. You might change table properties or recreate some tables so that queries for those tables can take advantage of the parallel query optimization. To learn how, see [Creating schema objects to take advantage of parallel query \(p. 895\)](#).
- Monitor which queries use parallel query. To learn how, see [Monitoring parallel query \(p. 898\)](#).
- Verify that parallel query is being used for the most data-intensive and long-running queries, and with the right level of concurrency for your workload. To learn how, see [Verifying which statements use parallel query \(p. 895\)](#).

- Fine-tune your SQL code to turn on parallel query to apply to the queries that you expect. To learn how, see [How parallel query works with SQL constructs \(p. 901\)](#).

Creating schema objects to take advantage of parallel query

Before you create or modify tables that you plan to use for parallel query, make sure to familiarize yourself with the requirements described in [Prerequisites \(p. 884\)](#) and [Limitations \(p. 885\)](#).

Because parallel query requires tables to use the `ROW_FORMAT=Compact` or `ROW_FORMAT=Dynamic` setting, check your Aurora configuration settings for any changes to the `INNODB_FILE_FORMAT` configuration option. Issue the `SHOW TABLE STATUS` statement to confirm the row format for all the tables in a database.

Before changing your schema to turn on parallel query to work with more tables, make sure to test. Your tests should confirm if parallel query results in a net increase in performance for those tables. Also, make sure that the schema requirements for parallel query are otherwise compatible with your goals.

For example, before switching from `ROW_FORMAT=Compressed` to `ROW_FORMAT=Compact` or `ROW_FORMAT=Dynamic`, test the performance of workloads for the original and new tables. Also, consider other potential effects such as increased data volume.

Verifying which statements use parallel query

In typical operation, you don't need to perform any special actions to take advantage of parallel query. After a query meets the essential requirements for parallel query, the query optimizer automatically decides whether to use parallel query for each specific query.

If you run experiments in a development or test environment, you might find that parallel query isn't used because your tables are too small in number of rows or overall data volume. The data for the table might also be entirely in the buffer pool, especially for tables that you created recently to perform experiments.

As you monitor or tune cluster performance, make sure to decide whether parallel query is being used in the appropriate contexts. You might adjust the database schema, settings, SQL queries, or even the cluster topology and application connection settings to take advantage of this feature.

To check if a query is using parallel query, check the query plan (also known as the "explain plan") by running the `EXPLAIN` statement. For examples of how SQL statements, clauses, and expressions affect `EXPLAIN` output for parallel query, see [How parallel query works with SQL constructs \(p. 901\)](#).

The following example demonstrates the difference between a traditional query plan and a parallel query plan. This explain plan is from Query 3 from the TPC-H benchmark. Many of the sample queries throughout this section use the tables from the TPC-H dataset. You can get the table definitions, queries, and the `dbgen` program that generates sample data from [the TPC-h website](#).

```
EXPLAIN SELECT l_orderkey,
    sum(l_extendedprice * (1 - l_discount)) AS revenue,
    o_orderdate,
    o_shippriority
FROM customer,
     orders,
     lineitem
WHERE c_mktsegment = 'AUTOMOBILE'
AND c_custkey = o_custkey
AND l_orderkey = o_orderkey
```

```

AND o_orderdate < date '1995-03-13'
AND l_shipdate > date '1995-03-13'
GROUP BY l_orderkey,
    o_orderdate,
    o_shipppriority
ORDER BY revenue DESC,
    o_orderdate LIMIT 10;

```

By default, the query might have a plan like the following. If you don't see hash join used in the query plan, make sure that optimization is turned on first.

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	customer	NULL	ALL	NULL	NULL	NULL	NULL	1480234	Using where; Using temporary; Using filesort
1	SIMPLE	orders	NULL	ALL	NULL	NULL	NULL	NULL	14875240	Using where; Using join buffer (Block Nested Loop)
1	SIMPLE	lineitem	NULL	ALL	NULL	NULL	NULL	NULL	59270573	Using where; Using join buffer (Block Nested Loop)

You can turn on hash join at the session level by issuing the following statement. Afterwards, try the EXPLAIN statement again.

```

# For Aurora MySQL version 3:
SET optimizer_switch='block_nested_loop=on';

# For Aurora MySQL version 2.09 and higher:
SET optimizer_switch='hash_join=on';

```

For information about how to use hash joins effectively, see [Optimizing large Aurora MySQL join queries with hash joins \(p. 1038\)](#).

With hash join turned on but parallel query turned off, the query might have a plan like the following, which uses hash join but not parallel query.

id	select_type	table	rows	Extra
1	SIMPLE	customer	5798330	Using where; Using index; Using temporary; Using filesort
1	SIMPLE	orders	154545408	Using where; Using join buffer (Hash Join Outer table orders)
1	SIMPLE	lineitem	606119300	Using where; Using join buffer (Hash Join Outer table lineitem)

After parallel query is turned on, two steps in this query plan can use the parallel query optimization, as shown under the Extra column in the EXPLAIN output. The I/O-intensive and CPU-intensive processing for those steps is pushed down to the storage layer.

```
+---+...
+-----+
+| id | ...| Extra
+-----+
+| 1 | ...| Using where; Using index; Using temporary; Using filesort
+| 1 | ...| Using where; Using join buffer (Hash Join Outer table orders); Using parallel query (4 columns, 1 filters, 1 exprs; 0 extra)
+| 1 | ...| Using where; Using join buffer (Hash Join Outer table lineitem); Using parallel query (4 columns, 1 filters, 1 exprs; 0 extra)
+-----+
+|
```

For information about how to interpret EXPLAIN output for a parallel query and the parts of SQL statements that parallel query can apply to, see [How parallel query works with SQL constructs \(p. 901\)](#).

The following example output shows the results of running the preceding query on a db.r4.2xlarge instance with a cold buffer pool. The query runs substantially faster when using parallel query.

Note

Because timings depend on many environmental factors, your results might be different. Always conduct your own performance tests to confirm the findings with your own environment, workload, and so on.

```
-- Without parallel query
+-----+-----+-----+
| l_orderkey | revenue | o_orderdate | o_shippriority |
+-----+-----+-----+
| 92511430 | 514726.4896 | 1995-03-06 | 0 |
.
.
| 28840519 | 454748.2485 | 1995-03-08 | 0 |
+-----+-----+-----+
10 rows in set (24 min 49.99 sec)
```

```
-- With parallel query
+-----+-----+-----+
| l_orderkey | revenue | o_orderdate | o_shippriority |
+-----+-----+-----+
| 92511430 | 514726.4896 | 1995-03-06 | 0 |
.
.
| 28840519 | 454748.2485 | 1995-03-08 | 0 |
+-----+-----+-----+
10 rows in set (1 min 49.91 sec)
```

Many of the sample queries throughout this section use the tables from this TPC-H dataset, particularly the PART table, which has 20 million rows and the following definition.

Field	Type	Null	Key	Default	Extra
p_partkey	int(11)	NO	PRI	NULL	
p_name	varchar(55)	NO		NULL	
p_mfgr	char(25)	NO		NULL	
p_brand	char(10)	NO		NULL	

p_type	varchar(25)	NO		NULL		
p_size	int(11)	NO		NULL		
p_container	char(10)	NO		NULL		
p_retailprice	decimal(15, 2)	NO		NULL		
p_comment	varchar(23)	NO		NULL		
+-----+-----+-----+-----+-----+-----+-----+						

Experiment with your workload to get a sense of whether individual SQL statements can take advantage of parallel query. Then use the following monitoring techniques to help verify how often parallel query is used in real workloads over time. For real workloads, extra factors such as concurrency limits apply.

Monitoring parallel query

If your Aurora MySQL cluster uses parallel query, you might see an increase in `VolumeReadIOPS` values. Parallel queries don't use the buffer pool. Thus, although the queries are fast, this optimized processing can result in an increase in read operations and associated charges.

In addition to the Amazon CloudWatch metrics described in [Viewing metrics in the Amazon RDS console \(p. 563\)](#), Aurora provides other global status variables. You can use these global status variables to help monitor parallel query execution. They can give you insights into why the optimizer might use or not use parallel query in a given situation. To access these variables, you can use the [SHOW GLOBAL STATUS](#) command. You can also find these variables listed following.

A parallel query session isn't necessarily a one-to-one mapping with the queries performed by the database. For example, suppose that your query plan has two steps that use parallel query. In that case, the query involves two parallel sessions and the counters for requests attempted and requests successful are incremented by two.

When you experiment with parallel query by issuing `EXPLAIN` statements, expect to see increases in the counters designated as "not chosen" even though the queries aren't actually running. When you work with parallel query in production, you can check if the "not chosen" counters are increasing faster than you expect. At this point, you can adjust so that parallel query runs for the queries that you expect. To do so, you can change your cluster settings, query mix, DB instances where parallel query is turned on, and so on.

These counters are tracked at the DB instance level. When you connect to a different endpoint, you might see different metrics because each DB instance runs its own set of parallel queries. You might also see different metrics when the reader endpoint connects to a different DB instance for each session.

Name	Description
<code>Aurora_pq_request_attempted</code>	The number of parallel query sessions requested. This value might represent more than one session per query, depending on SQL constructs such as subqueries and joins.
<code>Aurora_pq_request_executed</code>	The number of parallel query sessions run successfully.
<code>Aurora_pq_request_failed</code>	The number of parallel query sessions that returned an error to the client. In some cases, a request for a parallel query might fail, for example due to a problem in the storage layer. In these cases, the query part that failed is retried using the nonparallel query mechanism. If the retried query also fails, an error is returned to the client and this counter is incremented.

<code>Aurora_pq_pages_pushed_down</code>	The number of data pages (each with a fixed size of 16 KiB) where parallel query avoided a network transmission to the head node.
<code>Aurora_pq_bytes_returned</code>	The number of bytes for the tuple data structures transmitted to the head node during parallel queries. Divide by 16,384 to compare against <code>Aurora_pq_pages_pushed_down</code> .
<code>Aurora_pq_request_not_chosen</code>	The number of times parallel query wasn't chosen to satisfy a query. This value is the sum of several other more granular counters. An EXPLAIN statement can increment this counter even though the query isn't actually performed.
<code>Aurora_pq_request_not_chosen_below_min_rows</code>	The number of times parallel query wasn't chosen due to the number of rows in the table. An EXPLAIN statement can increment this counter even though the query isn't actually performed.
<code>Aurora_pq_request_not_chosen_small_table</code>	The number of times parallel query wasn't chosen due to the overall size of the table, as determined by number of rows and average row length. An EXPLAIN statement can increment this counter even though the query isn't actually performed.
<code>Aurora_pq_request_not_chosen_high_buffer_pool_pct</code>	The number of times parallel query wasn't chosen because a high percentage of the table data (currently, greater than 95 percent) was already in the buffer pool. In these cases, the optimizer determines that reading the data from the buffer pool is more efficient. An EXPLAIN statement can increment this counter even though the query isn't actually performed.
<code>Aurora_pq_request_not_chosen_few_pages_outside_buffer_pool</code>	The number of times parallel query wasn't chosen, even though less than 95 percent of the table data was in the buffer pool, because there wasn't enough unbuffered table data to make parallel query worthwhile.
<code>Aurora_pq_max_concurrent_requests</code>	The maximum number of parallel query sessions that can run concurrently on this Aurora DB instance. This is a fixed number that depends on the AWS DB instance class.
<code>Aurora_pq_request_in_progress</code>	The number of parallel query sessions currently in progress. This number applies to the particular Aurora DB instance that you are connected to, not the entire Aurora DB cluster. To see if a DB instance is close to its concurrency limit, compare this value to <code>Aurora_pq_max_concurrent_requests</code> .
<code>Aurora_pq_request_throttled</code>	The number of times parallel query wasn't chosen due to the maximum number of concurrent parallel queries already running on a particular Aurora DB instance.

<code>Aurora_pq_request_not_chosen_long_trx</code>	The number of parallel query requests that used the nonparallel query processing path, due to the query being started inside a long-running transaction. An EXPLAIN statement can increment this counter even though the query isn't actually performed.
<code>Aurora_pq_request_not_chosen_unsupported_access</code>	The number of parallel query requests that use the nonparallel query processing path because the WHERE clause doesn't meet the criteria for parallel query. This result can occur if the query doesn't require a data-intensive scan, or if the query is a DELETE or UPDATE statement.
<code>Aurora_pq_request_not_chosen_column_bit</code>	The number of parallel query requests that use the nonparallel query processing path because of an unsupported data type in the list of projected columns.
<code>Aurora_pq_request_not_chosen_column_geom</code>	The number of parallel query requests that use the nonparallel query processing path because the table has columns with the GEOMETRY data type. For information about Aurora MySQL versions that remove this limitation, see Upgrading parallel query clusters to Aurora MySQL version 3 (p. 893) .
<code>Aurora_pq_request_not_chosen_column_lob</code>	The number of parallel query requests that use the nonparallel query processing path because the table has columns with a LOB data type, or VARCHAR columns that are stored externally due to the declared length. For information about Aurora MySQL versions that remove this limitation, see Upgrading parallel query clusters to Aurora MySQL version 3 (p. 893) .
<code>Aurora_pq_request_not_chosen_column_virtual</code>	The number of parallel query requests that use the nonparallel query processing path because the table contains a virtual column.
<code>Aurora_pq_request_not_chosen_custom_charset</code>	The number of parallel query requests that use the nonparallel query processing path because the table has columns with a custom character set.
<code>Aurora_pq_request_not_chosen_fast_ddl</code>	The number of parallel query requests that use the nonparallel query processing path because the table is currently being altered by a fast DDL ALTER Statement.
<code>Aurora_pq_request_not_chosen_full_text_index</code>	The number of parallel query requests that use the nonparallel query processing path because the table has full-text indexes.
<code>Aurora_pq_request_not_chosen_index_hint</code>	The number of parallel query requests that use the nonparallel query processing path because the query includes an index hint.

<code>Aurora_pq_request_not_chosen_innodb_table</code>	The number of parallel query requests that use the nonparallel query processing path because the table uses an unsupported InnoDB row format. Aurora parallel query only applies to the COMPACT, REDUNDANT, and DYNAMIC row formats.
<code>Aurora_pq_request_not_chosen_no_where_clause</code>	The number of parallel query requests that use the nonparallel query processing path because the query doesn't include any WHERE clause.
<code>Aurora_pq_request_not_chosen_range_scan</code>	The number of parallel query requests that use the nonparallel query processing path because the query uses a range scan on an index.
<code>Aurora_pq_request_not_chosen_row_length</code>	The number of parallel query requests that use the nonparallel query processing path because the total combined length of all the columns is too long.
<code>Aurora_pq_request_not_chosen_temporary_table</code>	The number of parallel query requests that use the nonparallel query processing path because the query refers to temporary tables that use the unsupported MyISAM or memory table types.
<code>Aurora_pq_request_not_chosen_tx_isolation</code>	The number of parallel query requests that use the nonparallel query processing path because the query uses an unsupported transaction isolation level. On reader DB instances, parallel query only applies to the REPEATABLE READ and READ COMMITTED isolation levels.
<code>Aurora_pq_request_not_chosen_update_delete</code>	The number of parallel query requests that use the nonparallel query processing path because the query is part of an UPDATE or DELETE statement.

How parallel query works with SQL constructs

In the following section, you can find more detail about why particular SQL statements use or don't use parallel query. This section also details how Aurora MySQL features interact with parallel query. This information can help you diagnose performance issues for a cluster that uses parallel query or understand how parallel query applies for your particular workload.

The decision to use parallel query relies on many factors that occur at the moment that the statement runs. Thus, parallel query might be used for certain queries always, never, or only under certain conditions.

Tip

When you view these examples in HTML, you can use the **Copy** widget in the upper-right corner of each code listing to copy the SQL code to try yourself. Using the **Copy** widget avoids copying the extra characters around the `mysql>` prompt and `->` continuation lines.

Topics

- [EXPLAIN statement \(p. 902\)](#)
- [WHERE clause \(p. 903\)](#)
- [Data definition language \(DDL\) \(p. 905\)](#)
- [Column data types \(p. 905\)](#)

- [Partitioned tables \(p. 905\)](#)
- [Aggregate functions, GROUP BY clauses, and HAVING clauses \(p. 906\)](#)
- [Function calls in WHERE clause \(p. 907\)](#)
- [LIMIT clause \(p. 908\)](#)
- [Comparison operators \(p. 908\)](#)
- [Joins \(p. 908\)](#)
- [Subqueries \(p. 909\)](#)
- [UNION \(p. 910\)](#)
- [Views \(p. 910\)](#)
- [Data manipulation language \(DML\) statements \(p. 910\)](#)
- [Transactions and locking \(p. 911\)](#)
- [B-tree indexes \(p. 913\)](#)
- [Full-text search \(FTS\) indexes \(p. 913\)](#)
- [Virtual columns \(p. 913\)](#)
- [Built-in caching mechanisms \(p. 913\)](#)
- [MyISAM temporary tables \(p. 914\)](#)

EXPLAIN statement

As shown in examples throughout this section, the `EXPLAIN` statement indicates whether each stage of a query is currently eligible for parallel query. It also indicates which aspects of a query can be pushed down to the storage layer. The most important items in the query plan are the following:

- A value other than `NULL` for the `key` column suggests that the query can be performed efficiently using index lookups, and parallel query is unlikely.
- A small value for the `rows` column (a value not in the millions) suggests that the query isn't accessing enough data to make parallel query worthwhile. This means that parallel query is unlikely.
- The `Extra` column shows you if parallel query is expected to be used. This output looks like the following example.

```
Using parallel query (A columns, B filters, C exprs; D extra)
```

The `columns` number represents how many columns are referred to in the query block.

The `filters` number represents the number of `WHERE` predicates representing a simple comparison of a column value to a constant. The comparison can be for equality, inequality, or a range. Aurora can parallelize these kinds of predicates most effectively.

The `exprs` number represents the number of expressions such as function calls, operators, or other expressions that can also be parallelized, though not as effectively as a filter condition.

The `extra` number represents how many expressions can't be pushed down and are performed by the head node.

For example, consider the following `EXPLAIN` output.

```
mysql> explain select p_name, p_mfgr from part
-> where p_brand is not null
-> and upper(p_type) is not null
-> and round(p_retailprice) is not null;
```

```
+----+-----+-----+...+-----+
+-----+-----+-----+-----+
| id | select_type | table |...| rows      | Extra
|     |             |       |   |
+-----+-----+...+-----+
| 1 | SIMPLE      | part  |...| 20427936 | Using where; Using parallel query (5 columns, 1
| filters, 2 exprs; 0 extra) |
+-----+-----+...+-----+
+-----+
```

The information from the `Extra` column shows that five columns are extracted from each row to evaluate the query conditions and construct the result set. One `WHERE` predicate involves a filter, that is, a column that is directly tested in the `WHERE` clause. Two `WHERE` clauses require evaluating more complicated expressions, in this case involving function calls. The `0 extra` field confirms that all the operations in the `WHERE` clause are pushed down to the storage layer as part of parallel query processing.

In cases where parallel query isn't chosen, you can typically deduce the reason from the other columns of the `EXPLAIN` output. For example, the `rows` value might be too small, or the `possible_keys` column might indicate that the query can use an index lookup instead of a data-intensive scan. The following example shows a query where the optimizer can estimate that the query will scan only a small number of rows. It does so based on the characteristics of the primary key. In this case, parallel query isn't required.

```
mysql> explain select count(*) from part where p_partkey between 1 and 100;
+----+-----+-----+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+-----+-----+
| id | select_type | table | type  | possible_keys | key      | key_len | ref   | rows  |
|     |             |       | range | PRIMARY      | PRIMARY  | 4        | NULL  | 99    |
|     |             |       |       | Using where; Using index |
+-----+-----+-----+-----+-----+-----+-----+-----+
```

The output showing whether parallel query will be used takes into account all available factors at the moment that the `EXPLAIN` statement is run. The optimizer might make a different choice when the query is actually run, if the situation changed in the meantime. For example, `EXPLAIN` might report that a statement will use parallel query. But when the query is actually run later, it might not use parallel query based on the conditions then. Such conditions can include several other parallel queries running concurrently. They can also include rows being deleted from the table, a new index being created, too much time passing within an open transaction, and so on.

WHERE clause

For a query to use the parallel query optimization, it *must* include a `WHERE` clause.

The parallel query optimization speeds up many kinds of expressions used in the `WHERE` clause:

- Simple comparisons of a column value to a constant, known as *filters*. These comparisons benefit the most from being pushed down to the storage layer. The number of filter expressions in a query is reported in the `EXPLAIN` output.
- Other kinds of expressions in the `WHERE` clause are also pushed down to the storage layer where possible. The number of such expressions in a query is reported in the `EXPLAIN` output. These expressions can be function calls, `LIKE` operators, `CASE` expressions, and so on.
- Certain functions and operators aren't currently pushed down by parallel query. The number of such expressions in a query is reported as the `extra` counter in the `EXPLAIN` output. The rest of the query can still use parallel query.

- While expressions in the select list aren't pushed down, queries containing such functions can still benefit from reduced network traffic for the intermediate results of parallel queries. For example, queries that call aggregation functions in the select list can benefit from parallel query, even though the aggregation functions aren't pushed down.

For example, the following query does a full-table scan and processes all the values for the `P_BRAND` column. However, it doesn't use parallel query because the query doesn't include any `WHERE` clause.

```
mysql> explain select count(*), p_brand from part group by p_brand;
+-----+-----+-----+-----+-----+-----+
| id | select_type | table | type | possible_keys | key | key_len | ref | rows |
|    |             |       |      |               |     |         |    |      |
+-----+-----+-----+-----+-----+-----+
| 1 | SIMPLE      | part  | ALL  | NULL          | NULL | NULL    | NULL | 20427936 |
|   |              |       |      | Using temporary; Using filesort |
+-----+-----+-----+-----+-----+-----+
```

In contrast, the following query includes `WHERE` predicates that filter the results, so parallel query can be applied:

```
mysql> explain select count(*), p_brand from part where p_name is not null
      -> and p_mfgr in ('Manufacturer#1', 'Manufacturer#3') and p_retailprice > 1000
      -> group by p_brand;
+-----+-----+
| id | ... | rows      | Extra
|    |     |           |      |
+-----+-----+
| 1 | ... | 20427936 | Using where; Using temporary; Using filesort; Using parallel query (5
|   |       |           | columns, 1 filters, 2 exprs; 0 extra) |
+-----+-----+
```

If the optimizer estimates that the number of returned rows for a query block is small, parallel query isn't used for that query block. The following example shows a case where a greater-than operator on the primary key column applies to millions of rows, which causes parallel query to be used. The converse less-than test is estimated to apply to only a few rows and doesn't use parallel query.

```
mysql> explain select count(*) from part where p_partkey > 10;
+-----+-----+
| id | ... | rows      | Extra
|    |     |           |      |
+-----+-----+
| 1 | ... | 20427936 | Using where; Using parallel query (1 columns, 1 filters, 0 exprs; 0
|   |       |           | extra) |
+-----+-----+
```



```
mysql> explain select count(*) from part where p_partkey < 10;
+-----+-----+
| id | ... | rows      | Extra
|    |     |           |      |
+-----+-----+
```

```
| 1 |...| 9 | Using where; Using index |  
+---+---+-----+
```

Data definition language (DDL)

Before Aurora MySQL version 3, parallel query is only available for tables for which no fast data definition language (DDL) operations are pending. In Aurora MySQL version 3, you can use parallel query on a table at the same time as an instant DDL operation. Instant DDL in Aurora MySQL version 3 replaces the fast DDL feature in Aurora MySQL versions 1 and 2. For information about instant DDL, see [Instant DDL \(Aurora MySQL version 3\) \(p. 832\)](#).

Column data types

In Aurora MySQL version 3, parallel query can work with tables containing columns with data types `TEXT`, `BLOB`, `JSON`, and `GEOMETRY`. It can also work with `VARCHAR` and `CHAR` columns with a maximum declared length longer than 768 bytes. If your query refers to any columns containing such large object types, the additional work to retrieve them does add some overhead to query processing. In that case, check if the query can omit the references to those columns. If not, run benchmarks to confirm if such queries are faster with parallel query turned on or turned off.

Before Aurora MySQL version 3, parallel query has these limitations for large object types:

In these earlier versions, `TEXT`, `BLOB`, `JSON`, and `GEOMETRY` data types aren't supported with parallel query. A query that refers to any columns of these types can't use parallel query.

In these earlier versions, variable-length columns (`VARCHAR` and `CHAR` data types) are compatible with parallel query up to a maximum declared length of 768 bytes. A query that refers to any columns of the types declared with a longer maximum length can't use parallel query. For columns that use multibyte character sets, the byte limit takes into account the maximum number of bytes in the character set. For example, for the character set `utf8mb4` (which has a maximum character length of 4 bytes), a `VARCHAR(192)` column is compatible with parallel query but a `VARCHAR(193)` column isn't.

Partitioned tables

You can use partitioned tables with parallel query in Aurora MySQL version 3. Because partitioned tables are represented internally as multiple smaller tables, a query that uses parallel query on a nonpartitioned table might not use parallel query on an identical partitioned table. Aurora MySQL considers whether each partition is large enough to qualify for the parallel query optimization, instead of evaluating the size of the entire table. Check whether the `Aurora_pq_request_not_chosen_small_table` status variable is incremented if a query on a partitioned table doesn't use parallel query when you expect it to.

For example, consider one table partitioned with `PARTITION BY HASH (column) PARTITIONS 2` and another table partitioned with `PARTITION BY HASH (column) PARTITIONS 10`. In the table with two partitions, the partitions are five times as large as the table with ten partitions. Thus, parallel query is more likely to be used for queries against the table with fewer partitions. In the following example, the table `PART_BIG_PARTITIONS` has two partitions and `PART_SMALL_PARTITIONS` has ten partitions. With identical data, parallel query is more likely to be used for the table with fewer big partitions.

```
mysql> explain select count(*), p_brand from part_big_partitions where p_name is not null  
      ->      and p_mfgr in ('Manufacturer#1', 'Manufacturer#3') and p_retailprice > 1000 group  
      by p_brand;  
+-----+-----+-----+  
+-----+-----+-----+  
| id | select_type | table           | partitions | Extra          |  
+-----+-----+-----+
```

```
+-----+-----+
+-----+
| 1 | SIMPLE      | part_big_partitions | p0,p1      | Using where; Using temporary; Using
parallel query (4 columns, 1 filters, 1 exprs; 0 extra; 1 group-by, 1 aggrs) |
+-----+-----+
+
mysql> explain select count(*), p_brand from part_small_partitions where p_name is not null
      ->      and p_mfgr in ('Manufacturer#1', 'Manufacturer#3') and p_retailprice > 1000 group
by p_brand;
+-----+-----+
+-----+
| id | select_type | table           | partitions          | Extra
|
+-----+-----+
+-----+
| 1 | SIMPLE      | part_small_partitions | p0,p1,p2,p3,p4,p5,p6,p7,p8,p9 | Using where;
Using temporary |
+-----+-----+
+-----+
```

Aggregate functions, GROUP BY clauses, and HAVING clauses

Queries involving aggregate functions are often good candidates for parallel query, because they involve scanning large numbers of rows within large tables.

In Aurora MySQL 3, parallel query can optimize aggregate function calls in the select list and the `HAVING` clause.

Before Aurora MySQL 3, aggregate function calls in the select list or the `HAVING` clause aren't pushed down to the storage layer. However, parallel query can still improve the performance of such queries with aggregate functions. It does so by first extracting column values from the raw data pages in parallel at the storage layer. It then transmits those values back to the head node in a compact tuple format instead of as entire data pages. As always, the query requires at least one `WHERE` predicate for parallel query to be activated.

The following simple examples illustrate the kinds of aggregate queries that can benefit from parallel query. They do so by returning intermediate results in compact form to the head node, filtering nonmatching rows from the intermediate results, or both.

```
mysql> explain select sql_no_cache count(distinct p_brand) from part where p_mfgr =
'Manufacturer#5';
+-----+...+
| id | ...| Extra
+-----+...+
| 1 | ...| Using where; Using parallel query (2 columns, 1 filters, 0 exprs; 0 extra) |
+-----+...+
mysql> explain select sql_no_cache p_mfgr from part where p_retailprice > 1000 group by
p_mfgr having count(*) > 100;
+-----+...
+
| id | ...| Extra
|
+-----+...
+
| 1 | ...| Using where; Using temporary; Using filesort; Using parallel query (3 columns, 0
filters, 1 exprs; 0 extra) |
```

```
+----+...
+-----+
+
```

Function calls in WHERE clause

Aurora can apply the parallel query optimization to calls to most built-in functions in the `WHERE` clause. Parallelizing these function calls offloads some CPU work from the head node. Evaluating the predicate functions in parallel during the earliest query stage helps Aurora minimize the amount of data transmitted and processed during later stages.

Currently, the parallelization doesn't apply to function calls in the select list. Those functions are evaluated by the head node, even if identical function calls appear in the `WHERE` clause. The original values from relevant columns are included in the tuples transmitted from the storage nodes back to the head node. The head node performs any transformations such as `UPPER`, `CONCATENATE`, and so on to produce the final values for the result set.

In the following example, parallel query parallelizes the call to `LOWER` because it appears in the `WHERE` clause. Parallel query doesn't affect the calls to `SUBSTR` and `UPPER` because they appear in the select list.

```
mysql> explain select sql_no_cache distinct substr(upper(p_name),1,5) from part
      -> where lower(p_name) like '%cornflower%' or lower(p_name) like '%goldenrod%';
+----+...
+
| id | ... | Extra
  |
+----+...
+
| 1 | ... | Using where; Using temporary; Using parallel query (2 columns, 0 filters, 1
  exprs; 0 extra) |
+----+...
+
+
```

The same considerations apply to other expressions, such as `CASE` expressions or `LIKE` operators. For example, the following example shows that parallel query evaluates the `CASE` expression and `LIKE` operators in the `WHERE` clause.

```
mysql> explain select p_mfgr, p_retailprice from part
      -> where p_retailprice > case p_mfgr
      ->   when 'Manufacturer#1' then 1000
      ->   when 'Manufacturer#2' then 1200
      ->   else 950
      -> end
      -> and p_name like '%vanilla%'
      -> group by p_retailprice;
+----+...
+
| id | ... | Extra
  |
+----+...
+
| 1 | ... | Using where; Using temporary; Using filesort; Using parallel query (4 columns, 0
  filters, 2 exprs; 0 extra) |
+----+...
+
+
```

LIMIT clause

Currently, parallel query isn't used for any query block that includes a `LIMIT` clause. Parallel query might still be used for earlier query phases with `GROUP BY`, `ORDER BY`, or joins.

Comparison operators

The optimizer estimates how many rows to scan to evaluate comparison operators, and determines whether to use parallel query based on that estimate.

The first example following shows that an equality comparison against the primary key column can be performed efficiently without parallel query. The second example following shows that a similar comparison against an unindexed column requires scanning millions of rows and therefore can benefit from parallel query.

```
mysql> explain select * from part where p_partkey = 10;
+----+...+-----+
| id | ...| rows | Extra |
+----+...+-----+
| 1 | ...| 1 | NULL |
+----+...+-----+

mysql> explain select * from part where p_type = 'LARGE BRUSHED BRASS';
+-----+
| id | ...| rows      | Extra
|     |
+-----+
| 1 | ...| 20427936 | Using where; Using parallel query (9 columns, 1 filters, 0 exprs; 0
extra) |
+-----+
```

The same considerations apply for not-equals tests and for range comparisons such as less than, greater than or equal to, or `BETWEEN`. The optimizer estimates the number of rows to scan, and determines whether parallel query is worthwhile based on the overall volume of I/O.

Joins

Join queries with large tables typically involve data-intensive operations that benefit from the parallel query optimization. The comparisons of column values between multiple tables (that is, the join predicates themselves) currently aren't parallelized. However, parallel query can push down some of the internal processing for other join phases, such as constructing the Bloom filter during a hash join. Parallel query can apply to join queries even without a `WHERE` clause. Therefore, a join query is an exception to the rule that a `WHERE` clause is required to use parallel query.

Each phase of join processing is evaluated to check if it is eligible for parallel query. If more than one phase can use parallel query, these phases are performed in sequence. Thus, each join query counts as a single parallel query session in terms of concurrency limits.

For example, when a join query includes `WHERE` predicates to filter the rows from one of the joined tables, that filtering option can use parallel query. As another example, suppose that a join query uses the hash join mechanism, for example to join a big table with a small table. In this case, the table scan to produce the Bloom filter data structure might be able to use parallel query.

Note

Parallel query is typically used for the kinds of resource-intensive queries that benefit from the hash join optimization. The method for turning on the hash join optimization depends on the Aurora MySQL version. For details for each version, see [Turning on hash join for parallel query](#)

clusters (p. 892). For information about how to use hash joins effectively, see [Optimizing large Aurora MySQL join queries with hash joins \(p. 1038\)](#).

```
mysql> explain select count(*) from orders join customer where o_custkey = c_custkey;
+-----+...+-----+-----+-----+...+
+-----+
| id | ... | table   | type   | possible_keys | key           | ... | rows      | Extra
|     |
+-----+...+-----+-----+-----+...+
+-----+
| 1 | ... | customer | index | PRIMARY       | c_nationkey | ... | 15051972 | Using index
|
| 1 | ... | orders    | ALL    | o_custkey     | NULL         | ... | 154545408 | Using join
buffer (Hash Join Outer table orders); Using parallel query (1 columns, 0 filters, 1
exprs; 0 extra)
+-----+...+-----+-----+-----+...+
+-----+
|
```

For a join query that uses the nested loop mechanism, the outermost nested loop block might use parallel query. The use of parallel query depends on the same factors as usual, such as the presence of additional filter conditions in the WHERE clause.

```
mysql> -- Nested loop join with extra filter conditions can use parallel query.
mysql> explain select count(*) from part, partsupp where p_partkey != ps_partkey and p_name
is not null and ps_availqty > 0;
+-----+-----+...+
+-----+
| id | select_type | table   | ... | rows      | Extra
|     |
+-----+-----+...+
+-----+
| 1 | SIMPLE      | part     | ... | 20427936 | Using where; Using parallel query (2
columns, 1 filters, 0 exprs; 0 extra)
| 1 | SIMPLE      | partsupp | ... | 78164450 | Using where; Using join buffer (Block Nested
Loop)
+-----+-----+...+
+-----+
```

Subqueries

The outer query block and inner subquery block might each use parallel query, or not. Whether they do is based on the usual characteristics of the table, WHERE clause, and so on, for each block. For example, the following query uses parallel query for the subquery block but not the outer block.

```
mysql> explain select count(*) from part where
--> p_partkey < (select max(p_partkey) from part where p_name like '%vanilla%');
+-----+-----+...+
+-----+
| id | select_type | ... | rows      | Extra
|     |
+-----+-----+...+
+-----+
| 1 | PRIMARY     | ... | NULL      | Impossible WHERE noticed after reading const tables
| 2 | SUBQUERY     | ... | 20427936 | Using where; Using parallel query (2 columns, 0
filters, 1 exprs; 0 extra)
+-----+
```

+-----+-----+...+
+-----+-----+...+

Currently, correlated subqueries can't use the parallel query optimization.

UNION

Each query block in a UNION query can use parallel query, or not, based on the usual characteristics of the table, WHERE clause, and so on, for each part of the UNION.

```
mysql> explain select p_partkey from part where p_name like '%choco_ate%'
      -> union select p_partkey from part where p_name like '%vanil_a%';
+-----+-----+...+-----+
| id | select_type | ... | rows     | Extra
+-----+-----+...+-----+
|  1 | PRIMARY      | ... | 20427936 | Using where; Using parallel query (2 columns, 0
  filters, 1 exprs; 0 extra) |
|  2 | UNION         | ... | 20427936 | Using where; Using parallel query (2 columns, 0
  filters, 1 exprs; 0 extra) |
| NULL | UNION RESULT | <union1,2> | ... |      NULL | Using temporary
+-----+-----+...+-----+
```

Note

Each UNION clause within the query is run sequentially. Even if the query includes multiple stages that all use parallel query, it only runs a single parallel query at any one time. Therefore, even a complex multistage query only counts as 1 toward the limit of concurrent parallel queries.

Views

The optimizer rewrites any query using a view as a longer query using the underlying tables. Thus, parallel query works the same whether table references are views or real tables. All the same considerations about whether to use parallel query for a query, and which parts are pushed down, apply to the final rewritten query.

For example, the following query plan shows a view definition that usually doesn't use parallel query. When the view is queried with additional WHERE clauses, Aurora MySQL uses parallel query.

```
mysql> create view part_view as select * from part;
mysql> explain select count(*) from part_view where p_partkey is not null;
+-----+...+-----+
| id | ... | rows     | Extra
+-----+...+-----+
|  1 | ... | 20427936 | Using where; Using parallel query (1 columns, 0 filters, 0 exprs; 1
  extra) |
+-----+...+-----+
```

Data manipulation language (DML) statements

The `INSERT` statement can use parallel query for the `SELECT` phase of processing, if the `SELECT` part meets the other conditions for parallel query.

```
mysql> create table part_subset like part;
mysql> explain insert into part_subset select * from part where p_mfgr = 'Manufacturer#1';
+-----+...+-----+
+-----+...+-----+
| id | ... | rows      | Extra |
+-----+...+-----+
| 1  | ... | 20427936 | Using where; Using parallel query (9 columns, 1 filters, 0 exprs; 0 extra) |
+-----+...+-----+
```

Note

Typically, after an `INSERT` statement, the data for the newly inserted rows is in the buffer pool. Therefore, a table might not be eligible for parallel query immediately after inserting a large number of rows. Later, after the data is evicted from the buffer pool during normal operation, queries against the table might begin using parallel query again.

The `CREATE TABLE AS SELECT` statement doesn't use parallel query, even if the `SELECT` portion of the statement would otherwise be eligible for parallel query. The DDL aspect of this statement makes it incompatible with parallel query processing. In contrast, in the `INSERT ... SELECT` statement, the `SELECT` portion can use parallel query.

Parallel query is never used for `DELETE` or `UPDATE` statements, regardless of the size of the table and predicates in the `WHERE` clause.

```
mysql> explain delete from part where p_name is not null;
+-----+...+-----+
| id | select_type | ... | rows      | Extra |
+-----+...+-----+
| 1  | SIMPLE      | ... | 20427936 | Using where |
+-----+...+-----+
```

Transactions and locking

You can use all the isolation levels on the Aurora primary instance.

On Aurora reader DB instances, parallel query applies to statements performed under the `REPEATABLE READ` isolation level. Aurora MySQL versions 1.23 and 2.09 or higher can also use the `READ COMMITTED` isolation level on reader DB instances. `REPEATABLE READ` is the default isolation level for Aurora reader DB instances. To use `READ COMMITTED` isolation level on reader DB instances requires setting the `aurora_read_replica_read_committed` configuration option at the session level. The `READ COMMITTED` isolation level for reader instances complies with SQL standard behavior. However, the isolation is less strict on reader instances than when queries use `READ COMMITTED` isolation level on the writer instance.

For more information about Aurora isolation levels, especially the differences in `READ COMMITTED` between writer and reader instances, see [Aurora MySQL isolation levels \(p. 1070\)](#).

After a big transaction is finished, the table statistics might be stale. Such stale statistics might require an `ANALYZE TABLE` statement before Aurora can accurately estimate the number of rows. A large-scale DML statement might also bring a substantial portion of the table data into the buffer pool. Having this data in the buffer pool can lead to parallel query being chosen less frequently for that table until the data is evicted from the pool.

When your session is inside a long-running transaction (by default, 10 minutes), further queries inside that session don't use parallel query. A timeout can also occur during a single long-running query. This

type of timeout might happen if the query runs for longer than the maximum interval (currently 10 minutes) before the parallel query processing starts.

You can reduce the chance of starting long-running transactions accidentally by setting `autocommit=1` in MySQL sessions where you perform ad hoc (one-time) queries. Even a `SELECT` statement against a table begins a transaction by creating a read view. A *read view* is a consistent set of data for subsequent queries that remains until the transaction is committed. Be aware of this restriction also when using JDBC or ODBC applications with Aurora, because such applications might run with the `autocommit` setting turned off.

The following example shows how, with the `autocommit` setting turned off, running a query against a table creates a read view that implicitly begins a transaction. Queries that are run shortly afterward can still use parallel query. However, after a pause of several minutes, queries are no longer eligible for parallel query. Ending the transaction with `COMMIT` or `ROLLBACK` restores parallel query eligibility.

```
mysql> set autocommit=0;

mysql> explain select sql_no_cache count(*) from part where p_retailprice > 10.0;
+-----+...+-----+
+-----+...+-----+
| id | ... | rows      | Extra
|     |
+-----+...+-----+
|   1 | ... | 2976129 | Using where; Using parallel query (1 columns, 1 filters, 0 exprs; 0
extra)
+-----+...+-----+
+-----+...+-----+
+-----+...+-----+
mysql> select sleep(720); explain select sql_no_cache count(*) from part where
p_retailprice > 10.0;
+-----+
| sleep(720) |
+-----+
|          0 |
+-----+
1 row in set (12 min 0.00 sec)

+-----+...+-----+-----+
| id | ... | rows      | Extra      |
+-----+...+-----+-----+
|   1 | ... | 2976129 | Using where |
+-----+...+-----+-----+
mysql> commit;

mysql> explain select sql_no_cache count(*) from part where p_retailprice > 10.0;
+-----+...+-----+
+-----+...+-----+
| id | ... | rows      | Extra
|     |
+-----+...+-----+
|   1 | ... | 2976129 | Using where; Using parallel query (1 columns, 1 filters, 0 exprs; 0
extra)
+-----+...+-----+
+-----+...+-----+
```

To see how many times queries weren't eligible for parallel query because they were inside long-running transactions, check the status variable `Aurora_pq_request_not_chosen_long_trx`.

```
mysql> show global status like '%pq%trx%';
+-----+-----+
```

Variable_name	Value
Aurora_pg_request_not_chosen_long_trx	4

Any `SELECT` statement that acquires locks, such as the `SELECT FOR UPDATE` or `SELECT LOCK IN SHARE MODE` syntax, can't use parallel query.

Parallel query can work for a table that is locked by a `LOCK TABLES` statement.

```
mysql> explain select o_orderpriority, o_shipppriority from orders where o_clerk =
'Clerk#000095055';
+-----+-----+
| id | ... | rows      | Extra |
+-----+-----+
| 1  | ... | 154545408 | Using where; Using parallel query (3 columns, 1 filters, 0 exprs; 0
extra) |
+-----+-----+
mysql> explain select o_orderpriority, o_shipppriority from orders where o_clerk =
'Clerk#000095055' for update;
+-----+-----+
| id | ... | rows      | Extra      |
+-----+-----+
| 1  | ... | 154545408 | Using where |
+-----+-----+
```

B-tree indexes

The statistics gathered by the `ANALYZE TABLE` statement help the optimizer to decide when to use parallel query or index lookups, based on the characteristics of the data for each column. Keep statistics current by running `ANALYZE TABLE` after DML operations that make substantial changes to the data within a table.

If index lookups can perform a query efficiently without a data-intensive scan, Aurora might use index lookups. Doing so avoids the overhead of parallel query processing. There are also concurrency limits on the number of parallel queries that can run simultaneously on any Aurora DB cluster. Make sure to use best practices for indexing your tables, so that your most frequent and most highly concurrent queries use index lookups.

Full-text search (FTS) indexes

Currently, parallel query isn't used for tables that contain a full-text search index, regardless of whether the query refers to such indexed columns or uses the `MATCH` operator.

Virtual columns

Currently, parallel query isn't used for tables that contain a virtual column, regardless of whether the query refers to any virtual columns.

Built-in caching mechanisms

Aurora includes built-in caching mechanisms, namely the buffer pool and the query cache. The Aurora optimizer chooses between these caching mechanisms and parallel query depending on which one is most effective for a particular query.

When a parallel query filters rows and transforms and extracts column values, data is transmitted back to the head node as tuples rather than as data pages. Therefore, running a parallel query doesn't add any pages to the buffer pool, or evict pages that are already in the buffer pool.

Aurora checks the number of pages of table data that are present in the buffer pool, and what proportion of the table data that number represents. Aurora uses that information to determine whether it is more efficient to use parallel query (and bypass the data in the buffer pool). Alternatively, Aurora might use the nonparallel query processing path, which uses data cached in the buffer pool. Which pages are cached and how data-intensive queries affect caching and eviction depends on configuration settings related to the buffer pool. Therefore, it can be hard to predict whether any particular query uses parallel query, because the choice depends on the ever-changing data within the buffer pool.

Also, Aurora imposes concurrency limits on parallel queries. Because not every query uses parallel query, tables that are accessed by multiple queries simultaneously typically have a substantial portion of their data in the buffer pool. Therefore, Aurora often doesn't choose these tables for parallel queries.

When you run a sequence of nonparallel queries on the same table, the first query might be slow due to the data not being in the buffer pool. Then the second and subsequent queries are much faster because the buffer pool is now "warmed up". Parallel queries typically show consistent performance from the very first query against the table. When conducting performance tests, benchmark the nonparallel queries with both a cold and a warm buffer pool. In some cases, the results with a warm buffer pool can compare well to parallel query times. In these cases, consider factors such as the frequency of queries against that table. Also consider whether it is worthwhile to keep the data for that table in the buffer pool.

The query cache avoids rerunning a query when an identical query is submitted and the underlying table data hasn't changed. Queries optimized by parallel query feature can go into the query cache, effectively making them instantaneous when run again.

Note

When conducting performance comparisons, the query cache can produce artificially low timing numbers. Therefore, in benchmark-like situations, you can use the `sql_no_cache` hint. This hint prevents the result from being served from the query cache, even if the same query had been run previously. The hint comes immediately after the `SELECT` statement in a query. Many parallel query examples in this topic include this hint, to make query times comparable between versions of the query for which parallel query is turned on and turned off.

Make sure that you remove this hint from your source when you move to production use of parallel query.

MyISAM temporary tables

The parallel query optimization only applies to InnoDB tables. Because Aurora MySQL uses MyISAM behind the scenes for temporary tables, internal query phases involving temporary tables never use parallel query. These query phases are indicated by `Using temporary` in the `EXPLAIN` output.

Using Advanced Auditing with an Amazon Aurora MySQL DB cluster

You can use the high-performance Advanced Auditing feature in Amazon Aurora MySQL to audit database activity. To do so, you enable the collection of audit logs by setting several DB cluster parameters. When Advanced Auditing is enabled, you can use it to log any combination of supported events.

You can view or download the audit logs to review the audit information for one DB instance at a time. To do so, you can use the procedures in [Monitoring Amazon Aurora log files \(p. 695\)](#).

Tip

For an Aurora DB cluster containing multiple DB instances, you might find it more convenient to examine the audit logs for all instances in the cluster. To do so, you can use CloudWatch Logs. You can turn on a setting at the cluster level to publish the Aurora MySQL audit log data to a log group in CloudWatch. Then you can view, filter, and search the audit logs through the CloudWatch interface. For more information, see [Publishing Amazon Aurora MySQL logs to Amazon CloudWatch Logs \(p. 1017\)](#).

Enabling Advanced Auditing

Use the parameters described in this section to enable and configure Advanced Auditing for your DB cluster.

Use the `server_audit_logging` parameter to enable or disable Advanced Auditing.

Use the `server_audit_events` parameter to specify what events to log.

Use the `server_audit_incl_users` and `server_audit_excl_users` parameters to specify who gets audited. By default, all users are audited. For details about how these parameters work when one or both are left empty, or the same user names are specified in both, see [server_audit_incl_users \(p. 916\)](#) and [server_audit_excl_users \(p. 916\)](#).

Configure Advanced Auditing by setting these parameters in the parameter group used by your DB cluster. You can use the procedure shown in [Modifying parameters in a DB parameter group \(p. 347\)](#) to modify DB cluster parameters using the AWS Management Console. You can use the `modify-db-cluster-parameter-group` AWS CLI command or the `ModifyDBClusterParameterGroup` Amazon RDS API operation to modify DB cluster parameters programmatically.

Modifying these parameters doesn't require a DB cluster restart when the parameter group is already associated with your cluster. When you associate the parameter group with the cluster for the first time, a cluster restart is required.

Topics

- [server_audit_logging \(p. 915\)](#)
- [server_audit_events \(p. 915\)](#)
- [server_audit_incl_users \(p. 916\)](#)
- [server_audit_excl_users \(p. 916\)](#)

server_audit_logging

Enables or disables Advanced Auditing. This parameter defaults to OFF; set it to ON to enable Advanced Auditing.

No audit data appears in the logs unless you also define one or more types of events to audit using the `server_audit_events` parameter.

To confirm that audit data is logged for a DB instance, check that some log files for that instance have names of the form `audit/audit.log.other_identifying_information`. To see the names of the log files, follow the procedure in [Viewing and listing database log files \(p. 695\)](#).

server_audit_events

Contains the comma-delimited list of events to log. Events must be specified in all caps, and there should be no white space between the list elements, for example: `CONNECT,QUERY_DDL`. This parameter defaults to an empty string.

You can log any combination of the following events:

- CONNECT – Logs both successful and failed connections and also disconnections. This event includes user information.
- QUERY – Logs all queries in plain text, including queries that fail due to syntax or permission errors.

Tip

With this event type turned on, the audit data includes information about the continuous monitoring and health-checking information that Aurora does automatically. If you are only interested in particular kinds of operations, you can use the more specific kinds of events.

You can also use the CloudWatch interface to search in the logs for events related to specific databases, tables, or users.

- QUERY_DCL – Similar to the QUERY event, but returns only data control language (DCL) queries (GRANT, REVOKE, and so on).
- QUERY_DDL – Similar to the QUERY event, but returns only data definition language (DDL) queries (CREATE, ALTER, and so on).
- QUERY_DML – Similar to the QUERY event, but returns only data manipulation language (DML) queries (INSERT, UPDATE, and so on, and also SELECT).
- TABLE – Logs the tables that were affected by query execution.

server_audit_incl_users

Contains the comma-delimited list of user names for users whose activity is logged. There should be no white space between the list elements, for example: user_3,user_4. This parameter defaults to an empty string. The maximum length is 1024 characters. Specified user names must match corresponding values in the User column of the mysql.user table. For more information about user names, see [the MySQL documentation](#).

If server_audit_incl_users and server_audit_excl_users are both empty (the default), all users are audited.

If you add users to server_audit_incl_users and leave server_audit_excl_users empty, then only those users are audited.

If you add users to server_audit_excl_users and leave server_audit_incl_users empty, then all users are audited, except for those listed in server_audit_excl_users.

If you add the same users to both server_audit_excl_users and server_audit_incl_users, then those users are audited. When the same user is listed in both settings, server_audit_incl_users is given higher priority.

Connect and disconnect events aren't affected by this variable; they are always logged if specified. A user is logged even if that user is also specified in the server_audit_excl_users parameter, because server_audit_incl_users has higher priority.

server_audit_excl_users

Contains the comma-delimited list of user names for users whose activity isn't logged. There should be no white space between the list elements, for example: rdsadmin,user_1,user_2. This parameter defaults to an empty string. The maximum length is 1024 characters. Specified user names must match corresponding values in the User column of the mysql.user table. For more information about user names, see [the MySQL documentation](#).

If server_audit_incl_users and server_audit_excl_users are both empty (the default), all users are audited.

If you add users to `server_audit_excl_users` and leave `server_audit_incl_users` empty, then only those users that you list in `server_audit_excl_users` are not audited, and all other users are.

If you add the same users to both `server_audit_excl_users` and `server_audit_incl_users`, then those users are audited. When the same user is listed in both settings, `server_audit_incl_users` is given higher priority.

Connect and disconnect events aren't affected by this variable; they are always logged if specified. A user is logged if that user is also specified in the `server_audit_incl_users` parameter, because that setting has higher priority than `server_audit_excl_users`.

Viewing audit logs

You can view and download the audit logs by using the console. On the **Databases** page, choose the DB instance to show its details, then scroll to the **Logs** section. The audit logs produced by the Advanced Auditing feature have names of the form `audit/audit.log.other_identifying_information`.

To download a log file, choose that file in the **Logs** section and then choose **Download**.

You can also get a list of the log files by using the `describe-db-log-files` AWS CLI command. You can download the contents of a log file by using the `download-db-log-file-portion` AWS CLI command. For more information, see [Viewing and listing database log files \(p. 695\)](#) and [Downloading a database log file \(p. 696\)](#).

Audit log details

Log files are represented as comma-separated variable (CSV) files in UTF-8 format. The audit log is stored separately on the local (ephemeral) storage of each instance. Each Aurora instance distributes writes across four log files at a time. The maximum size of the logs is 100 MB in aggregate. When this non-configurable limit is reached, Aurora rotates the files and generates four new files.

Tip

Log file entries are not in sequential order. To order the entries, use the timestamp value. To see the latest events, you might have to review all log files. For more flexibility in sorting and searching the log data, turn on the setting to upload the audit logs to CloudWatch and view them using the CloudWatch interface.

To view audit data with more types of fields and with output in JSON format, you can also use the Database Activity Streams feature. For more information, see [Monitoring Amazon Aurora with Database Activity Streams \(p. 714\)](#).

The audit log files include the following comma-delimited information in rows, in the specified order:

Field	Description
<code>timestamp</code>	The Unix time stamp for the logged event with microsecond precision.
<code>serverhost</code>	The name of the instance that the event is logged for.
<code>username</code>	The connected user name of the user.
<code>host</code>	The host that the user connected from.
<code>connectionid</code>	The connection ID number for the logged operation.
<code>queryid</code>	The query ID number, which can be used for finding the relational table events and related queries. For <code>TABLE</code> events, multiple lines are added.
<code>operation</code>	The recorded action type. Possible values are: <code>CONNECT</code> , <code>QUERY</code> , <code>READ</code> , <code>WRITE</code> , <code>CREATE</code> , <code>ALTER</code> , <code>RENAME</code> , and <code>DROP</code> .

Field	Description
database	The active database, as set by the <code>USE</code> command.
object	For <code>QUERY</code> events, this value indicates the query that the database performed. For <code>TABLE</code> events, it indicates the table name.
retcode	The return code of the logged operation.

Single-master replication with Amazon Aurora MySQL

The Aurora MySQL replication features are key to the high availability and performance of your cluster. Aurora makes it easy to create or resize clusters with up to 15 Aurora Replicas.

All the replicas work from the same underlying data. If some database instances go offline, others remain available to continue processing queries or to take over as the writer if needed. Aurora automatically spreads your read-only connections across multiple database instances, helping an Aurora cluster to support query-intensive workloads.

Following, you can find information about how Aurora MySQL replication works and how to fine-tune replication settings for best availability and performance.

Note

Following, you can learn about replication features for Aurora clusters using single-master replication. This kind of cluster is the default for Aurora. For information about Aurora multi-master clusters, see [Working with Aurora multi-master clusters \(p. 958\)](#).

Topics

- [Using Aurora replicas \(p. 918\)](#)
- [Replication options for Amazon Aurora MySQL \(p. 919\)](#)
- [Performance considerations for Amazon Aurora MySQL replication \(p. 920\)](#)
- [Zero-downtime restart \(ZDR\) for Amazon Aurora MySQL \(p. 920\)](#)
- [Monitoring Amazon Aurora MySQL replication \(p. 922\)](#)
- [Replicating Amazon Aurora MySQL DB clusters across AWS Regions \(p. 922\)](#)
- [Replication between Aurora and MySQL or between Aurora and another Aurora DB cluster \(binary log replication\) \(p. 932\)](#)
- [Using GTID-based replication for Aurora MySQL \(p. 954\)](#)

Using Aurora replicas

Aurora Replicas are independent endpoints in an Aurora DB cluster, best used for scaling read operations and increasing availability. Up to 15 Aurora Replicas can be distributed across the Availability Zones that a DB cluster spans within an AWS Region. Although the DB cluster volume is made up of multiple copies of the data for the DB cluster, the data in the cluster volume is represented as a single, logical volume to the primary instance and to Aurora Replicas in the DB cluster. For more information about Aurora Replicas, see [Aurora Replicas \(p. 70\)](#).

Aurora Replicas work well for read scaling because they are fully dedicated to read operations on your cluster volume. Write operations are managed by the primary instance. Because the cluster volume is shared among all instances in your Aurora MySQL DB cluster, no additional work is required to replicate

a copy of the data for each Aurora Replica. In contrast, MySQL read replicas must replay, on a single thread, all write operations from the source DB instance to their local data store. This limitation can affect the ability of MySQL read replicas to support large volumes of read traffic.

With Aurora MySQL, when an Aurora Replica is deleted, its instance endpoint is removed immediately, and the Aurora Replica is removed from the reader endpoint. If there are statements running on the Aurora Replica that is being deleted, there is a three minute grace period. Existing statements can finish gracefully during the grace period. When the grace period ends, the Aurora Replica is shut down and deleted.

Important

Aurora Replicas for Aurora MySQL always use the `REPEATABLE READ` default transaction isolation level for operations on InnoDB tables. You can use the `SET TRANSACTION ISOLATION LEVEL` command to change the transaction level only for the primary instance of an Aurora MySQL DB cluster. This restriction avoids user-level locks on Aurora Replicas, and allows Aurora Replicas to scale to support thousands of active user connections while still keeping replica lag to a minimum.

Note

DDL statements that run on the primary instance might interrupt database connections on the associated Aurora Replicas. If an Aurora Replica connection is actively using a database object, such as a table, and that object is modified on the primary instance using a DDL statement, the Aurora Replica connection is interrupted.

Note

The China (Ningxia) Region does not support cross-Region read replicas.

Replication options for Amazon Aurora MySQL

You can set up replication between any of the following options:

- Two Aurora MySQL DB clusters in different AWS Regions, by creating a cross-Region read replica of an Aurora MySQL DB cluster.

For more information, see [Replicating Amazon Aurora MySQL DB clusters across AWS Regions \(p. 922\)](#).

- Two Aurora MySQL DB clusters in the same AWS Region, by using MySQL binary log (binlog) replication.

For more information, see [Replication between Aurora and MySQL or between Aurora and another Aurora DB cluster \(binary log replication\) \(p. 932\)](#).

- An RDS for MySQL DB instance as the source and an Aurora MySQL DB cluster, by creating an Aurora read replica of an RDS for MySQL DB instance.

You can use this approach to bring existing and ongoing data changes into Aurora MySQL during migration to Aurora. For more information, see [Migrating data from a MySQL DB instance to an Amazon Aurora MySQL DB cluster by using a DB snapshot \(p. 797\)](#).

You can also use this approach to increase the scalability of read queries for your data. You do so by querying the data using one or more DB instances within a read-only Aurora MySQL cluster. For more information, see [Using Amazon Aurora to scale reads for your MySQL database \(p. 945\)](#).

- An Aurora MySQL DB cluster in one AWS Region and up to five Aurora read-only Aurora MySQL DB clusters in different Regions, by creating an Aurora global database.

You can use an Aurora global database to support applications with a world-wide footprint. The primary Aurora MySQL DB cluster has a Writer instance and up to 15 Aurora Replicas. The read-only secondary Aurora MySQL DB clusters can each be made up of as many as 16 Aurora Replicas. For more information, see [Using Amazon Aurora global databases \(p. 225\)](#).

Note

Rebooting the primary instance of an Amazon Aurora DB cluster also automatically reboots the Aurora Replicas for that DB cluster, to re-establish an entry point that guarantees read/write consistency across the DB cluster.

Performance considerations for Amazon Aurora MySQL replication

The following features help you to fine-tune the performance of Aurora MySQL replication.

Starting in Aurora MySQL 1.17.4, the replica log compression feature automatically reduces network bandwidth for replication messages. Because each message is transmitted to all Aurora Replicas, the benefits are greater for larger clusters. This feature involves some CPU overhead on the writer node to perform the compression. Thus, the feature is only available on the 8xlarge and 16xlarge instance classes, which have high CPU capacity. It is enabled by default on these instance classes. You can control this feature by turning off the `aurora_enable_replica_log_compression` parameter. For example, you might turn off replica log compression for larger instance classes if your writer node is near its maximum CPU capacity.

Starting in Aurora MySQL 1.17.4, the binlog filtering feature automatically reduces network bandwidth for replication messages. Because the Aurora Replicas don't use the binlog information that is included in the replication messages, that data is omitted from the messages sent to those nodes. You control this feature by changing the `aurora_enable_repl_bin_log_filtering` parameter. This parameter is on by default. Because this optimization is intended to be transparent, you might turn off this setting only during diagnosis or troubleshooting for issues related to replication. For example, you can do so to match the behavior of an older Aurora MySQL cluster where this feature was not available.

Zero-downtime restart (ZDR) for Amazon Aurora MySQL

The zero-downtime restart (ZDR) feature can preserve some or all of the active connections to DB instances during certain kinds of restarts. ZDR applies to restarts that Aurora performs automatically to resolve error conditions, for example when a replica begins to lag too far behind the source.

Important

The ZDR mechanism operates on a best-effort basis. The Aurora MySQL versions, instance classes, error conditions, compatible SQL operations, and other factors that determine where ZDR applies are subject to change at any time.

In Aurora MySQL 1.* versions where ZDR is available, you turn on this feature by turning on the `aurora_enable_zdr` parameter in the cluster parameter group. ZDR for Aurora MySQL 2.* requires version 2.10 and higher. ZDR is available in all minor versions of Aurora MySQL 3.*. In Aurora MySQL version 2 and 3, the ZDR mechanism is turned on by default and Aurora doesn't use the `aurora_enable_zdr` parameter.

Aurora reports on the **Events** page activities related to zero-downtime restart. Aurora records an event when it attempts a restart using the ZDR mechanism. This event states why Aurora performs the restart. Then Aurora records another event when the restart finishes. This final event reports how long the process took, and how many connections were preserved or dropped during the restart. You can consult the database error log to see more details about what happened during the restart.

Although connections remain intact following a successful ZDR operation, some variables and features are reinitialized. The following kinds of information aren't preserved through a restart caused by zero-downtime restart:

- Global variables. Aurora restores session variables, but it doesn't restore global variables after the restart.
- Status variables. In particular, the uptime value reported by the engine status is reset.
- `LAST_INSERT_ID`.
- In-memory `auto_increment` state for tables. The in-memory auto-increment state is reinitialized. For more information about auto-increment values, see [MySQL Reference Manual](#).
- Diagnostic information from `INFORMATION_SCHEMA` and `PERFORMANCE_SCHEMA` tables. This diagnostic information also appears in the output of commands such as `SHOW PROFILE` and `SHOW PROFILES`.

The following table shows the versions, instance roles, instance classes, and other circumstances that determine whether Aurora can use the ZDR mechanism when restarting DB instances in your cluster.

Aurora MySQL version	Does ZDR apply to the writer?	Does ZDR apply to readers?	Notes
Aurora MySQL version 1.*, 1.17.3 and lower	No	No	ZDR isn't available for these versions.
Aurora MySQL version 1.*, 1.17.4 and higher	No	Yes	<p>In these Aurora MySQL versions, the following conditions apply to the ZDR mechanism:</p> <ul style="list-style-type: none"> • Aurora doesn't use the ZDR mechanism if binary logging is turned on for the DB instance. • Aurora rolls back any transactions that are in progress on active connections. Your application must retry the transactions. • Aurora cancels any connections that use TLS/SSL, temporary tables, table locks, or user locks.
Aurora MySQL version 2.*, before 2.10.0	No	No	ZDR isn't available for these versions. The <code>aurora_enable_zdr</code> parameter isn't available in the default cluster parameter group for Aurora MySQL version 2.
Aurora MySQL version 2.*, 2.10.0 and higher	Yes	Yes	<p>The ZDR mechanism is always enabled.</p> <p>In these Aurora MySQL versions, the following conditions apply to the ZDR mechanism:</p> <ul style="list-style-type: none"> • Aurora rolls back any transactions that are in progress on active connections. Your application must retry the transactions. • Aurora cancels any connections that use TLS/SSL, temporary tables, table locks, or user locks.
Aurora MySQL version 3.*	Yes	Yes	<p>The ZDR mechanism is always enabled.</p> <p>The same conditions apply as in Aurora MySQL version 2.10. ZDR applies to all instance classes.</p>

Monitoring Amazon Aurora MySQL replication

Read scaling and high availability depend on minimal lag time. You can monitor how far an Aurora Replica is lagging behind the primary instance of your Aurora MySQL DB cluster by monitoring the Amazon CloudWatch `AuroraReplicaLag` metric. The `AuroraReplicaLag` metric is recorded in each Aurora Replica.

The primary DB instance also records the `AuroraReplicaLagMaximum` and `AuroraReplicaLag` Amazon CloudWatch metrics. The `AuroraReplicaLagMaximum` metric records the maximum amount of lag between the primary DB instance and each Aurora Replica in the DB cluster. The `AuroraReplicaLag` metric records the minimum amount of lag between the primary DB instance and each Aurora Replica in the DB cluster.

If you need the most current value for Aurora Replica lag, you can query the `recrystallisations` table on the primary instance in your Aurora MySQL DB cluster and check the value in the `Replica_lag_in_msec` column. This column value is provided to Amazon CloudWatch as the value for the `AuroraReplicaLag` metric. The Aurora Replica lag is also recorded on each Aurora Replica in the `INFORMATION_SCHEMA.REPLICA_HOST_STATUS` table in your Aurora MySQL DB cluster.

For more information on monitoring RDS instances and CloudWatch metrics, see [Monitoring metrics in an Amazon Aurora cluster \(p. 541\)](#).

Replicating Amazon Aurora MySQL DB clusters across AWS Regions

You can create an Amazon Aurora MySQL DB cluster as a read replica in a different AWS Region than the source DB cluster. Taking this approach can improve your disaster recovery capabilities, let you scale read operations into an AWS Region that is closer to your users, and make it easier to migrate from one AWS Region to another.

You can create read replicas of both encrypted and unencrypted DB clusters. The read replica must be encrypted if the source DB cluster is encrypted.

For each source DB cluster, you can have up to five cross-Region DB clusters that are read replicas.

Note

As an alternative to cross-Region read replicas, you can scale read operations with minimal lag time by using an Aurora global database. An Aurora global database has a primary Aurora DB cluster in one AWS Region and up to five secondary read-only DB clusters in different Regions. Each secondary DB cluster can include up to 16 (rather than 15) Aurora Replicas. Replication from the primary DB cluster to all secondaries is handled by the Aurora storage layer rather than by the database engine, so lag time for replicating changes is minimal—typically, less than 1 second. Keeping the database engine out of the replication process means that the database engine is dedicated to processing workloads. It also means that you don't need to configure or manage Aurora MySQL's binlog (binary logging) replication. To learn more, see [Using Amazon Aurora global databases \(p. 225\)](#).

When you create an Aurora MySQL DB cluster read replica in another AWS Region, you should be aware of the following:

- Both your source DB cluster and your cross-Region read replica DB cluster can have up to 15 Aurora Replicas, along with the primary instance for the DB cluster. By using this functionality, you can scale read operations for both your source AWS Region and your replication target AWS Region.
- In a cross-Region scenario, there is more lag time between the source DB cluster and the read replica due to the longer network channels between AWS Regions.

- Data transferred for cross-Region replication incurs Amazon RDS data transfer charges. The following cross-Region replication actions generate charges for the data transferred out of the source AWS Region:
 - When you create the read replica, Amazon RDS takes a snapshot of the source cluster and transfers the snapshot to the AWS Region that holds the read replica.
 - For each data modification made in the source databases, Amazon RDS transfers data from the source region to the AWS Region that holds the read replica.

For more information about Amazon RDS data transfer pricing, see [Amazon Aurora pricing](#).

- You can run multiple concurrent create or delete actions for read replicas that reference the same source DB cluster. However, you must stay within the limit of five read replicas for each source DB cluster.
- For replication to operate effectively, each read replica should have the same amount of compute and storage resources as the source DB cluster. If you scale the source DB cluster, you should also scale the read replicas.

Topics

- [Before you begin \(p. 923\)](#)
- [Creating an Amazon Aurora MySQL DB cluster that is a cross-Region read replica \(p. 923\)](#)
- [Viewing Amazon Aurora MySQL cross-Region replicas \(p. 930\)](#)
- [Promoting a read replica to be a DB cluster \(p. 930\)](#)
- [Troubleshooting Amazon Aurora MySQL cross Region replicas \(p. 931\)](#)

Before you begin

Before you can create an Aurora MySQL DB cluster that is a cross-Region read replica, you must turn on binary logging on your source Aurora MySQL DB cluster. Cross-region replication for Aurora MySQL uses MySQL binary replication to replay changes on the cross-Region read replica DB cluster.

To turn on binary logging on an Aurora MySQL DB cluster, update the `binlog_format` parameter for your source DB cluster. The `binlog_format` parameter is a cluster-level parameter that is in the default cluster parameter group. If your DB cluster uses the default DB cluster parameter group, create a new DB cluster parameter group to modify `binlog_format` settings. We recommend that you set the `binlog_format` to `MIXED`. However, you can also set `binlog_format` to `ROW` or `STATEMENT` if you need a specific binlog format. Reboot your Aurora DB cluster for the change to take effect.

For more information about using binary logging with Aurora MySQL, see [Replication between Aurora and MySQL or between Aurora and another Aurora DB cluster \(binary log replication\) \(p. 932\)](#). For more information about modifying Aurora MySQL configuration parameters, see [Amazon Aurora DB cluster and DB instance parameters \(p. 341\)](#) and [Working with DB parameter groups and DB cluster parameter groups \(p. 339\)](#).

Creating an Amazon Aurora MySQL DB cluster that is a cross-Region read replica

You can create an Aurora DB cluster that is a cross-Region read replica by using the AWS Management Console, the AWS Command Line Interface (AWS CLI), or the Amazon RDS API. You can create cross-Region read replicas from both encrypted and unencrypted DB clusters.

When you create a cross-Region read replica for Aurora MySQL by using the AWS Management Console, Amazon RDS creates a DB cluster in the target AWS Region, and then automatically creates a DB instance that is the primary instance for that DB cluster.

When you create a cross-Region read replica using the AWS CLI or RDS API, you first create the DB cluster in the target AWS Region and wait for it to become active. Once it is active, you then create a DB instance that is the primary instance for that DB cluster.

Replication begins when the primary instance of the read replica DB cluster becomes available.

Use the following procedures to create a cross-Region read replica from an Aurora MySQL DB cluster. These procedures work for creating read replicas from either encrypted or unencrypted DB clusters.

Console

To create an Aurora MySQL DB cluster that is a cross-Region read replica with the AWS Management Console

1. Sign in to the AWS Management Console and open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. In the top-right corner of the AWS Management Console, select the AWS Region that hosts your source DB cluster.
3. In the navigation pane, choose **Instances**.
4. Choose the check box for the DB instance that you want to create a cross-Region read replica for. For **Actions**, choose **Create cross region read replica**.
5. On the **Create cross region read replica** page, choose the option settings for your cross-Region read replica DB cluster, as described in the following table.

Option	Description
Destination region	Choose the AWS Region to host the new cross-Region read replica DB cluster.
Destination DB subnet group	Choose the DB subnet group to use for the cross-Region read replica DB cluster.
Publicly accessible	Choose Yes to give the cross-Region read replica DB cluster a public IP address; otherwise, select No .
Encryption	Select Enable Encryption to turn on encryption at rest for this DB cluster. For more information, see Encrypting Amazon Aurora resources (p. 1709) .
AWS KMS key	Only available if Encryption is set to Enable Encryption . Select the AWS KMS key to use for encrypting this DB cluster. For more information, see Encrypting Amazon Aurora resources (p. 1709) .
DB instance class	Choose a DB instance class that defines the processing and memory requirements for the primary instance in the DB cluster. For more information about DB instance class options, see Aurora DB instance classes (p. 54) .
Multi-AZ deployment	Choose Yes to create a read replica of the new DB cluster in another Availability Zone in the target AWS Region for failover support. For more information about multiple Availability Zones, see Regions and Availability Zones (p. 11) .
Read replica source	Choose the source DB cluster to create a cross-Region read replica for.

Option	Description
DB instance identifier	<p>Type a name for the primary instance in your cross-Region read replica DB cluster. This identifier is used in the endpoint address for the primary instance of the new DB cluster.</p> <p>The DB instance identifier has the following constraints:</p> <ul style="list-style-type: none"> • It must contain from 1 to 63 alphanumeric characters or hyphens. • Its first character must be a letter. • It cannot end with a hyphen or contain two consecutive hyphens. • It must be unique for all DB instances for each AWS account, for each AWS Region. <p>Because the cross-Region read replica DB cluster is created from a snapshot of the source DB cluster, the master user name and master password for the read replica are the same as the master user name and master password for the source DB cluster.</p>
DB cluster identifier	<p>Type a name for your cross-Region read replica DB cluster that is unique for your account in the target AWS Region for your replica. This identifier is used in the cluster endpoint address for your DB cluster. For information on the cluster endpoint, see Amazon Aurora connection management (p. 32).</p> <p>The DB cluster identifier has the following constraints:</p> <ul style="list-style-type: none"> • It must contain from 1 to 63 alphanumeric characters or hyphens. • Its first character must be a letter. • It cannot end with a hyphen or contain two consecutive hyphens. • It must be unique for all DB clusters for each AWS account, for each AWS Region.
Priority	<p>Choose a failover priority for the primary instance of the new DB cluster. This priority determines the order in which Aurora Replicas are promoted when recovering from a primary instance failure. If you don't select a value, the default is tier-1. For more information, see Fault tolerance for an Aurora DB cluster (p. 69).</p>
Database port	<p>Specify the port for applications and utilities to use to access the database. Aurora DB clusters default to the default MySQL port, 3306. Firewalls at some companies block connections to this port. If your company firewall blocks the default port, choose another port for the new DB cluster.</p>

Option	Description
Enhanced monitoring	Choose Enable enhanced monitoring to turn on gathering metrics in real time for the operating system that your DB cluster runs on. For more information, see Monitoring OS metrics with Enhanced Monitoring (p. 626) .
Monitoring Role	Only available if Enhanced Monitoring is set to Enable enhanced monitoring . Choose the IAM role that you created to permit Amazon RDS to communicate with Amazon CloudWatch Logs for you, or choose Default to have RDS create a role for you named <code>rds-monitoring-role</code> . For more information, see Monitoring OS metrics with Enhanced Monitoring (p. 626) .
Granularity	Only available if Enhanced Monitoring is set to Enable enhanced monitoring . Set the interval, in seconds, between when metrics are collected for your DB cluster.
Auto minor version upgrade	This setting doesn't apply to Aurora MySQL DB clusters. For more information about engine updates for Aurora MySQL, see Database engine updates for Amazon Aurora MySQL (p. 1082) .

6. Choose **Create** to create your cross-Region read replica for Aurora.

AWS CLI

To create an Aurora MySQL DB cluster that is a cross-Region read replica with the CLI

1. Call the AWS CLI `create-db-cluster` command in the AWS Region where you want to create the read replica DB cluster. Include the `--replication-source-identifier` option and specify the Amazon Resource Name (ARN) of the source DB cluster to create a read replica for.

For cross-Region replication where the DB cluster identified by `--replication-source-identifier` is encrypted, you must specify the `--kms-key-id` option and the `--storage-encrypted` option. You must also specify either the `--source-region` or `--pre-signed-url` option. Using `--source-region` autogenerated a presigned URL that is a valid request for the `CreateDBCluster` API operation that can be performed in the source AWS Region that contains the encrypted DB cluster to be replicated. Using `--pre-signed-url` requires you to construct a presigned URL manually instead. The KMS key identifier is used to encrypt the read replica. It must be a KMS key valid for the destination AWS Region. To learn more about these options, see [create-db-cluster](#).

Note

You can set up cross-Region replication from an unencrypted DB cluster to an encrypted read replica by specifying `--storage-encrypted` and providing a value for `--kms-key-id`. In this case, you don't need to specify `--source-region` or `--pre-signed-url`.

You can't specify the `--master-username` and `--master-user-password` parameters. Those values are taken from the source DB cluster.

The following code example creates a read replica in the us-east-1 Region from an unencrypted DB cluster snapshot in the us-west-2 Region. The command is called in the us-east-1 Region.

For Linux, macOS, or Unix:

```
aws rds create-db-cluster \
--db-cluster-identifier sample-replica-cluster \
--engine aurora \
--replication-source-identifier arn:aws:rds:us-west-2:123456789012:cluster:sample-
master-cluster
```

For Windows:

```
aws rds create-db-cluster ^
--db-cluster-identifier sample-replica-cluster ^
--engine aurora ^
--replication-source-identifier arn:aws:rds:us-west-2:123456789012:cluster:sample-
master-cluster
```

The following code example creates a read replica in the us-east-1 Region from an encrypted DB cluster snapshot in the us-west-2 Region. The command is called in the us-east-1 Region.

For Linux, macOS, or Unix:

```
aws rds create-db-cluster \
--db-cluster-identifier sample-replica-cluster \
--engine aurora \
--replication-source-identifier arn:aws:rds:us-west-2:123456789012:cluster:sample-
master-cluster \
--kms-key-id my-us-east-1-key \
--source-region us-west-2 \
--storage-encrypted
```

For Windows:

```
aws rds create-db-cluster ^
--db-cluster-identifier sample-replica-cluster ^
--engine aurora ^
--replication-source-identifier arn:aws:rds:us-west-2:123456789012:cluster:sample-
master-cluster ^
--kms-key-id my-us-east-1-key ^
--source-region us-west-2 ^
--storage-encrypted
```

2. Check that the DB cluster has become available to use by using the AWS CLI [describe-db-clusters](#) command, as shown in the following example.

```
aws rds describe-db-clusters --db-cluster-identifier sample-replica-cluster
```

When the [describe-db-clusters](#) results show a status of available, create the primary instance for the DB cluster so that replication can begin. To do so, use the AWS CLI [create-db-instance](#) command as shown in the following example.

For Linux, macOS, or Unix:

```
aws rds create-db-instance \
--db-cluster-identifier sample-replica-cluster \
--db-instance-class db.r3.large \
--db-instance-identifier sample-replica-instance \
--engine aurora
```

For Windows:

```
aws rds create-db-instance ^
--db-cluster-identifier sample-replica-cluster ^
--db-instance-class db.r3.large ^
--db-instance-identifier sample-replica-instance ^
--engine aurora
```

When the DB instance is created and available, replication begins. You can determine if the DB instance is available by calling the AWS CLI [describe-db-instances](#) command.

RDS API

To create an Aurora MySQL DB cluster that is a cross-Region read replica with the API

1. Call the RDS API [CreateDBCluster](#) action in the AWS Region where you want to create the read replica DB cluster. Include the `ReplicationSourceIdentifier` parameter and specify the Amazon Resource Name (ARN) of the source DB cluster to create a read replica for.

For cross-Region replication where the DB cluster identified by `ReplicationSourceIdentifier` is encrypted, you must specify the `KmsKeyId` parameter and set the `StorageEncrypted` parameter to `true`. You must also specify the `PreSignedUrl` parameter. The presigned URL must be a valid request for the `CreateDBCluster` API operation that can be performed in the source AWS Region that contains the encrypted DB cluster to be replicated. The KMS key identifier is used to encrypt the read replica, and must be a KMS key valid for the destination AWS Region. To automatically rather than manually generate a presigned URL, use the AWS CLI [create-db-cluster](#) command with the `--source-region` option instead.

Note

You can set up cross-Region replication from an unencrypted DB cluster to an encrypted read replica by specifying `StorageEncrypted` as `true` and providing a value for `KmsKeyId`. In this case, you don't need to specify `PreSignedUrl`.

You don't need to include the `MasterUsername` and `MasterUserPassword` parameters, because those values are taken from the source DB cluster.

The following code example creates a read replica in the us-east-1 Region from an unencrypted DB cluster snapshot in the us-west-2 Region. The action is called in the us-east-1 Region.

```
https://rds.us-east-1.amazonaws.com/
?Action=CreateDBCluster
&ReplicationSourceIdentifier=arn:aws:rds:us-west-2:123456789012:cluster:sample-
master-cluster
&DBClusterIdentifier=sample-replica-cluster
&Engine=aurora
&SignatureMethod=HmacSHA256
&SignatureVersion=4
&Version=2014-10-31
&X-Amz-Algorithm=AWS4-HMAC-SHA256
&X-Amz-Credential=AKIADQKE4SARGYLE/20161117/us-east-1/rds/aws4_request
&X-Amz-Date=20160201T001547Z
&X-Amz-SignedHeaders=content-type;host;user-agent;x-amz-content-sha256;x-amz-date
&X-Amz-Signature=a04c831a0b54b5e4cd236a90dc9f5fab7185eb3b72b5ebe9a70a4e95790c8b7
```

The following code example creates a read replica in the us-east-1 Region from an encrypted DB cluster snapshot in the us-west-2 Region. The action is called in the us-east-1 Region.

```
https://rds.us-east-1.amazonaws.com/
?Action=CreateDBCluster
&KmsKeyId=my-us-east-1-key
```

```

&StorageEncrypted=true
&PreSignedUrl=https%253A%252F%252Frds.us-west-2.amazonaws.com%252F
    %253FAction%253DCreateDBCluster
    %2526DestinationRegion%253Dus-east-1
    %2526KmsKeyId%253Dmy-us-east-1-key
    %2526ReplicationSourceIdentifier%253Darn%25253Aaws%25253Ards%25253Aus-
west-2%25253A123456789012%25253Acluster%25253Asample-master-cluster
    %2526SignatureMethod%253DHmacSHA256
    %2526SignatureVersion%253D4
    %2526Version%253D2014-10-31
    %2526X-Amz-Algorithm%253DAWS4-HMAC-SHA256
    %2526X-Amz-Credential%253DAKIADQKE4SARGYLE%252F20161117%252Fus-west-2%252Frds
%252Faws4_request
    %2526X-Amz-Date%253D20161117T215409Z
    %2526X-Amz-Expires%253D3600
    %2526X-Amz-SignedHeaders%253Dcontent-type%253Bhost%253Buser-agent%253Bx-amz-
content-sha256%253Bx-amz-date
    %2526X-Amz-Signature
%253D255a0f17b4e717d3b67fad163c3ec26573b882c03a65523522cf890a67fca613
    &ReplicationSourceIdentifier=arn:aws:rds:us-west-2:123456789012:cluster:sample-
master-cluster
    &DBClusterIdentifier=sample-replica-cluster
    &Engine=aurora
    &SignatureMethod=HmacSHA256
    &SignatureVersion=4
    &Version=2014-10-31
    &X-Amz-Algorithm=AWS4-HMAC-SHA256
    &X-Amz-Credential=AKIADQKE4SARGYLE/20161117/us-east-1/rds/aws4_request
    &X-Amz-Date=20160201T001547Z
    &X-Amz-SignedHeaders=content-type;host;user-agent;x-amz-content-sha256;x-amz-date
    &X-Amz-Signature=a04c831a0b54b5e4cd236a90dc9f5fab7185eb3b72b5ebe9a70a4e95790c8b7

```

- Check that the DB cluster has become available to use by using the RDS API [DescribeDBClusters](#) action, as shown in the following example.

```

https://rds.us-east-1.amazonaws.com/
?Action=DescribeDBClusters
&DBClusterIdentifier=sample-replica-cluster
&SignatureMethod=HmacSHA256
&SignatureVersion=4
&Version=2014-10-31
&X-Amz-Algorithm=AWS4-HMAC-SHA256
&X-Amz-Credential=AKIADQKE4SARGYLE/20161117/us-east-1/rds/aws4_request
&X-Amz-Date=20160201T002223Z
&X-Amz-SignedHeaders=content-type;host;user-agent;x-amz-content-sha256;x-amz-date
&X-Amz-Signature=84c2e4f8fba7c577ac5d820711e34c6e45ffcd35be8a6b7c50f329a74f35f426

```

When the [DescribeDBClusters](#) results show a status of `available`, create the primary instance for the DB cluster so that replication can begin. To do so, use the RDS API [CreateDBInstance](#) action as shown in the following example.

```

https://rds.us-east-1.amazonaws.com/
?Action/CreateDBInstance
&DBClusterIdentifier=sample-replica-cluster
&DBInstanceClass=db.r3.large
&DBInstanceIdentifier=sample-replica-instance
&Engine=aurora
&SignatureMethod=HmacSHA256
&SignatureVersion=4
&Version=2014-10-31
&X-Amz-Algorithm=AWS4-HMAC-SHA256
&X-Amz-Credential=AKIADQKE4SARGYLE/20161117/us-east-1/rds/aws4_request
&X-Amz-Date=20160201T003808Z

```

```
&X-Amz-SignedHeaders=content-type;host;user-agent;x-amz-content-sha256;x-amz-date
&X-Amz-Signature=125fe575959f5bbcebd53f2365f907179757a08b5d7a16a378dfa59387f58cdb
```

When the DB instance is created and available, replication begins. You can determine if the DB instance is available by calling the AWS CLI [DescribeDBInstances](#) command.

Viewing Amazon Aurora MySQL cross-Region replicas

You can view the cross-Region replication relationships for your Amazon Aurora MySQL DB clusters by calling the [describe-db-clusters](#) AWS CLI command or the [DescribeDBClusters](#) RDS API operation. In the response, refer to the `ReadReplicaIdentifiers` field for the DB cluster identifiers of any cross-Region read replica DB clusters, and refer to the `ReplicationSourceIdentifier` element for the ARN of the source DB cluster that is the replication source.

Promoting a read replica to be a DB cluster

You can promote an Aurora MySQL read replica to a standalone DB cluster. When you promote an Aurora MySQL read replica, its DB instances are rebooted before they become available.

Typically, you promote an Aurora MySQL read replica to a standalone DB cluster as a data recovery scheme if the source DB cluster fails.

To do this, first create a read replica and then monitor the source DB cluster for failures. In the event of a failure, do the following:

1. Promote the read replica.
2. Direct database traffic to the promoted DB cluster.
3. Create a replacement read replica with the promoted DB cluster as its source.

When you promote a read replica, the read replica becomes a standalone Aurora DB cluster. The promotion process can take several minutes or longer to complete, depending on the size of the read replica. After you promote the read replica to a new DB cluster, it's just like any other DB cluster. For example, you can create read replicas from it and perform point-in-time restore operations. You can also create Aurora Replicas for the DB cluster.

Because the promoted DB cluster is no longer a read replica, you can't use it as a replication target.

The following steps show the general process for promoting a read replica to a DB cluster:

1. Stop any transactions from being written to the read replica source DB cluster, and then wait for all updates to be made to the read replica. Database updates occur on the read replica after they have occurred on the source DB cluster, and this replication lag can vary significantly. Use the `ReplicaLag` metric to determine when all updates have been made to the read replica. The `ReplicaLag` metric records the amount of time a read replica DB instance lags behind the source DB instance. When the `ReplicaLag` metric reaches 0, the read replica has caught up to the source DB instance.
2. Promote the read replica by using the **Promote** option on the Amazon RDS console, the AWS CLI command [promote-read-replica-db-cluster](#), or the [PromoteReadReplicaDBCluster](#) Amazon RDS API operation.

You choose an Aurora MySQL DB instance to promote the read replica. After the read replica is promoted, the Aurora MySQL DB cluster is promoted to a standalone DB cluster. The DB instance with the highest failover priority is promoted to the primary DB instance for the DB cluster. The other DB instances become Aurora Replicas.

Note

The promotion process takes a few minutes to complete. When you promote a read replica, replication is stopped and the DB instances are rebooted. When the reboot is complete, the read replica is available as a new DB cluster.

Console

To promote an Aurora MySQL read replica to a DB cluster

1. Sign in to the AWS Management Console and open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. On the console, choose **Instances**.
The **Instance** pane appears.
3. In the **Instances** pane, choose the read replica that you want to promote.
The read replicas appear as Aurora MySQL DB instances.
4. For **Actions**, choose **Promote read replica**.
5. On the acknowledgment page, choose **Promote read replica**.

AWS CLI

To promote a read replica to a DB cluster, use the AWS CLI `promote-read-replica-db-cluster` command.

Example

For Linux, macOS, or Unix:

```
aws rds promote-read-replica-db-cluster \
--db-cluster-identifier mydbcluster
```

For Windows:

```
aws rds promote-read-replica-db-cluster ^
--db-cluster-identifier mydbcluster
```

RDS API

To promote a read replica to a DB cluster, call `PromoteReadReplicaDBCluster`.

Troubleshooting Amazon Aurora MySQL cross Region replicas

Following you can find a list of common error messages that you might encounter when creating an Amazon Aurora cross-Region read replica, and how to resolve the specified errors.

Source cluster [DB cluster ARN] doesn't have binlogs enabled

To resolve this issue, turn on binary logging on the source DB cluster. For more information, see [Before you begin \(p. 923\)](#).

Source cluster [DB cluster ARN] doesn't have cluster parameter group in sync on writer

You receive this error if you have updated the `binlog_format` DB cluster parameter, but have not rebooted the primary instance for the DB cluster. Reboot the primary instance (that is, the writer) for the DB cluster and try again.

Source cluster [DB cluster ARN] already has a read replica in this region

You can have up to five cross-Region DB clusters that are read replicas for each source DB cluster in any AWS Region. If you already have the maximum number of read replicas for a DB cluster in a particular AWS Region, you must delete an existing one before you can create a new cross-Region DB cluster in that Region.

DB cluster [DB cluster ARN] requires a database engine upgrade for cross-Region replication support

To resolve this issue, upgrade the database engine version for all of the instances in the source DB cluster to the most recent database engine version, and then try creating a cross-Region read replica DB again.

Replication between Aurora and MySQL or between Aurora and another Aurora DB cluster (binary log replication)

Because Amazon Aurora MySQL is compatible with MySQL, you can set up replication between a MySQL database and an Amazon Aurora MySQL DB cluster. This type of replication uses the MySQL binary log replication, also referred to as *binlog replication*. If you use binary log replication with Aurora, we recommend that your MySQL database run MySQL version 5.5 or later. You can set up replication where your Aurora MySQL DB cluster is the replication source or the replica. You can replicate with an Amazon RDS MySQL DB instance, a MySQL database external to Amazon RDS, or another Aurora MySQL DB cluster.

Note

You can't use binlog replication to or from certain kinds of Aurora clusters. In particular, binlog replication isn't available for Aurora Serverless v1 and multi-master clusters. If the `SHOW MASTER STATUS` and `SHOW SLAVE STATUS` (Aurora MySQL version 1 and 2) or `SHOW REPLICA STATUS` (Aurora MySQL version 3) statement returns no output, check that the cluster you're using is one that supports binlog replication.

You can also replicate with an RDS for MySQL DB instance or Aurora MySQL DB cluster in another AWS Region. When you're performing replication across AWS Regions, ensure that your DB clusters and DB instances are publicly accessible. Aurora MySQL DB clusters must be part of a public subnet in your VPC.

If you want to configure replication between an Aurora MySQL DB cluster and an Aurora MySQL DB cluster in another region, you can create an Aurora MySQL DB cluster as a read replica in a different AWS Region than the source DB cluster. For more information, see [Replicating Amazon Aurora MySQL DB clusters across AWS Regions \(p. 922\)](#).

With Aurora MySQL 2.04 and higher, you can replicate between Aurora MySQL and an external source or target that uses global transaction identifiers (GTIDs) for replication. Ensure that the GTID-related parameters in the Aurora MySQL DB cluster have settings that are compatible with the GTID status of the external database. To learn how to do this, see [Using GTID-based replication for Aurora MySQL \(p. 954\)](#). In Aurora MySQL version 3.01 and higher, you can choose how to assign GTIDs to transactions that are replicated from a source that doesn't use GTIDs. For information about the stored procedure that controls that setting, see [mysql.rds_assign_gtids_to_anonymous_transactions \(Aurora MySQL version 3 and higher\) \(p. 1076\)](#).

Warning

When you replicate between Aurora MySQL and MySQL, ensure that you use only InnoDB tables. If you have MyISAM tables that you want to replicate, you can convert them to InnoDB before setting up replication with the following command.

```
alter table <schema>.<table_name> engine=innodb, algorithm=copy;
```

Setting up MySQL replication with Aurora MySQL involves the following steps, which are discussed in detail following in this topic:

1. Turn on binary logging on the replication source ([p. 933](#))
2. Retain binary logs on the replication source until no longer needed ([p. 936](#))
3. Create a snapshot of your replication source ([p. 938](#))
4. Load the snapshot into your replica target ([p. 940](#))
5. Turn on replication on your replica target ([p. 941](#))
6. Monitor your replica ([p. 943](#))

Setting up replication with MySQL or another Aurora DB cluster

To set up Aurora replication with MySQL, take the following steps.

1. Turn on binary logging on the replication source

Find instructions on how to turn on binary logging on the replication source for your database engine following.

Database engine	Instructions
Aurora	<p>To turn on binary logging on an Aurora MySQL DB cluster</p> <p>Set the <code>binlog_format</code> parameter to <code>ROW</code>, <code>STATEMENT</code>, or <code>MIXED</code>. <code>MIXED</code> is recommended unless you have a need for a specific binlog format. The <code>binlog_format</code> parameter is a cluster-level parameter that is in the default cluster parameter group. If you are changing the <code>binlog_format</code> parameter from <code>OFF</code> to another value, then you need to reboot your Aurora DB cluster for the change to take effect.</p> <p>For more information, see Amazon Aurora DB cluster and DB instance parameters (p. 341) and Working with DB parameter groups and DB cluster parameter groups (p. 339).</p>
RDS for MySQL	<p>To turn on binary logging on an Amazon RDS DB instance</p> <p>You can't turn on binary logging directly for an Amazon RDS DB instance, but you can turn it on by doing one of the following:</p> <ul style="list-style-type: none">• Turn on automated backups for the DB instance. You can turn on automated backups when you create a DB instance, or you can turn on backups by modifying an existing DB instance. For more information, see Creating a DB instance in the Amazon RDS User Guide.• Create a read replica for the DB instance. For more information, see Working with read replicas in the Amazon RDS User Guide.
MySQL (external)	<p>To set up encrypted replication</p>

Database engine	Instructions
	<p>To replicate data securely with Aurora MySQL version 5.6, you can use encrypted replication.</p> <p>Currently, encrypted replication with an external MySQL database is only supported for Aurora MySQL version 5.6.</p> <p>Note If you don't need to use encrypted replication, you can skip these steps.</p> <p>The following are prerequisites for using encrypted replication:</p> <ul style="list-style-type: none"> Secure Sockets Layer (SSL) must be enabled on the external MySQL source database. A client key and client certificate must be prepared for the Aurora MySQL DB cluster. <p>During encrypted replication, the Aurora MySQL DB cluster acts a client to the MySQL database server. The certificates and keys for the Aurora MySQL client are in files in .pem format.</p> <ol style="list-style-type: none"> 1. Ensure that you are prepared for encrypted replication: <ul style="list-style-type: none"> If you don't have SSL enabled on the external MySQL source database and don't have a client key and client certificate prepared, turn on SSL on the MySQL database server and generate the required client key and client certificate. If SSL is enabled on the external source, supply a client key and certificate for the Aurora MySQL DB cluster. If you don't have these, generate a new key and certificate for the Aurora MySQL DB cluster. To sign the client certificate, you must have the certificate authority key that you used to configure SSL on the external MySQL source database. <p>For more information, see Creating SSL certificates and keys using openssl in the MySQL documentation.</p> <p>You need the certificate authority certificate, the client key, and the client certificate.</p> <ol style="list-style-type: none"> 2. Connect to the Aurora MySQL DB cluster as the master user using SSL. <p>For information about connecting to an Aurora MySQL DB cluster with SSL, see Using SSL/TLS with Aurora MySQL DB clusters (p. 775).</p> <ol style="list-style-type: none"> 3. Run the <code>mysql.rds_import_binlog_ssl_material</code> stored procedure to import the SSL information into the Aurora MySQL DB cluster. <p>For the <code>ssl_material_value</code> parameter, insert the information from the .pem format files for the Aurora MySQL DB cluster in the correct JSON payload.</p> <p>The following example imports SSL information into an Aurora MySQL DB cluster. In .pem format files, the body code typically is longer than the body code shown in the example.</p> <pre style="border: 1px solid black; padding: 5px;"> call mysql.rds_import_binlog_ssl_material('{"ssl_ca": "-----BEGIN CERTIFICATE-----\nAAAAB3NzaC1yc2EAAAQABAAQClKsfkNkuSevGj3eYhCe53pcjqP3maAhDFcvBS706V\nhz2ItxCih+PnDSUaw+WNQn/mZphTk/a/gU8jEzoOWbkM4yxyb/wB96xbiFveSFJuOp/\nd6RJhJOI0iBXr\n1sLnBITntckij7FbtJMxLvvwJryDUilBMTjYtwB+QhYXUMOzce5Pjz5/i8SeJtjnV3iAoG/\ncQk+0Fzz"}'); </pre>

Database engine	Instructions
	<pre> qaeJAAHco+CY/5WrUBkrHmFJr6HcXkvJdWPkYQS3xqC0+FmUzofz221CBt5IMucxXPkX4rWi +z7wB3Rb BQoQzd8v7yeb7OzlPnWOyN0qFU0XA246RA8QFYiCNYwI3f05p6KLxEXAMPLE -----END CERTIFICATE-----\n", "ssl_cert": "-----BEGIN CERTIFICATE-----\n AAAAB3NzaC1yc2EAAAQABAAQClKsfkNkuSevGj3eYhCe53pcjqP3maAhDFcvBS706V hz2ItxCih+PnDSUaw+WNQn/mZphTk/a/gU8jEzoOWbkM4yxyb/wB96xbiFveSFJuOp/ d6RJhJOI0iBXr lsLnBItnckij7FbtJMxLvvwJryDUilBMTjYtwB+QhYXUMOzce5Pjz5/i8SeJtjnV3iAoG/ cQk+0Fzz qaeJAAHco+CY/5WrUBkrHmFJr6HcXkvJdWPkYQS3xqC0+FmUzofz221CBt5IMucxXPkX4rWi +z7wB3Rb BQoQzd8v7yeb7OzlPnWOyN0qFU0XA246RA8QFYiCNYwI3f05p6KLxEXAMPLE -----END CERTIFICATE-----\n", "ssl_key": "-----BEGIN RSA PRIVATE KEY-----\n AAAAB3NzaC1yc2EAAAQABAAQClKsfkNkuSevGj3eYhCe53pcjqP3maAhDFcvBS706V hz2ItxCih+PnDSUaw+WNQn/mZphTk/a/gU8jEzoOWbkM4yxyb/wB96xbiFveSFJuOp/ d6RJhJOI0iBXr lsLnBItnckij7FbtJMxLvvwJryDUilBMTjYtwB+QhYXUMOzce5Pjz5/i8SeJtjnV3iAoG/ cQk+0Fzz qaeJAAHco+CY/5WrUBkrHmFJr6HcXkvJdWPkYQS3xqC0+FmUzofz221CBt5IMucxXPkX4rWi +z7wB3Rb BQoQzd8v7yeb7OzlPnWOyN0qFU0XA246RA8QFYiCNYwI3f05p6KLxEXAMPLE -----END RSA PRIVATE KEY-----\n"}'); </pre>

For more information, see [mysql_rds_import_binlog_ssl_material](#) and [Using SSL/TLS with Aurora MySQL DB clusters \(p. 775\)](#).

Note

After running the procedure, the secrets are stored in files. To erase the files later, you can run the [mysql_rds_remove_binlog_ssl_material](#) stored procedure.

To turn on binary logging on an external MySQL database

- From a command shell, stop the mysql service.

```
sudo service mysqld stop
```

- Edit the my.cnf file (this file is usually under /etc).

```
sudo vi /etc/my.cnf
```

Add the `log_bin` and `server_id` options to the [mysqld] section. The `log_bin` option provides a file name identifier for binary log files. The `server_id` option provides a unique identifier for the server in source-replica relationships.

If encrypted replication isn't required, ensure that the external MySQL database is started with binlogs enabled and SSL is turned off.

The following are the relevant entries in the /etc/my.cnf file for unencrypted data.

```

log-bin=mysql-bin
server-id=2133421
innodb_flush_log_at_trx_commit=1
sync_binlog=1

```

Database engine	Instructions
	<p>If encrypted replication is required, ensure that the external MySQL database is started with SSL and binlogs enabled.</p> <p>The entries in the <code>/etc/my.cnf</code> file include the .pem file locations for the MySQL database server.</p> <pre>log-bin=mysql-bin server-id=2133421 innodb_flush_log_at_trx_commit=1 sync_binlog=1 # Setup SSL. ssl-ca=/home/sslcerts/ca.pem ssl-cert=/home/sslcerts/server-cert.pem ssl-key=/home/sslcerts/server-key.pem</pre> <p>Additionally, the <code>sql_mode</code> option for your MySQL DB instance must be set to 0, or must not be included in your <code>my.cnf</code> file.</p> <p>While connected to the external MySQL database, record the external MySQL database's binary log position.</p> <pre>mysql> SHOW MASTER STATUS;</pre> <p>Your output should be similar to the following:</p> <pre>+-----+-----+-----+ File Position Binlog_Do_DB Binlog_Ignore_DB Executed_Gtid_Set +-----+-----+-----+ mysql-bin.000031 107 +-----+-----+-----+ 1 row in set (0.00 sec)</pre> <p>For more information, see Setting the replication source configuration in the MySQL documentation.</p> <p>3. Start the mysql service.</p> <pre>sudo service mysqld start</pre>

2. Retain binary logs on the replication source until no longer needed

When you use MySQL binary log replication, Amazon RDS doesn't manage the replication process. As a result, you need to ensure that the binlog files on your replication source are retained until after the changes have been applied to the replica. This maintenance helps ensure that you can restore your source database in the event of a failure.

Find instructions on how to retain binary logs for your database engine following.

Database engine	Instructions
Aurora	<p>To retain binary logs on an Aurora MySQL DB cluster</p> <p>You do not have access to the binlog files for an Aurora MySQL DB cluster. As a result, you must choose a time frame to retain the binlog files on your replication source long enough to ensure that the changes have been applied to your replica before the binlog file is deleted by Amazon RDS. You can retain binlog files on an Aurora MySQL DB cluster for up to 90 days.</p> <p>If you are setting up replication with a MySQL database or RDS for MySQL DB instance as the replica, and the database that you are creating a replica for is very large, choose a large time frame to retain binlog files until the initial copy of the database to the replica is complete and the replica lag has reached 0.</p> <p>To set the binary log retention time frame, use the mysql_rds_set_configuration procedure and specify a configuration parameter of 'binlog retention hours' along with the number of hours to retain binlog files on the DB cluster, up to 2160 (90 days). The following example that sets the retention period for binlog files to 6 days:</p> <pre data-bbox="474 846 1299 874">CALL mysql.rds_set_configuration('binlog retention hours', 144);</pre> <p>After replication has been started, you can verify that changes have been applied to your replica by running the <code>SHOW SLAVE STATUS</code> (Aurora MySQL version 1 and 2) or <code>SHOW REPLICAS STATUS</code> (Aurora MySQL version 3) command on your replica and checking the <code>Seconds behind master</code> field. If the <code>Seconds behind master</code> field is 0, then there is no replica lag. When there is no replica lag, reduce the length of time that binlog files are retained by setting the <code>binlog retention hours</code> configuration parameter to a smaller time frame.</p> <p>If this setting isn't specified, the default for Aurora MySQL is 24 (1 day).</p> <p>If you specify a value for 'binlog retention hours' that is higher than 2160, then Aurora MySQL uses a value of 2160.</p>
RDS for MySQL	<p>To retain binary logs on an Amazon RDS DB instance</p> <p>You can retain binary log files on an Amazon RDS DB instance by setting the binlog retention hours just as with an Aurora MySQL DB cluster, described in the previous section.</p> <p>You can also retain binlog files on an Amazon RDS DB instance by creating a read replica for the DB instance. This read replica is temporary and solely for the purpose of retaining binlog files. After the read replica has been created, call the mysql_rds_stop_replication procedure on the read replica. While replication is stopped, Amazon RDS doesn't delete any of the binlog files on the replication source. After you have set up replication with your permanent replica, you can delete the read replica when the replica lag (<code>Seconds behind master</code> field) between your replication source and your permanent replica reaches 0.</p>
MySQL (external)	<p>To retain binary logs on an external MySQL database</p> <p>Because binlog files on an external MySQL database are not managed by Amazon RDS, they are retained until you delete them.</p> <p>After replication has been started, you can verify that changes have been applied to your replica by running the <code>SHOW SLAVE STATUS</code> (Aurora MySQL version 1 and 2) or <code>SHOW</code></p>

Database engine	Instructions
	REPLICA STATUS (Aurora MySQL version 3) command on your replica and checking the Seconds behind master field. If the Seconds behind master field is 0, then there is no replica lag. When there is no replica lag, you can delete old binlog files.

3. Create a snapshot of your replication source

You use a snapshot of your replication source to load a baseline copy of your data onto your replica and then start replicating from that point on.

Find instructions on how to create a snapshot of your replication source for your database engine following.

Database engine	Instructions
Aurora	<p>To create a snapshot of an Aurora MySQL DB cluster</p> <ol style="list-style-type: none"> 1. Create a DB cluster snapshot of your Amazon Aurora DB cluster. For more information, see Creating a DB cluster snapshot (p. 495). 2. Create a new Aurora DB cluster by restoring from the DB cluster snapshot that you just created. Be sure to retain the same DB parameter group for your restored DB cluster as your original DB cluster. Doing this ensures that the copy of your DB cluster has binary logging enabled. For more information, see Restoring from a DB cluster snapshot (p. 497). 3. In the console, choose Databases and choose the primary instance (writer) for your restored Aurora DB cluster to show its details. Scroll to Recent Events. An event message shows that includes the binlog file name and position. The event message is in the following format. <div style="border: 1px solid black; padding: 5px; margin-top: 10px;"> <p style="margin: 0;">Binlog position from crash recovery is binlog-file-name binlog-position</p> </div> <p>Save the binlog file name and position values for when you start replication.</p> <p>You can also get the binlog file name and position by calling the describe-events command from the AWS CLI. The following shows an example describe-events command with example output.</p> <div style="border: 1px solid black; padding: 5px; margin-top: 10px;"> <pre>PROMPT> aws rds describe-events</pre> </div> <div style="border: 1px solid black; padding: 5px; margin-top: 10px;"> <pre>{ "Events": [{ "EventCategories": [], "SourceType": "db-instance", "SourceArn": "arn:aws:rds:us-west-2:123456789012:db:sample-restored-instance", "Date": "2016-10-28T19:43:46.862Z", "Message": "Binlog position from crash recovery is mysql-bin-changelog.000003 4278", "SourceIdentifier": "sample-restored-instance" }] }</pre> </div>

Database engine	Instructions
	<p>You can also get the binlog file name and position by checking the MySQL error log for the last MySQL binlog file position.</p> <p>4. If your replica target is an Aurora DB cluster owned by another AWS account, an external MySQL database, or an RDS for MySQL DB instance, then you can't load the data from an Amazon Aurora DB cluster snapshot. Instead, create a dump of your Amazon Aurora DB cluster by connecting to your DB cluster using a MySQL client and issuing the <code>mysqldump</code> command. Be sure to run the <code>mysqldump</code> command against the copy of your Amazon Aurora DB cluster that you created. The following is an example.</p> <div style="border: 1px solid black; padding: 5px; margin-top: 10px;"> <pre>PROMPT> mysqldump --databases <database_name> --single-transaction --order-by-primary -r backup.sql -u <local_user> -p</pre> </div> <p>5. When you have finished creating the dump of your data from the newly created Aurora DB cluster, delete that DB cluster as it is no longer needed.</p>
RDS for MySQL	<p>To create a snapshot of an Amazon RDS DB instance</p> <ol style="list-style-type: none"> 1. Create a read replica of your Amazon RDS DB instance. For more information, see Creating a read replica in the <i>Amazon Relational Database Service User Guide</i>. 2. Connect to your read replica and stop replication by running the <code>mysql_rds_stop_replication</code> procedure. 3. While the read replica is Stopped, Connect to the read replica and run the <code>SHOW SLAVE STATUS</code> (Aurora MySQL version 1 and 2) or <code>SHOW REPLICAS STATUS</code> (Aurora MySQL version 3) command. Retrieve the current binary log file name from the <code>Relay_Master_Log_File</code> field and the log file position from the <code>Exec_Master_Log_Pos</code> field. Save these values for when you start replication. 4. While the read replica remains Stopped, create a DB snapshot of the read replica. For more information, see Creating a DB snapshot in the <i>Amazon Relational Database Service User Guide</i>. 5. Delete the read replica.
MySQL (external)	<p>To create a snapshot of an external MySQL database</p> <ol style="list-style-type: none"> 1. Before you create a snapshot, you need to ensure that the binlog location for the snapshot is current with the data in your source instance. To do this, you must first stop any write operations to the instance with the following command: <div style="border: 1px solid black; padding: 5px; margin-top: 10px;"> <pre>mysql> FLUSH TABLES WITH READ LOCK;</pre> </div> 2. Create a dump of your MySQL database using the <code>mysqldump</code> command as shown following: <div style="border: 1px solid black; padding: 5px; margin-top: 10px;"> <pre>PROMPT> sudo mysqldump --databases <database_name> --master-data=2 -- single-transaction \ --order-by-primary -r backup.sql -u <local_user> -p</pre> </div> 3. After you have created the snapshot, unlock the tables in your MySQL database with the following command: <div style="border: 1px solid black; padding: 5px; margin-top: 10px;"> <pre>mysql> UNLOCK TABLES;</pre> </div>

4. Load the snapshot into your replica target

If you plan to load data from a dump of a MySQL database that is external to Amazon RDS, then you might want to create an EC2 instance to copy the dump files to, and then load the data into your DB cluster or DB instance from that EC2 instance. Using this approach, you can compress the dump file(s) before copying them to the EC2 instance in order to reduce the network costs associated with copying data to Amazon RDS. You can also encrypt the dump file or files to secure the data as it is being transferred across the network.

Find instructions on how to load the snapshot of your replication source into your replica target for your database engine following.

Database engine	Instructions
Aurora	<p>To load a snapshot into an Aurora MySQL DB cluster</p> <ul style="list-style-type: none"> • If the snapshot of your replication source is a DB cluster snapshot, then you can restore from the DB cluster snapshot to create a new Aurora MySQL DB cluster as your replica target. For more information, see Restoring from a DB cluster snapshot (p. 497). • If the snapshot of your replication source is a DB snapshot, then you can migrate the data from your DB snapshot into a new Aurora MySQL DB cluster. For more information, see Migrating data to an Amazon Aurora DB cluster (p. 366). • If the snapshot of your replication source is the output from the <code>mysqldump</code> command, then follow these steps: <ol style="list-style-type: none"> 1. Copy the output of the <code>mysqldump</code> command from your replication source to a location that can also connect to your Aurora MySQL DB cluster. 2. Connect to your Aurora MySQL DB cluster using the <code>mysql</code> command. The following is an example: <pre>PROMPT> mysql -h <host_name> -port=3306 -u <db_master_user> -p</pre> <ol style="list-style-type: none"> 3. At the <code>mysql</code> prompt, run the <code>source</code> command and pass it the name of your database dump file to load the data into the Aurora MySQL DB cluster, for example: <pre>mysql> source backup.sql;</pre>
RDS for MySQL	<p>To load a snapshot into an Amazon RDS DB instance</p> <ol style="list-style-type: none"> 1. Copy the output of the <code>mysqldump</code> command from your replication source to a location that can also connect to your MySQL DB instance. 2. Connect to your MySQL DB instance using the <code>mysql</code> command. The following is an example: <pre>PROMPT> mysql -h <host_name> -port=3306 -u <db_master_user> -p</pre> <ol style="list-style-type: none"> 3. At the <code>mysql</code> prompt, run the <code>source</code> command and pass it the name of your database dump file to load the data into the MySQL DB instance, for example: <pre>mysql> source backup.sql;</pre>
MySQL (external)	<p>To load a snapshot into an external MySQL database</p> <p>You cannot load a DB snapshot or a DB cluster snapshot into an external MySQL database. Instead, you must use the output from the <code>mysqldump</code> command.</p>

Database engine	Instructions
	<p>1. Copy the output of the <code>mysqldump</code> command from your replication source to a location that can also connect to your MySQL database.</p> <p>2. Connect to your MySQL database using the <code>mysql</code> command. The following is an example.</p> <pre>PROMPT> mysql -h <host_name> -port=3306 -u <db_master_user> -p</pre> <p>3. At the <code>mysql</code> prompt, run the <code>source</code> command and pass it the name of your database dump file to load the data into your MySQL database. The following is an example.</p> <pre>mysql> source backup.sql;</pre>

5. Turn on replication on your replica target

Before you turn on replication, we recommend that you take a manual snapshot of the Aurora MySQL DB cluster or RDS for MySQL DB instance replica target. If a problem arises and you need to re-establish replication with the DB cluster or DB instance replica target, you can restore the DB cluster or DB instance from this snapshot instead of having to import the data into your replica target again.

Also, create a user ID that is used solely for replication. The following is an example.

```
mysql> CREATE USER 'repl_user'@'<domain_name>' IDENTIFIED BY '<password>';
```

The user requires the REPLICATION CLIENT and REPLICATION SLAVE privileges. Grant these privileges to the user.

```
GRANT REPLICATION CLIENT, REPLICATION SLAVE ON *.* TO 'repl_user'@'<domain_name>';
```

If you need to use encrypted replication, require SSL connections for the replication user. For example, you can use one of the following statement to require SSL connections on the user account `repl_user`.

```
GRANT USAGE ON *.* TO 'repl_user'@'<domain_name>' REQUIRE SSL;
```

Note

If `REQUIRE SSL` isn't included, the replication connection might silently fall back to an unencrypted connection.

Find instructions on how to turn on replication for your database engine following.

Database engine	Instructions
Aurora	<p>To turn on replication from an Aurora MySQL DB cluster</p> <p>1. If your DB cluster replica target was created from a DB cluster snapshot, then connect to the DB cluster replica target and issue the <code>SHOW MASTER STATUS</code> command. Retrieve the current binary log file name from the <code>File</code> field and the log file position from the <code>Position</code> field.</p> <p>If your DB cluster replica target was created from a DB snapshot, then you need the binlog file and binlog position that are the starting place for replication. You retrieved</p>

Database engine	Instructions
	<p>these values from the <code>SHOW SLAVE STATUS</code> (Aurora MySQL version 1 and 2) or <code>SHOW REPLICA STATUS</code> (Aurora MySQL version 3) command when you created the snapshot of your replication source.</p> <p>2. Connect to the DB cluster and issue the mysql_rds_set_external_master (Aurora MySQL version 1 and 2) mysql_rds_set_external_source (Aurora MySQL version 3 and higher) and mysql_rds_start_replication procedures to start replication with your replication source using the binary log file name and location from the previous step. The following is an example.</p> <pre> For Aurora MySQL version 1 and 2: CALL mysql.rds_set_external_master ('mydbinstance.123456789012.us-east-1.rds.amazonaws.com', 3306, 'repl_user', '<password>', 'mysql-bin-changelog.000031', 107, 0); For Aurora MySQL version 3 and higher: CALL mysql.rds_set_external_source ('mydbinstance.123456789012.us-east-1.rds.amazonaws.com', 3306, 'repl_user', '<password>', 'mysql-bin-changelog.000031', 107, 0); For all versions: CALL mysql.rds_start_replication; </pre>
RDS for MySQL	<p>To turn on replication from an Amazon RDS DB instance</p> <p>1. If your DB instance replica target was created from a DB snapshot, then you need the binlog file and binlog position that are the starting place for replication. You retrieved these values from the <code>SHOW SLAVE STATUS</code> (Aurora MySQL version 1 and 2) or <code>SHOW REPLICA STATUS</code> (Aurora MySQL version 3) command when you created the snapshot of your replication source.</p> <p>2. Connect to the DB instance and issue the mysql_rds_set_external_master and mysql_rds_start_replication procedures to start replication with your replication source using the binary log file name and location from the previous step. The following is an example.</p> <pre> For Aurora MySQL version 1 and 2: CALL mysql.rds_set_external_master ('mydbcluster.cluster-123456789012.us-east-1.rds.amazonaws.com', 3306, 'repl_user', '<password>', 'mysql-bin-changelog.000031', 107, 0); For Aurora MySQL version 3 and higher: CALL mysql.rds_set_external_source ('mydbcluster.cluster-123456789012.us-east-1.rds.amazonaws.com', 3306, 'repl_user', '<password>', 'mysql-bin-changelog.000031', 107, 0); For all versions: CALL mysql.rds_start_replication; </pre>

Database engine	Instructions
MySQL (external)	<p>To turn on replication from an external MySQL database</p> <p>1. Retrieve the binlog file and binlog position that are the starting place for replication. You retrieved these values from the <code>SHOW SLAVE STATUS</code> (Aurora MySQL version 1 and 2) or <code>SHOW REPLICAS STATUS</code> (Aurora MySQL version 3) command when you created the snapshot of your replication source. If your external MySQL replica target was populated from the output of the <code>mysqlfdump</code> command with the <code>--master-data=2</code> option, then the binlog file and binlog position are included in the output. The following is an example.</p> <pre style="border: 1px solid black; padding: 5px;"> -- Position to start replication or point-in-time recovery from -- CHANGE MASTER TO MASTER_LOG_FILE='mysql-bin-changelog.000031', MASTER_LOG_POS=107;</pre> <p>2. Connect to the external MySQL replica target, and issue <code>CHANGE MASTER TO</code> and <code>START SLAVE</code> (Aurora MySQL version 1 and 2) or <code>START REPLICA</code> (Aurora MySQL version 3) to start replication with your replication source using the binary log file name and location from the previous step, for example:</p> <pre style="border: 1px solid black; padding: 5px;"> CHANGE MASTER TO MASTER_HOST = 'mydbcluster.cluster-123456789012.us- east-1.rds.amazonaws.com', MASTER_PORT = 3306, MASTER_USER = 'repl_user', MASTER_PASSWORD = '<password>', MASTER_LOG_FILE = 'mysql-bin-changelog.000031', MASTER_LOG_POS = 107; -- And one of these statements depending on your engine version: START SLAVE; -- Aurora MySQL version 1 and 2 START REPLICA; -- Aurora MySQL version 3</pre>

If replication fails, it can result in a large increase in unintentional I/O on the replica, which can degrade performance. If replication fails or is no longer needed, you can run the `mysql.rds_reset_external_master` stored procedure to remove the replication configuration.

6. Monitor your replica

When you set up MySQL replication with an Aurora MySQL DB cluster, you must monitor failover events for the Aurora MySQL DB cluster when it is the replica target. If a failover occurs, then the DB cluster that is your replica target might be recreated on a new host with a different network address. For information on how to monitor failover events, see [Using Amazon RDS event notification \(p. 675\)](#).

You can also monitor how far the replica target is behind the replication source by connecting to the replica target and running the `SHOW SLAVE STATUS` (Aurora MySQL version 1 and 2) or `SHOW REPLICAS STATUS` (Aurora MySQL version 3) command. In the command output, the `Seconds Behind Master` field tells you how far the replica target is behind the source.

Stopping replication between Aurora and MySQL or between Aurora and another Aurora DB cluster

To stop binary log replication with a MySQL DB instance, external MySQL database, or another Aurora DB cluster, follow these steps, discussed in detail following in this topic.

1. Stop binary log replication on the replica target (p. 944)

2. Turn off binary logging on the replication source (p. 944)

1. Stop binary log replication on the replica target

Find instructions on how to stop binary log replication for your database engine following.

Database engine	Instructions
Aurora	To stop binary log replication on an Aurora MySQL DB cluster replica target Connect to the Aurora DB cluster that is the replica target, and call the mysql_rds_stop_replication procedure.
RDS for MySQL	To stop binary log replication on an Amazon RDS DB instance Connect to the RDS DB instance that is the replica target and call the mysql_rds_stop_replication procedure. The <code>mysql.rds_stop_replication</code> procedure is only available for MySQL versions 5.5 and later, 5.6 and later, and 5.7 and later.
MySQL (external)	To stop binary log replication on an external MySQL database Connect to the MySQL database and call the <code>STOP REPLICATION</code> command.

2. Turn off binary logging on the replication source

Find instructions on how to turn off binary logging on the replication source for your database engine following.

Database engine	Instructions
Aurora	To turn off binary logging on an Amazon Aurora DB cluster 1. Connect to the Aurora DB cluster that is the replication source, and set the binary log retention time frame to 0. To set the binary log retention time frame, use the mysql_rds_set_configuration procedure and specify a configuration parameter of 'binlog_retention_hours' along with the number of hours to retain binlog files on the DB cluster, in this case 0, as shown in the following example. <pre>CALL mysql.rds_set_configuration('binlog_retention_hours', 0);</pre> 2. Set the <code>binlog_format</code> parameter to OFF on the replication source. The <code>binlog_format</code> parameter is a cluster-level parameter that is in the default cluster parameter group. After you have changed the <code>binlog_format</code> parameter value, reboot your DB cluster for the change to take effect.

Database engine	Instructions
	For more information, see Amazon Aurora DB cluster and DB instance parameters (p. 341) and Modifying parameters in a DB parameter group (p. 347) .
RDS for MySQL	<p>To turn off binary logging on an Amazon RDS DB instance</p> <p>You can't turn off binary logging directly for an Amazon RDS DB instance, but you can turn it off by doing the following:</p> <ol style="list-style-type: none"> 1. Turn off automated backups for the DB instance. You can turn off automated backups by modifying an existing DB instance and setting the Backup Retention Period to 0. For more information, see Modifying an Amazon RDS DB instance and Working with backups in the <i>Amazon Relational Database Service User Guide</i>. 2. Delete all read replicas for the DB instance. For more information, see Working with read replicas of MariaDB, MySQL, and PostgreSQL DB instances in the <i>Amazon Relational Database Service User Guide</i>.
MySQL (external)	<p>To turn off binary logging on an external MySQL database</p> <p>Connect to the MySQL database and call the <code>STOP REPLICATION</code> command.</p> <ol style="list-style-type: none"> 1. From a command shell, stop the <code>mysqld</code> service, <div style="border: 1px solid black; padding: 5px; margin-top: 10px;"> <pre>sudo service mysqld stop</pre> </div> 2. Edit the <code>my.cnf</code> file (this file is usually under <code>/etc</code>). <div style="border: 1px solid black; padding: 5px; margin-top: 10px;"> <pre>sudo vi /etc/my.cnf</pre> </div> <p>Delete the <code>log_bin</code> and <code>server_id</code> options from the <code>[mysqld]</code> section.</p> <p>For more information, see Setting the replication source configuration in the MySQL documentation.</p> 3. Start the <code>mysql</code> service. <div style="border: 1px solid black; padding: 5px; margin-top: 10px;"> <pre>sudo service mysqld start</pre> </div>

Using Amazon Aurora to scale reads for your MySQL database

You can use Amazon Aurora with your MySQL DB instance to take advantage of the read scaling capabilities of Amazon Aurora and expand the read workload for your MySQL DB instance. To use Aurora to read scale your MySQL DB instance, create an Amazon Aurora MySQL DB cluster and make it a read replica of your MySQL DB instance. This applies to an RDS for MySQL DB instance, or a MySQL database running external to Amazon RDS.

For information on creating an Amazon Aurora DB cluster, see [Creating an Amazon Aurora DB cluster \(p. 125\)](#).

When you set up replication between your MySQL DB instance and your Amazon Aurora DB cluster, be sure to follow these guidelines:

- Use the Amazon Aurora DB cluster endpoint address when you reference your Amazon Aurora MySQL DB cluster. If a failover occurs, then the Aurora Replica that is promoted to the primary instance for the Aurora MySQL DB cluster continues to use the DB cluster endpoint address.
- Maintain the binlogs on your writer instance until you have verified that they have been applied to the Aurora Replica. This maintenance ensures that you can restore your writer instance in the event of a failure.

Important

When using self-managed replication, you're responsible for monitoring and resolving any replication issues that may occur. For more information, see [Diagnosing and resolving lag between read replicas \(p. 1818\)](#).

Note

The permissions required to start replication on an Amazon Aurora MySQL DB cluster are restricted and not available to your Amazon RDS master user. Because of this, you must use the Amazon RDS [mysql_rds_set_external_master](#) and [mysql_rds_start_replication](#) procedures to set up replication between your Amazon Aurora MySQL DB cluster and your MySQL DB instance.

Start replication between an external source instance and a MySQL DB instance on Amazon RDS

1. Make the source MySQL DB instance read-only:

```
mysql> FLUSH TABLES WITH READ LOCK;
mysql> SET GLOBAL read_only = ON;
```

2. Run the SHOW MASTER STATUS command on the source MySQL DB instance to determine the binlog location. You receive output similar to the following example:

File	Position
mysql-bin-changelog.000031	107

3. Copy the database from the external MySQL DB instance to the Amazon Aurora MySQL DB cluster using mysqldump. For very large databases, you might want to use the procedure in [Importing data to a MySQL or MariaDB DB instance with reduced downtime](#) in the *Amazon Relational Database Service User Guide*.

For Linux, macOS, or Unix:

```
mysqldump \
--databases <database_name> \
--single-transaction \
--compress \
--order-by-primary \
-u <local_user> \
-p <local_password> | mysql \
--host aurora_cluster_endpoint_address \
--port 3306 \
-u <RDS_user_name> \
-p <RDS_password>
```

For Windows:

```
mysqldump ^
--databases <database_name> ^
--single-transaction ^
```

```
--compress ^
--order-by-primary ^
-u <local_user> ^
-p <local_password> | mysql ^
--host aurora_cluster_endpoint_address ^
--port 3306 ^
-u <RDS_user_name> ^
-p <RDS_password>
```

Note

Make sure that there is not a space between the `-p` option and the entered password.

Use the `--host`, `--user` (`-u`), `--port` and `-p` options in the `mysql` command to specify the hostname, user name, port, and password to connect to your Aurora DB cluster. The host name is the DNS name from the Amazon Aurora DB cluster endpoint, for example, `mydbcluster.cluster-123456789012.us-east-1.rds.amazonaws.com`. You can find the endpoint value in the cluster details in the Amazon RDS Management Console.

4. Make the source MySQL DB instance writeable again:

```
mysql> SET GLOBAL read_only = OFF;
mysql> UNLOCK TABLES;
```

For more information on making backups for use with replication, see [Backing up a source or replica by making it read only](#) in the MySQL documentation.

5. In the Amazon RDS Management Console, add the IP address of the server that hosts the source MySQL database to the VPC security group for the Amazon Aurora DB cluster. For more information on modifying a VPC security group, see [Security groups for your VPC](#) in the *Amazon Virtual Private Cloud User Guide*.

You might also need to configure your local network to permit connections from the IP address of your Amazon Aurora DB cluster, so that it can communicate with your source MySQL instance. To find the IP address of the Amazon Aurora DB cluster, use the `host` command.

```
host <aurora_endpoint_address>
```

The host name is the DNS name from the Amazon Aurora DB cluster endpoint.

6. Using the client of your choice, connect to the external MySQL instance and create a MySQL user to be used for replication. This account is used solely for replication and must be restricted to your domain to improve security. The following is an example.

```
CREATE USER 'repl_user'@'mydomain.com' IDENTIFIED BY '<password>';
```

7. For the external MySQL instance, grant REPLICATION CLIENT and REPLICATION SLAVE privileges to your replication user. For example, to grant the REPLICATION CLIENT and REPLICATION SLAVE privileges on all databases for the 'repl_user' user for your domain, issue the following command.

```
GRANT REPLICATION CLIENT, REPLICATION SLAVE ON *.* TO 'repl_user'@'mydomain.com'
IDENTIFIED BY '<password>';
```

8. Take a manual snapshot of the Aurora MySQL DB cluster to be the read replica before setting up replication. If you need to reestablish replication with the DB cluster as a read replica, you can restore the Aurora MySQL DB cluster from this snapshot instead of having to import the data from your MySQL DB instance into a new Aurora MySQL DB cluster.
9. Make the Amazon Aurora DB cluster the replica. Connect to the Amazon Aurora DB cluster as the master user and identify the source MySQL database as the replication master by using the

`mysql_rds_set_external_master` procedure. Use the master log file name and master log position that you determined in Step 2. The following is an example.

```
For Aurora MySQL version 1 and 2:  
CALL mysql.rds_set_external_master ('mymasterserver.mydomain.com', 3306,  
    'repl_user', '<password>', 'mysql-bin-changelog.000031', 107, 0);  
  
For Aurora MySQL version 3 and higher:  
CALL mysql.rds_set_external_source ('mymasterserver.mydomain.com', 3306,  
    'repl_user', '<password>', 'mysql-bin-changelog.000031', 107, 0);
```

10On the Amazon Aurora DB cluster, issue the `mysql_rds_start_replication` procedure to start replication.

```
CALL mysql.rds_start_replication;
```

After you have established replication between your source MySQL DB instance and your Amazon Aurora DB cluster, you can add Aurora Replicas to your Amazon Aurora DB cluster. You can then connect to the Aurora Replicas to read scale your data. For information on creating an Aurora Replica, see [Adding Aurora Replicas to a DB cluster \(p. 392\)](#).

Optimizing binary log replication

Following, you can learn how to optimize binary log replication performance and troubleshoot related issues in Aurora MySQL.

Tip

This discussion presumes that you are familiar with the MySQL binary log replication mechanism and how it works. For background information, see [Replication Implementation](#) in the MySQL documentation.

Multithreaded binary log replication (Aurora MySQL version 3 and higher)

With multithreaded binary log replication, a SQL thread reads events from the relay log and queues them up for SQL worker threads to apply. The SQL worker threads are managed by a coordinator thread. The binary log events are applied in parallel when possible.

When an Aurora MySQL instance is configured to use binary log replication, by default the replica instance uses single-threaded replication. To enable multithreaded replication, you update the `replica_parallel_workers` parameter to a value greater than zero in your custom parameter group.

The following configuration options help you to fine-tune multithreaded replication. For usage information, see [Replication and Binary Logging Options and Variables](#) in the *MySQL Reference Manual*.

Optimal configuration depends on several factors. For example, performance for binary log replication is influenced by your database workload characteristics and the DB instance class the replica is running on. Thus, we recommend that you thoroughly test all changes to these configuration parameters before applying new parameter settings to a production instance.

- `replica_parallel_workers`
- `replica_parallel_type`
- `replica_preserve_commit_order`
- `binlog_transaction_dependency_tracking`
- `binlog_transaction_dependency_history_size`
- `binlog_group_commit_sync_delay`
- `binlog_group_commit_sync_no_delay_count`

Optimizing binlog replication (Aurora MySQL 2.10 and higher)

In Aurora MySQL 2.10 and higher, Aurora automatically applies an optimization known as the binlog I/O cache to binary log replication. By caching the most recently committed binlog events, this optimization is designed to improve binlog dump thread performance while limiting the impact to foreground transactions on the binlog source instance.

Note

This memory used for this feature is independent of the MySQL `binlog_cache_size` setting. This feature doesn't apply to Aurora DB instances that use the `db.t2` and `db.t3` instance classes.

You don't need to adjust any configuration parameters to turn on this optimization. In particular, if you adjust the configuration parameter `aurora_binlog_replication_max_yield_seconds` to a nonzero value in earlier Aurora MySQL versions, set it back to zero for Aurora MySQL 2.10 and higher.

The status variables `aurora_binlog_io_cache_reads` and `aurora_binlog_io_cache_read_requests` are available in Aurora MySQL 2.10 and higher. These status variables help you to monitor how often the data is read from the binlog I/O cache.

- `aurora_binlog_io_cache_read_requests` shows the number of binlog I/O read requests from the cache.
- `aurora_binlog_io_cache_reads` shows the number of binlog I/O reads that retrieve information from the cache.

The following SQL query computes the percentage of binlog read requests that take advantage of the cached information. In this case, the closer the ratio is to 100, the better it is.

```
mysql> SELECT
    (SELECT VARIABLE_VALUE FROM INFORMATION_SCHEMA.GLOBAL_STATUS
     WHERE VARIABLE_NAME='aurora_binlog_io_cache_reads')
   / (SELECT VARIABLE_VALUE FROM INFORMATION_SCHEMA.GLOBAL_STATUS
     WHERE VARIABLE_NAME='aurora_binlog_io_cache_read_requests')
   * 100
   as binlog_io_cache_hit_ratio;
+-----+
| binlog_io_cache_hit_ratio |
+-----+
|      99.99847949080622 |
+-----+
```

The binlog I/O cache feature also includes new metrics related to the binlog dump threads. *Dump threads* are the threads that are created when new binlog replicas are connected to the binlog source instance.

The dump thread metrics are printed to the database log every 60 seconds with the prefix [`Dump thread metrics`]. The metrics include information for each binlog replica such as `Secondary_id`, `Secondary_uuid`, binlog file name, and the position that each replica is reading. The metrics also include `Bytes_behind_primary` representing the distance in bytes between replication source and replica. This metric measures the lag of the replica I/O thread. That figure is different from the lag of the replica SQL applier thread, which is represented by the `seconds_behind_master` metric on the binlog replica. You can determine whether binlog replicas are catching up to the source or falling behind by checking whether the distance decreases or increases.

Optimizing binlog replication (Aurora MySQL 2.04.5 through 2.09)

To optimize binary log replication for Aurora MySQL, you adjust the following cluster-level optimization parameters. These parameters help you to specify the right balance between latency on the binlog source instance and replication lag.

- `aurora_binlog_use_large_read_buffer`
- `aurora_binlog_read_buffer_size`
- `aurora_binlog_replication_max_yield_seconds`

Note

For MySQL 5.7-compatible clusters, you can use these parameters in Aurora MySQL version 2.04.5 through 2.09.*. In Aurora MySQL 2.10.0 and higher, these parameters are superseded by the binlog I/O cache optimization and you don't need to use them.

For MySQL 5.6-compatible clusters, you can use these parameters in Aurora MySQL version 1.17.6 and later.

Topics

- [Overview of the large read buffer and max-yield optimizations \(p. 950\)](#)
- [Related parameters \(p. 951\)](#)
- [Enabling the max-yield mechanism for binary log replication \(p. 952\)](#)
- [Turning off the binary log replication max-yield optimization \(p. 953\)](#)
- [Turning off the large read buffer \(p. 953\)](#)

Overview of the large read buffer and max-yield optimizations

You might experience reduced binary log replication performance when the binary log dump thread accesses the Aurora cluster volume while the cluster processes a high number of transactions. You can use the parameters `aurora_binlog_use_large_read_buffer`, `aurora_binlog_replication_max_yield_seconds`, and `aurora_binlog_read_buffer_size` to help minimize this type of contention.

Suppose that you have a situation where `aurora_binlog_replication_max_yield_seconds` is set to greater than 0 and the current binlog file of the dump thread is active. In this case, the binary log dump thread waits up to a specified number of seconds for the current binlog file to be filled by transactions. This wait period avoids contention that can arise from replicating each binlog event individually. However, doing so increases the replica lag for binary log replicas. Those replicas can fall behind the source by the same number of seconds as the `aurora_binlog_replication_max_yield_seconds` setting.

The current binlog file means the binlog file that the dump thread is currently reading to perform replication. We consider that a binlog file is active when the binlog file is updating or open to be updated by incoming transactions. After Aurora MySQL fills up the active binlog file, MySQL creates and switches to a new binlog file. The old binlog file becomes inactive. It isn't updated by incoming transactions any longer.

Note

Before adjusting these parameters, measure your transaction latency and throughput over time. You might find that binary log replication performance is stable and has low latency even if there is occasional contention.

`aurora_binlog_use_large_read_buffer`

If this parameter is set to 1, Aurora MySQL optimizes binary log replication based on the settings of the parameters `aurora_binlog_read_buffer_size` and `aurora_binlog_replication_max_yield_seconds`. If `aurora_binlog_use_large_read_buffer` is 0, Aurora MySQL ignores the values of the `aurora_binlog_read_buffer_size` and `aurora_binlog_replication_max_yield_seconds` parameters.

`aurora_binlog_read_buffer_size`

Binary log dump threads with larger read buffer minimize the number of read I/O operations by reading more events for each I/O. The parameter `aurora_binlog_read_buffer_size` sets the read buffer size. The large read buffer can reduce binary log contention for workloads that generate a large amount of binlog data.

Note

This parameter only has an effect when the cluster also has the setting `aurora_binlog_use_large_read_buffer=1`.

Increasing the size of the read buffer doesn't affect the performance of binary log replication. Binary log dump threads don't wait for updating transactions to fill up the read buffer.

`aurora_binlog_replication_max_yield_seconds`

If your workload requires low transaction latency, and you can tolerate some replication lag, you can increase the `aurora_binlog_replication_max_yield_seconds` parameter. This parameter controls the maximum yield property of binary log replication in your cluster.

Note

This parameter only has an effect when the cluster also has the setting `aurora_binlog_use_large_read_buffer=1`.

Aurora MySQL recognizes any change to the `aurora_binlog_replication_max_yield_seconds` parameter value immediately. You don't need to restart the DB instance. However, when you turn on this setting, the dump thread only starts to yield when the current binlog file reaches its maximum size of 128 MB and is rotated to a new file.

Related parameters

Use the following DB cluster parameters to turn on the binlog optimization.

Binlog optimization parameters for Aurora MySQL version 2.04.5 and later

Parameter	Default	Valid Values	Description
<code>aurora_binlog_use_large_read_buffer</code>		0, 1	Switch for turning on the feature of replication improvement. When it is 1, the binary log dump thread uses <code>aurora_binlog_read_buffer_size</code> for binary log replication; otherwise default buffer size (8K) is used.
<code>aurora_binlog_read_buffer_size</code>	51242880	8192-536870912	Read buffer size used by binary log dump thread when the parameter <code>aurora_binlog_use_large_read_buffer</code> is set to 1.
<code>aurora_binlog_replication_max_yield_seconds</code>	0.36000	0.36000	For Aurora MySQL versions 2.04.5–2.04.8 and 2.05–2.08.*, the maximum accepted value is 45. You can tune it to a higher value.

Parameter	Default	Valid Values	Description
			value on 2.04.9 and later versions of 2.04.* , and on 2.09 and later versions. This parameter works only when the parameter <code>aurora_binlog_use_large_read_buffer</code> is set to 1.

Binlog optimization parameters for Aurora MySQL version 1.17.6 and later

Parameter	Default	Valid Values	Description
<code>aurora_binlog_use_large_read_buffer</code>		0, 1	Switch for turning on the feature of replication improvement. When it is 1, the binary log dump thread uses <code>aurora_binlog_read_buffer_size</code> for binary log replication. Otherwise, the default buffer size (8 KB) is used.
<code>aurora_binlog_read_buffer_size</code>	5142280	8192-536870912	Read buffer size used by binary log dump thread when the parameter <code>aurora_binlog_use_large_read_buffer</code> is set to 1.
<code>aurora_binlog_replication_max_yield_seconds</code>	0.16000		Maximum seconds to yield when the binary log dump thread replicates the current binlog file (the file used by foreground queries) to replicas. This parameter works only when the parameter <code>aurora_binlog_use_large_read_buffer</code> is set to 1.

Enabling the max-yield mechanism for binary log replication

You can turn on the binary log replication max-yield optimization as follows. Doing so minimizes latency for transactions on the binlog source instance. However, you might experience higher replication lag.

To turn on the max-yield binlog optimization for an Aurora MySQL cluster

1. Create or edit a DB cluster parameter group using the following parameter settings:
 - `aurora_binlog_use_large_read_buffer`: turn on with a value of ON or 1.
 - `aurora_binlog_replication_max_yield_seconds`: specify a value greater than 0.

2. Associate the DB cluster parameter group with the Aurora MySQL cluster that works as the binlog source. To do so, follow the procedures in [Working with DB parameter groups and DB cluster parameter groups \(p. 339\)](#).
3. Confirm that the parameter change takes effect. To do so, run the following query on the binlog source instance.

```
SELECT @@aurora_binlog_use_large_read_buffer,  
      @@aurora_binlog_replication_max_yield_seconds;
```

Your output should be similar to the following.

```
+-----+  
+-----+  
| @@aurora_binlog_use_large_read_buffer | @@aurora_binlog_replication_max_yield_seconds  
|  
+-----+  
+-----+  
| 1 | 45  
|  
+-----+  
+-----+
```

Turning off the binary log replication max-yield optimization

You can turn off the binary log replication max-yield optimization as follows. Doing so minimizes replication lag. However, you might experience higher latency for transactions on the binlog source instance.

To turn off the max-yield optimization for an Aurora MySQL cluster

1. Make sure that the DB cluster parameter group associated with the Aurora MySQL cluster has `aurora_binlog_replication_max_yield_seconds` set to 0. For more information about setting configuration parameters using parameter groups, see [Working with DB parameter groups and DB cluster parameter groups \(p. 339\)](#).
2. Confirm that the parameter change takes effect. To do so, run the following query on the binlog source instance.

```
SELECT @@aurora_binlog_replication_max_yield_seconds;
```

Your output should be similar to the following.

```
+-----+  
| @@aurora_binlog_replication_max_yield_seconds |  
+-----+  
| 0 |  
+-----+
```

Turning off the large read buffer

You can turn off the entire large read buffer feature as follows.

To turn off the large binary log read buffer for an Aurora MySQL cluster

1. Reset the `aurora_binlog_use_large_read_buffer` to OFF or 0.

Make sure that the DB cluster parameter group associated with the Aurora MySQL cluster has `aurora_binlog_use_large_read_buffer` set to 0. For more information about setting configuration parameters using parameter groups, see [Working with DB parameter groups and DB cluster parameter groups \(p. 339\)](#).

2. On the binlog source instance, run the following query.

```
SELECT @@ aurora_binlog_use_large_read_buffer;
```

Your output should be similar to the following.

```
+-----+  
| @@aurora_binlog_use_large_read_buffer |  
+-----+  
| 0 |  
+-----+
```

Synchronizing passwords between replication source and target

When you change user accounts and passwords on the replication source using SQL statements, those changes are replicated to the replication target automatically.

If you use the AWS Management Console, the AWS CLI, or the RDS API to change the master password on the replication source, those changes are not automatically replicated to the replication target. If you want to synchronize the master user and master password between the source and target systems, you must make the same change on the replication target yourself.

Using GTID-based replication for Aurora MySQL

Following, you can learn how to use global transaction identifiers (GTIDs) with binary log (binlog) replication between an Aurora MySQL cluster and an external source.

Note

For Aurora, you can only use this feature with Aurora MySQL clusters that use binlog replication to or from an external MySQL database. The other database might be an Amazon RDS MySQL instance, an on-premises MySQL database, or an Aurora DB cluster in a different AWS Region. To learn how to configure that kind of replication, see [Replication between Aurora and MySQL or between Aurora and another Aurora DB cluster \(binary log replication\) \(p. 932\)](#).

If you use binlog replication and aren't familiar with GTID-based replication with MySQL, see [Replication with global transaction identifiers](#) in the MySQL documentation for background.

GTID-based replication is supported for MySQL 5.7-compatible clusters in Aurora MySQL version 2.04 and higher. GTID-based replication isn't supported for MySQL 5.6-compatible clusters in Aurora MySQL version 1.

Topics

- [Overview of global transaction identifiers \(GTIDs\) \(p. 955\)](#)
- [Parameters for GTID-based replication \(p. 955\)](#)
- [Configuring GTID-based replication for an Aurora MySQL cluster \(p. 956\)](#)
- [Disabling GTID-based replication for an Aurora MySQL DB cluster \(p. 957\)](#)

Overview of global transaction identifiers (GTIDs)

Global transaction identifiers (GTIDs) are unique identifiers generated for committed MySQL transactions. You can use GTIDs to make binlog replication simpler and easier to troubleshoot.

Note

When Aurora synchronizes data among the DB instances in a cluster, that replication mechanism doesn't involve the binary log (binlog). For Aurora MySQL, GTID-based replication only applies when you also use binlog replication to replicate into or out of an Aurora MySQL DB cluster from an external MySQL-compatible database.

MySQL uses two different types of transactions for binlog replication:

- *GTID transactions* – Transactions that are identified by a GTID.
- *Anonymous transactions* – Transactions that don't have a GTID assigned.

In a replication configuration, GTIDs are unique across all DB instances. GTIDs simplify replication configuration because when you use them, you don't have to refer to log file positions. GTIDs also make it easier to track replicated transactions and determine whether the source instance and replicas are consistent.

You typically use GTID-based replication with Aurora when replicating from an external MySQL-compatible database into an Aurora cluster. You can set up this replication configuration as part of a migration from an on-premises or Amazon RDS database into Aurora MySQL. If the external database already uses GTIDs, enabling GTID-based replication for the Aurora cluster simplifies the replication process.

You configure GTID-based replication for an Aurora MySQL cluster by first setting the relevant configuration parameters in a DB cluster parameter group. You then associate that parameter group with the cluster.

Parameters for GTID-based replication

Use the following parameters to configure GTID-based replication.

Parameter	Valid values	Description
gtid_mode	OFF, OFF_PERMISSIVE, ON_PERMISSIVE, ON	OFF specifies that new transactions are anonymous transactions (that is, don't have GTIDs), and a transaction must be anonymous to be replicated. OFF_PERMISSIVE specifies that new transactions are anonymous transactions, but all transactions can be replicated. ON_PERMISSIVE specifies that new transactions are GTID transactions, but all transactions can be replicated. ON specifies that new transactions are GTID transactions, and a transaction must be a GTID transaction to be replicated.
enforce_gtid_consistency	OFF, ON	OFF allows transactions to violate GTID consistency. ON prevents transactions from violating GTID consistency.

Parameter	Valid values	Description
		WARN allows transactions to violate GTID consistency but generates a warning when a violation occurs.

Note

In the AWS Management Console, the `gtid_mode` parameter appears as `gtid-mode`.

For GTID-based replication, use these settings for the DB cluster parameter group for your Aurora MySQL DB cluster:

- `ON` and `ON_PERMISSIVE` apply only to outgoing replication from an RDS DB instance or Aurora MySQL cluster. Both of these values cause your RDS DB instance or Aurora DB cluster to use GTIDs for transactions that are replicated to an external database. `ON` requires that the external database also use GTID-based replication. `ON_PERMISSIVE` makes GTID-based replication optional on the external database.
- `OFF_PERMISSIVE`, if set, means that your RDS DB instances or Aurora DB cluster can accept incoming replication from an external database. It can do this whether the external database uses GTID-based replication or not.
- `OFF`, if set, means that your RDS DB instances or Aurora DB cluster only accept incoming replication from external databases that don't use GTID-based replication.

Tip

Incoming replication is the most common binlog replication scenario for Aurora MySQL clusters. For incoming replication, we recommend that you set the GTID mode to `OFF_PERMISSIVE`. That setting allows incoming replication from external databases regardless of the GTID settings at the replication source.

For more information about parameter groups, see [Working with DB parameter groups and DB cluster parameter groups \(p. 339\)](#).

Configuring GTID-based replication for an Aurora MySQL cluster

When GTID-based replication is enabled for an Aurora MySQL DB cluster, the GTID settings apply to both inbound and outbound binlog replication.

To enable GTID-based replication for an Aurora MySQL cluster

1. Create or edit a DB cluster parameter group using the following parameter settings:
 - `gtid_mode` – `ON` or `ON_PERMISSIVE`
 - `enforce_gtid_consistency` – `ON`
2. Associate the DB cluster parameter group with the Aurora MySQL cluster. To do so, follow the procedures in [Working with DB parameter groups and DB cluster parameter groups \(p. 339\)](#).
3. In Aurora MySQL version 3 and higher, optionally specify how to assign GTIDs to transactions that don't include them. To do so, call the stored procedure in [mysql.rds_assign_gtids_to_anonymous_transactions \(Aurora MySQL version 3 and higher\) \(p. 1076\)](#).

Disabling GTID-based replication for an Aurora MySQL DB cluster

You can disable GTID-based replication for an Aurora MySQL DB cluster. Doing so means that the Aurora cluster can't perform inbound or outbound binlog replication with external databases that use GTID-based replication.

Note

In the following procedure, *read replica* means the replication target in an Aurora configuration with binlog replication to or from an external database. It doesn't mean the read-only Aurora Replica DB instances. For example, when an Aurora cluster accepts incoming replication from an external source, the Aurora primary instance acts as the read replica for binlog replication.

For more details about the stored procedures mentioned in this section, see [Aurora MySQL stored procedures \(p. 1076\)](#).

To disable GTID-based replication for an Aurora MySQL DB cluster

1. On the Aurora primary instance, run the following procedure.

```
CALL mysql.rds_set_master_auto_position(0); (Aurora MySQL version 1 and 2)
CALL mysql.rds_set_source_auto_position(0); (Aurora MySQL version 3 and higher)
```

2. Reset the `gtid_mode` to `ON_PERMISSIVE`.

- a. Make sure that the DB cluster parameter group associated with the Aurora MySQL cluster has `gtid_mode` set to `ON_PERMISSIVE`.

For more information about setting configuration parameters using parameter groups, see [Working with DB parameter groups and DB cluster parameter groups \(p. 339\)](#).

- b. Restart the Aurora MySQL DB cluster.

3. Reset the `gtid_mode` to `OFF_PERMISSIVE`:

- a. Make sure that the DB cluster parameter group associated with the Aurora MySQL cluster has `gtid_mode` set to `OFF_PERMISSIVE`.
- b. Restart the Aurora MySQL DB cluster.

4. a. On the Aurora primary instance, run the `SHOW MASTER STATUS` command.

Your output should be similar to the following.

File	Position
mysql-bin-changelog.000031	107

Note the file and position in your output.

- b. On each read replica, use the file and position information from its source instance in the previous step to run the following query.

```
SELECT MASTER_POS_WAIT('file', position);
```

For example, if the file name is `mysql-bin-changelog.000031` and the position is `107`, run the following statement.

```
SELECT MASTER_POS_WAIT('mysql-bin-changelog.000031', 107);
```

If the read replica is past the specified position, the query returns immediately. Otherwise, the function waits. When the query returns for all read replicas, go to the next step.

5. Reset the GTID parameters to disable GTID-based replication:
 - a. Make sure that the DB cluster parameter group associated with the Aurora MySQL cluster has the following parameter settings:
 - `gtid_mode` – OFF
 - `enforce_gtid_consistency` – OFF
 - b. Restart the Aurora MySQL DB cluster.

Working with Aurora multi-master clusters

Following, you can learn about Aurora multi-master clusters. In a multi-master cluster, all DB instances have read/write capability. Multi-master clusters have different availability characteristics, support for database features, and procedures for monitoring and troubleshooting than single-master clusters.

Topics

- [Overview of Aurora multi-master clusters \(p. 958\)](#)
- [Creating an Aurora multi-master cluster \(p. 963\)](#)
- [Managing Aurora multi-master clusters \(p. 969\)](#)
- [Application considerations for Aurora multi-master clusters \(p. 972\)](#)
- [Performance considerations for Aurora multi-master clusters \(p. 981\)](#)
- [Approaches to Aurora multi-master clusters \(p. 983\)](#)

Overview of Aurora multi-master clusters

Use the following background information to help you choose a multi-master or single-master cluster when you set up a new Aurora cluster. For you to make an informed choice, we recommend that you first understand how you plan to adapt your schema design and application logic to work best with a multi-master cluster.

For each new Amazon Aurora cluster, you can choose whether to create a single-master or multi-master cluster.

Most kinds of Aurora clusters are *single-master* clusters. For example, provisioned, Aurora Serverless, parallel query, and Global Database clusters are all single-master clusters. In a single-master cluster, a single DB instance performs all write operations and any other DB instances are read-only. If the writer DB instance becomes unavailable, a failover mechanism promotes one of the read-only instances to be the new writer.

In a *multi-master* cluster, all DB instances can perform write operations. The notions of a single read/write primary instance and multiple read-only Aurora Replicas don't apply. There isn't any failover when a writer DB instance becomes unavailable, because another writer DB instance is immediately available to take over the work of the failed instance. We refer to this type of availability as *continuous availability*, to distinguish it from the high availability (with brief downtime during failover) offered by a single-master cluster.

Multi-master clusters work differently in many ways from the other kinds of Aurora clusters, such as provisioned, Aurora Serverless, and parallel query clusters. With multi-master clusters, you consider

different factors in areas such as high availability, monitoring, connection management, and database features. For example, in applications where you can't afford even brief downtime for database write operations, a multi-master cluster can help to avoid an outage when a writer instance becomes unavailable. The multi-master cluster doesn't use the failover mechanism, because it doesn't need to promote another DB instance to have read/write capability. With a multi-master cluster, you examine metrics related to DML throughput, latency, and deadlocks for all DB instances instead of a single primary instance.

Currently, multi-master clusters require Aurora MySQL version 1, which is compatible with MySQL 5.6. When specifying the DB engine version in the AWS Management Console, AWS CLI, or RDS API, choose 5.6.10a.

To create a multi-master cluster, you choose **Multiple writers** under **Database features** when creating the cluster. Doing so enables different behavior for replication among DB instances, availability, and performance than other kinds of Aurora clusters. This choice remains in effect for the life of the cluster. Make sure that you understand the specialized use cases that are appropriate for multi-master clusters.

Topics

- [Multi-master cluster terminology \(p. 959\)](#)
- [Multi-master cluster architecture \(p. 960\)](#)
- [Recommended workloads for multi-master clusters \(p. 961\)](#)
- [Advantages of multi-master clusters \(p. 962\)](#)
- [Limitations of multi-master clusters \(p. 962\)](#)

Multi-master cluster terminology

You can understand the terminology about multi-master clusters by learning the following definitions. These terms are used throughout the documentation for multi-master clusters.

Writer

A DB instance that can perform write operations. In an Aurora multi-master cluster, all DB instances are writers. This is a significant difference from Aurora single-master clusters, where only one DB instance can act as a writer. With a single-master cluster, if the writer becomes unavailable, the failover mechanism promotes another DB instance to become the new writer. With a multi-master cluster, your application can redirect write operations from the failed DB instance to any other DB instance in the cluster.

Multi-master

An architecture for Aurora clusters where each DB instance can perform both read and write operations. Contrast this with *single-master*. Multi-master clusters are best suited for segmented workloads, such as for multitenant applications.

Single-master

The default architecture for Aurora clusters. A single DB instance (the primary instance) performs writes. All other DB instances (the Aurora Replicas) handle read-only query traffic. Contrast this with *multi-master*. This architecture is appropriate for general-purpose applications. In such applications, a single DB instance can handle all the data manipulation language (DML) and data definition language (DDL) statements. Scalability issues mostly involve SELECT queries.

Write conflict

A situation that occurs when different DB instances attempt to modify the same data page at the same time. Aurora reports a write conflict to your application as a deadlock error. This error condition causes the transaction to roll back. Your application must detect the error code and retry the transaction.

The main design consideration and performance tuning goal with Aurora multi-master clusters is to divide your write operations between DB instances in a way that minimizes write conflicts. That is why multi-master clusters are well-suited for sharded applications. For details about the write conflict mechanism, see [Conflict resolution for multi-master clusters \(p. 982\)](#).

Sharding

A particular class of segmented workloads. The data is physically divided into many partitions, tables, databases, or even separate clusters. The containers for specific portions of the data are known as *shards*. In an Aurora multi-master cluster, each shard is managed by a specific DB instance, and a DB instance can be responsible for multiple shards. A sharded schema design maps well to the way you manage connections in an Aurora multi-master cluster.

Shard

The unit of granularity within a sharded deployment. It might be a table, a set of related tables, a database, a partition, or even an entire cluster. With Aurora multi-master clusters, you can consolidate the data for a sharded application into a single Aurora shared storage volume, making the database continuously available and the data easy to manage. You decide which shards are managed by each DB instance. You can change this mapping at any time, without physically reorganizing the data.

Resharding

Physically reorganizing sharded data so that different DB instances can handle specific tables or databases. You don't need to physically reorganize data inside Aurora multi-master clusters in response to changing workload or DB instance failures. You can avoid reshading operations because all DB instances in a cluster can access all databases and tables through the shared storage volume.

Multitenant

A particular class of segmented workloads. The data for each customer, client, or user is kept in a separate table or database. This design ensures isolation and helps you to manage capacity and resources at the level of individual users.

Bring-your-own-shard (BYOS)

A situation where you already have a database schema and associated applications that use sharding. You can transfer such deployments relatively easily to Aurora multi-master clusters. In this case, you can devote your effort to investigating the Aurora benefits such as server consolidation and high availability. You don't need to create new application logic to handle multiple connections for write requests.

Global read-after-write (GRAW)

A setting that introduces synchronization so that any read operations always see the most current state of the data. By default, the data seen by a read operation in a multi-master cluster is subject to replication lag, typically a few milliseconds. During this brief interval, a query on one DB instance might retrieve stale data if the same data is modified at the same time by a different DB instance. To enable this setting, change `aurora_mm_session_consistency_level` from its default setting of `INSTANCE_RAW` to `REGIONAL_RAW`. Doing so ensures cluster-wide consistency for read operations regardless of the DB instances that perform the reads and writes. For details on GRAW mode, see [Consistency model for multi-master clusters \(p. 974\)](#).

Multi-master cluster architecture

Multi-master clusters have a different architecture than other kinds of Aurora clusters. In multi-master clusters, all DB instances have read/write capability. Other kinds of Aurora clusters have a single dedicated DB instance that performs all write operations, while all other DB instances are read-only and handle only `SELECT` queries. Multi-master clusters don't have a primary instance or read-only Aurora Replicas.

Your application controls which write requests are handled by which DB instance. Thus, with a multi-master cluster, you connect to individual instance endpoints to issue DML and DDL statements. That's different than other kinds of Aurora clusters, where you typically direct all write operations to the single cluster endpoint and all read operations to the single reader endpoint.

The underlying storage for Aurora multi-master clusters is similar to storage for single-master clusters. Your data is still stored in a highly reliable, shared storage volume that grows automatically. The core difference lies in the number and type of DB instances. In multi-master clusters, there are N read/write nodes. Currently, the maximum for N is 4.

Multi-master clusters have no dedicated read-only nodes. Thus, the Aurora procedures and guidelines about Aurora Replicas don't apply to multi-master clusters. You can temporarily make a DB instance read-only to place read and write workloads on different DB instances. To do so, see [Using instance read-only mode \(p. 980\)](#).

Multi-master cluster nodes are connected using low-latency and low-lag Aurora replication. Multi-master clusters use all-to-all peer-to-peer replication. Replication works directly between writers. Every writer replicates its changes to all other writers.

DB instances in a multi-master cluster handle restart and recovery independently. If a writer restarts, there is no requirement for other writers to also restart. For details, see [High availability considerations for Aurora multi-master clusters \(p. 971\)](#).

Multi-master clusters keep track of all changes to data within all database instances. The unit of measurement is the data page, which has a fixed size of 16 KB. These changes include modifications to table data, secondary indexes, and system tables. Changes can also result from Aurora internal housekeeping tasks. Aurora ensures consistency between the multiple physical copies that Aurora keeps for each data page in the shared storage volume, and in memory on the DB instances.

If two DB instances attempt to modify the same data page at almost the same instant, a write conflict occurs. The earliest change request is approved using a quorum voting mechanism. That change is saved to permanent storage. The DB instance whose change isn't approved rolls back the entire transaction containing the attempted change. Rolling back the transaction ensures that data is kept in a consistent state, and applications always see a predictable view of the data. Your application can detect the deadlock condition and retry the entire transaction.

For details about how to minimize write conflicts and associated performance overhead, see [Conflict resolution for multi-master clusters \(p. 982\)](#).

Recommended workloads for multi-master clusters

Multi-master clusters work best with certain kinds of workloads.

Active-passive workloads

With an *active-passive* workload, you perform all read and write operations on one DB instance at a time. You hold any other DB instances in the Aurora cluster in reserve. If the original active DB instance becomes unavailable, you immediately switch all read and write operations to the other DB instance. With this configuration, you minimize any downtime for write operations. The other DB instance can take over all processing for your application without performing a failover.

Active-active workloads

With an *active-active* workload, you perform read and write operations to all the DB instances at the same time. In this configuration, you typically segment the workload so that the different DB instances don't modify the same underlying data at the same time. Doing so minimizes the chance for write conflicts.

Multi-master clusters work well with application logic that's designed for a *segmented workload*. In this type of workload, you divide write operations by database instance, database, table, or table partition. For example, you can run multiple applications on the same cluster, each assigned to a specific DB instance. Alternatively, you can run an application that uses multiple small tables, such as one table for each user of an online service. Ideally, you design your schema so that write operations for different DB instances don't perform simultaneous updates to overlapping rows within the same tables. Sharded applications are one example of this kind of architecture.

For examples of designs for active-active workloads, see [Using a multi-master cluster for a sharded database \(p. 983\)](#).

Advantages of multi-master clusters

You can take advantage of the following benefits with Aurora multi-master clusters:

- Multi-master clusters improve Aurora's already high availability. You can restart a read/write DB instance without causing other DB instances in the cluster to restart. There is no failover process and associated delay when a read/write DB instance becomes unavailable.
- Multi-master clusters are well-suited to sharded or multitenant applications. As you manage the data, you can avoid complex resharding operations. You might be able to consolidate sharded applications with a smaller number of clusters or DB instances. For details, see [Using a multi-master cluster for a sharded database \(p. 983\)](#).
- Aurora detects write conflicts immediately, not when the transaction commits. For details about the write conflict mechanism, see [Conflict resolution for multi-master clusters \(p. 982\)](#).

Limitations of multi-master clusters

Note

Aurora multi-master clusters are highly specialized for continuous availability use cases. Thus, such clusters might not be generally applicable to all workloads. Your requirements for performance, scalability, and availability might be satisfied by using a larger DB instance class with an Aurora single-master cluster. If so, consider using a provisioned or Aurora Serverless cluster.

AWS and Aurora limitations

The following limitations currently apply to the AWS and Aurora features that you can use with multi-master clusters:

- Currently, you can have a maximum of four DB instances in a multi-master cluster.
- Currently, all DB instances in a multi-master cluster must be in the same AWS Region.
- You can't enable cross-Region replicas from multi-master clusters.
- Multi-master clusters are available in the following AWS Regions:
 - US East (N. Virginia) Region
 - US East (Ohio) Region
 - US West (Oregon) Region
 - Asia Pacific (Mumbai) Region
 - Asia Pacific (Seoul) Region
 - Asia Pacific (Tokyo) Region
 - Europe (Frankfurt) Region
 - Europe (Ireland) Region
- The Stop action isn't available for multi-master clusters.

- The Aurora survivable page cache, also known as the survivable buffer pool, isn't supported for multi-master clusters.
- A multi-master cluster doesn't do any load balancing for connections. Your application must implement its own connection management logic to distribute read and write operations among multiple DB instance endpoints. Typically, in a bring-your-own-shard (BYOS) application, you already have logic to map each shard to a specific connection. To learn how to adapt the connection management logic in your application, see [Connection management for multi-master clusters \(p. 973\)](#).
- Multi-master clusters have some processing and network overhead for coordination between DB instances. This overhead has the following consequences for write-intensive and read-intensive applications:
 - Throughput benefits are most obvious on busy clusters with multiple concurrent write operations. In many cases, a traditional Aurora cluster with a single primary instance can handle the write traffic for a cluster. In these cases, the benefits of multi-master clusters are mostly for high availability rather than performance.
 - Single-query performance is generally lower than for an equivalent single-master cluster.
- You can't take a snapshot created on a single-master cluster and restore it on a multi-master cluster, or the opposite. Instead, to transfer all data from one kind of cluster to the other, use a logical dump produced by a tool such as AWS Database Migration Service (AWS DMS) or the `mysqldump` command.
- You can't use the parallel query, Aurora Serverless, or Global Database features on a multi-master cluster.

The multi-master aspect is a permanent choice for a cluster. You can't switch an existing Aurora cluster between a multi-master cluster and another kind such as Aurora Serverless or parallel query.

- The zero-downtime patching (ZDP) and zero-downtime restart (ZDR) features aren't available for multi-master clusters.
- Integration with other AWS services such as AWS Lambda, Amazon S3, and AWS Identity and Access Management isn't available for multi-master clusters.
- The Performance Insights feature isn't available for multi-master clusters.
- You can't clone a multi-master cluster.
- You can't enable the backtrack feature for multi-master clusters.

Database engine limitations

The following limitations apply to the database engine features that you can use with a multi-master cluster:

- You can't perform binary log (binlog) replication to or from a multi-master cluster. This limitation means you also can't use global transaction ID (GTID) replication in a multi-master cluster.
- The event scheduler isn't available for multi-master clusters.
- The hash join optimization isn't enabled on multi-master clusters.
- The query cache isn't available on multi-master clusters.
- You can't use certain SQL language features on multi-master clusters. For the full list of SQL differences, and instructions about adapting your SQL code to address these limitations, see [SQL considerations for multi-master clusters \(p. 972\)](#).

Creating an Aurora multi-master cluster

You choose the multi-master or single-master architecture at the time you create an Aurora cluster. The following procedures show where to make the multi-master choice. If you haven't created any Aurora clusters before, you can learn the general procedure in [Creating an Amazon Aurora DB cluster \(p. 125\)](#).

Console

To create an Aurora multi-master cluster from the AWS Management Console, you make the following choices. On the first screen, you select an Aurora cluster:

Create database

Database settings



Quick create [Info](#)

Provides the fastest way to get started with your database. You can modify

Engine options

Amazon Aurora

Amazon
Aurora

MySQL



PostgreSQL



Oracle

ORACLE®

You also choose MySQL 5.6 compatibility and location **Regional**:

Edition

- Amazon Aurora with MySQL compatibility
- Amazon Aurora with PostgreSQL compatibility

DB engine version [Info](#)

Aurora MySQL 1.21.0 (Compatible with MySQL 5.6)

Select engine version Aurora MySQL 1.21.0 (Compatible with MySQL 5.6) to use create parallel query, multiple writers, serverless, or global databases.

Database location

Regional [Info](#)

You provision your Aurora database in a single region.

Global [Info](#)

You can provision your Aurora database in multiple regions. Writes in the primary typical latency of <1 sec to secondary regions.

On the second screen, choose **Multiple writers** under **Database features**.

Database features

One writer and multiple readers

The readers connect to the same storage volume as the writer instance and supports only read operations. [Info](#)

Multiple writers

Supports read and write operations, and performs all of the data modifications to the cluster volume. [Info](#)

One writer

You provi...
Aurora im...
pushing p...
[Info](#)

Serverless

You speci...
resources
on databa...

Fill in the other settings for the cluster. This part of the procedure is the same as the general procedure for creating an Aurora cluster in [Creating a DB cluster \(p. 126\)](#).

After you create the multi-master cluster, add two DB instances to it by following the procedure in [Adding Aurora Replicas to a DB cluster \(p. 392\)](#). Use the same AWS instance class for all DB instances within the multi-master cluster.

After you create the multi-master cluster and associated DB instances, you see the cluster in the AWS Management Console **Databases** page as follows. All DB instances show the role **Writer**.

The screenshot shows the 'Databases' section of the Amazon RDS console. At the top, there is a search bar labeled 'Filter databases'. Below it, a table lists database instances. The columns are 'DB Name', 'Role', and 'CPU'. There are two rows under 'db-multi-master': 'instance-name1' (Writer, 6.72 CPU) and 'instance-name2' (Writer, 6.72 CPU). The 'DB Name' column has a dropdown arrow icon.

DB Name	Role	CPU
db-multi-master	Cluster	-
instance-name1	Writer	6.72
instance-name2	Writer	6.72

AWS CLI

To create a multi-master cluster with the AWS CLI, run the [create-db-cluster](#) AWS CLI command and include the option `--engine_mode=multimaster`.

The following command shows the syntax for creating an Aurora cluster with multi-master replication. For the general procedure to create an Aurora cluster, see [Creating a DB cluster \(p. 126\)](#).

For Linux, macOS, or Unix:

```
aws rds create-db-cluster --db-cluster-identifier sample-cluster --engine aurora \
    --engine-version 5.6.10a --master-username user-name --master-user-password password \
    --db-subnet-group-name my_subnet_group --vpc-security-group-ids my_vpc_id \
    --engine-mode multimaster
```

For Windows:

```
aws rds create-db-cluster --db-cluster-identifier sample-cluster --engine aurora ^
    --engine-version 5.6.10a --master-username user-name --master-user-password password ^
    --db-subnet-group-name my_subnet_group --vpc-security-group-ids my_vpc_id ^
    --engine-mode multimaster
```

After you create the multi-master cluster, add a second DB instance to it by following the procedure in [Adding Aurora Replicas to a DB cluster \(p. 392\)](#). Use the same AWS instance class for all DB instances within the multi-master cluster.

RDS API

To create a multi-master cluster with the RDS API, run the [CreateDBCluster](#) operation. Specify the value `multimaster` for the `EngineMode` parameter. For the general procedure to create an Aurora cluster, see [Creating a DB cluster \(p. 126\)](#).

After you create the multi-master cluster, add two DB instances to it by following the procedure in [Adding Aurora Replicas to a DB cluster \(p. 392\)](#). Use the same AWS instance class for all DB instances within the multi-master cluster.

Adding a DB instance to a multi-master cluster

You need more than one DB instance to see the benefits of a multi-master cluster. After you create the first instance, you can create other DB instances, up to a maximum of four DB instances, using the procedures from [Adding Aurora Replicas to a DB cluster \(p. 392\)](#). The difference for multi-master clusters is that the new DB instances all have read/write capability instead of being read-only Aurora Replicas. Use the same AWS instance class for all DB instances within the multi-master cluster.

Managing Aurora multi-master clusters

You do most management and administration for Aurora multi-master clusters the same way as for other kinds of Aurora clusters. The following sections explain the differences and unique features of multi-master clusters for administration and management.

Topics

- [Monitoring an Aurora multi-master cluster \(p. 969\)](#)
- [Data ingestion performance for multi-master clusters \(p. 970\)](#)
- [Exporting data from a multi-master cluster \(p. 970\)](#)
- [High availability considerations for Aurora multi-master clusters \(p. 971\)](#)
- [Replication between multi-master clusters and other clusters \(p. 971\)](#)
- [Upgrading a multi-master cluster \(p. 971\)](#)

Monitoring an Aurora multi-master cluster

Most of the monitoring and diagnostic features supported by MySQL and Aurora single-master clusters are also supported for multi-master clusters:

- MySQL error logs, general logs and slow query logs.
- MySQL built-in diagnostic features such as SHOW commands, status variables, InnoDB runtime status tables, and so on.
- MySQL Performance Schema.
- Advanced Auditing.
- CloudWatch metrics.
- Enhanced Monitoring.

Aurora multi-master clusters don't currently support the following monitoring features:

- Performance Insights.

Data ingestion performance for multi-master clusters

One best practice for DML operations on a multi-master cluster is to keep transactions small and brief. Also, route write operations for a particular table or database to a specific DB instance. Doing a bulk import might require relaxing the guidance for transaction size. However, you can still distribute the write operations to minimize the chance of write conflicts.

To distribute the write workload from a bulk import

1. Issue a separate `mysqldump` command for each database, table, or other object in your schema. Store the results of each `mysqldump` in a file whose name reflects the object being dumped. As an alternative, you can use a specialized dump and import tool that can automatically dump multiple tables in parallel, such as `mydumper`.
2. Run a separate `mysql` session for each data file, connecting to the appropriate instance endpoint that handles the corresponding schema object. Again, as an alternative, you can use a specialized parallel import command, such as `myloader`.
3. Run the import sessions in parallel across the DB instances in the multi-master cluster, instead of waiting for each to finish before starting the next.

You can use the following techniques to import data into an Aurora multi-master cluster:

- You can import logical (SQL-format) dumps from other MySQL-compatible servers to Aurora multi-master clusters, if the statements don't use any features that aren't supported in Aurora. For example, a logical dump from a table containing MySQL Full-Text Search (FTS) indexes doesn't work because the FTS feature is not supported on multi-master clusters.
- You can use managed services such as DMS to migrate data into an Aurora multi-master cluster.
- For migrations into an Aurora multi-master cluster from a server that isn't compatible with MySQL, follow existing instructions for heterogeneous Aurora migrations.
- Aurora multi-master clusters can produce MySQL-compatible logical dumps in SQL format. Any migration tool (for example, AWS DMS) that can understand such format can consume data dumps from Aurora multi-master clusters.
- Aurora doesn't support binary logging with the multi-master cluster as the binlog master or worker. You can't use binlog-based CDC tools with multi-master clusters.
- When migrating from non-MySQL-compatible servers, you can replicate into a multi-master cluster using the continuous change data capture (CDC) feature of AWS DMS. That type of replication transmits SQL statements to the destination cluster, thus the restriction on binlog replication doesn't apply.

For a detailed discussion of migration techniques and recommendations, see the [Amazon Aurora migration handbook](#) AWS whitepaper. Some of the migration methods listed in the handbook might not apply to Aurora multi-master clusters, but the document is a great overall source of knowledge about Aurora migration topics.

Exporting data from a multi-master cluster

You can save a snapshot of a multi-master cluster and restore it to another multi-master cluster. Currently, you can't restore a multi-master cluster snapshot into a single-master cluster.

To migrate data from a multi-master cluster to a single-master cluster, use a logical dump and restore with a tool such as `mysqldump`.

You can't use a multi-master cluster as the source or destination for binary log replication.

High availability considerations for Aurora multi-master clusters

In an Aurora multi-master cluster, any DB instance can restart without causing any other instance to restart. This behavior provides a higher level of availability for read/write and read-only connections than for Aurora single-master clusters. We refer to this availability level as *continuous availability*. In multi-master clusters, there is no downtime for write availability when a writer DB instance fails. Multi-master clusters don't use the failover mechanism, because all cluster instances are writable. If a DB instance fails in a multi-master cluster, your application can redirect the workload towards the remaining healthy instances.

In a single-master cluster, restarting the primary instance makes write operations unavailable until the failover mechanism promotes a new primary instance. Read-only operations also experience a brief downtime because all the Aurora Replicas in the cluster restart.

To minimize downtime for applications in a multi-master cluster, implement frequent SQL-level health checks. If a DB instance in a multi-master cluster becomes unavailable, you can decide what to do based on the expected length of the outage and the urgency of write operations in the workload. If you expect the outage to be brief and the write operations aren't urgent, you can wait for the DB instance to recover before resuming the workload that is normally handled by that DB instance. Alternatively, you can redirect that workload to a different DB instance. The underlying data remains available at all time to all DB instances in the cluster. The highly distributed Aurora storage volume keeps the data continuously available even in the unlikely event of a failure affecting an entire AZ. For information about the timing considerations for switching write operations away from an unavailable DB instance, see [Using a multi-master cluster as an active standby \(p. 984\)](#).

Replication between multi-master clusters and other clusters

Multi-master clusters don't support incoming or outgoing binary log replication.

Upgrading a multi-master cluster

Aurora multi-master clusters use the same version numbering scheme, with major and minor version numbers, as other kinds of Aurora clusters. However, the **Enable auto minor version upgrade** setting doesn't apply for multi-master clusters.

When you upgrade an Aurora multi-master cluster, typically the upgrade procedure moves the database engine from the current version to the next higher version. If you upgrade to an Aurora version that increments the version number by more than one, the upgrade uses a multi-step approach. Each DB instance is upgraded to the next higher version, then the next one after that, and so on until it reaches the specified upgrade version.

The approach is different depending on whether there are any backwards-incompatible changes between the old and new versions. For example, updates to the system schema are considered backwards-incompatible changes. You can check whether a specific version contains any backwards-incompatible changes by consulting the release notes.

If there aren't any incompatible changes between the old and new versions, each DB instance is upgraded and restarted individually. The upgrades are staggered so that the overall cluster doesn't experience any downtime. At least one DB instance is available at any time during the upgrade process.

If there are incompatible changes between the old and new versions, Aurora performs the upgrade in offline mode. All cluster nodes are upgraded and restarted at the same time. The cluster experiences some downtime, to avoid an older engine writing to newer system tables.

Zero-downtime patching (ZDP) isn't currently supported for Aurora multi-master clusters.

Application considerations for Aurora multi-master clusters

Following, you can learn any changes that might be required in your applications due to differences in feature support or behavior between multi-master and single-master clusters.

Topics

- [SQL considerations for multi-master clusters \(p. 972\)](#)
- [Connection management for multi-master clusters \(p. 973\)](#)
- [Consistency model for multi-master clusters \(p. 974\)](#)
- [Multi-master clusters and transactions \(p. 975\)](#)
- [Write conflicts and deadlocks in multi-master clusters \(p. 975\)](#)
- [Multi-master clusters and locking reads \(p. 976\)](#)
- [Performing DDL operations on a multi-master cluster \(p. 977\)](#)
- [Using autoincrement columns \(p. 978\)](#)
- [Multi-master clusters feature reference \(p. 979\)](#)

SQL considerations for multi-master clusters

The following are the major limitations that apply to the SQL language features you can use with a multi-master cluster:

- In a multi-master cluster, you can't use certain settings or column types that change the row layout. You can't enable the `innodb_large_prefix` configuration option. You can't use the column types `MEDIUMTEXT`, `MEDIUMBLOB`, `LONGTEXT`, or `LONGBLOB`.
- You can't use the `CASCADE` clause with any foreign key columns in a multi-master cluster.
- Multi-master clusters can't contain any tables with full-text search (FTS) indexes. Such tables can't be created on or imported into multi-master clusters.
- DDL works differently on multi-master and single-master clusters. For example, the fast DDL mechanism isn't available for multi-master clusters. You can't write to a table in a multi-master cluster while the table is undergoing DDL. For full details on DDL differences, see [Performing DDL operations on a multi-master cluster \(p. 977\)](#).
- You can't use the `SERIALIZABLE` transaction isolation level on multi-master clusters. On Aurora single-master clusters, you can use this isolation level on the primary instance.
- Autoincrement columns are handled using the `auto_increment_increment` and `auto_increment_offset` parameters. Parameter values are predetermined and not configurable. The parameter `auto_increment_increment` is set to 16, which is the maximum number of instances in any Aurora cluster. However, multi-master clusters currently have a lower limit on the number of DB instances. For details, see [Using autoincrement columns \(p. 978\)](#).

When adapting an application for an Aurora multi-master cluster, approach that activity the same as a migration. You might have to stop using certain SQL features, and change your application logic for other SQL features:

- In your `CREATE TABLE` statements, change any columns defined as `MEDIUMTEXT`, `MEDIUMBLOB`, `LONGTEXT`, or `LONGBLOB` to shorter types that don't require off-page storage.
- In your `CREATE TABLE` statements, remove the `CASCADE` clause from any foreign key declarations. Add application logic if necessary to emulate the `CASCADE` effects through `INSERT` or `DELETE` statements.

- Remove any use of InnoDB fulltext search (FTS) indexes. Check your source code for `MATCH()` operators in `SELECT` statements, and `FULLTEXT` keywords in DDL statements. Check if any table names from the `INFORMATION_SCHEMA.INNODB_SYS_TABLES` system table contain the string `FTS_`.
- Check the frequency of DDL operations such as `CREATE TABLE` and `DROP TABLE` in your application. Because DDL operations have more overhead in multi-master clusters, avoid running many small DDL statements. For example, look for opportunities to create needed tables ahead of time. For information about DDL differences with multi-master clusters, see [Performing DDL operations on a multi-master cluster \(p. 977\)](#).
- Examine your use of autoincrement columns. The sequences of values for autoincrement columns are different for multi-master clusters than other kinds of Aurora clusters. Check for the `AUTO_INCREMENT` keyword in DDL statements, the function name `last_insert_id()` in `SELECT` statements, and the name `innodb_autoinc_lock_mode` in your custom configuration settings. For details about the differences and how to handle them, see [Using autoincrement columns \(p. 978\)](#).
- Check your code for the `SERIALIZABLE` keyword. You can't use this transaction isolation level with a multi-master cluster.

Connection management for multi-master clusters

The main connectivity consideration for multi-master clusters is the number and type of the available DNS endpoints. With multi-master clusters, you often use the instance endpoints, which you rarely use in other kinds of Aurora clusters.

Aurora multi-master clusters have the following kinds of endpoints:

Cluster endpoint

This type of endpoint always points to a DB instance with read/write capability. Each multi-master cluster has one cluster endpoint.

Because applications in multi-master clusters typically include logic to manage connections to specific DB instances, you rarely need to use this endpoint. It's mostly useful for connecting to a multi-master cluster to perform administration.

You can also connect to this endpoint to examine the cluster topology when you don't know the status of the DB instances in the cluster. To learn that procedure, see [Describing cluster topology \(p. 980\)](#).

DB instance endpoint

This type of endpoint connects to specific named DB instances. For Aurora multi-master clusters, your application typically uses the DB instance endpoints for all or nearly all connections. You decide which DB instance to use for each SQL statement based on the mapping between your shards and the DB instances in the cluster. Each DB instance has one such endpoint. Thus the multi-master cluster has one or more of these endpoints, and the number changes as DB instances are added to or removed from a multi-master cluster.

The way you use DB instance endpoints is different between single-master and multi-master clusters. For single-master clusters, you typically don't use this endpoint often.

Custom endpoint

This type of endpoint is optional. You can create one or more custom endpoints to group together DB instances for a specific purpose. When you connect to the endpoint, Aurora returns the IP address of a different DB instance each time. In multi-master clusters, you typically use custom endpoints to designate a set of DB instances to use mostly for read operations. We recommend not using custom endpoints with multi-master clusters to load-balance write operations, because doing so increases the chance of write conflicts.

Multi-master clusters don't have reader endpoints. Where practical, issue `SELECT` queries using the same DB instance endpoint that normally writes to the same table. Doing so makes more effective use of cached data from the buffer pool, and avoids potential issues with stale data due to replication lag within the cluster. If you don't locate `SELECT` statements on the same DB instances that write to the same tables, and you require strict read after write guarantee for certain queries, consider running those queries using the global read-after-write (GRAW) mechanism described in [Consistency model for multi-master clusters \(p. 974\)](#).

For general best practices of Aurora and MySQL connection management, see the [Amazon Aurora migration handbook](#) AWS whitepaper.

For information about how to emulate read-only DB instances in multi-master DB clusters, see [Using instance read-only mode \(p. 980\)](#).

Follow these guidelines when creating custom DNS endpoints and designing drivers and connectors for Aurora multi-master clusters:

- For DDL, DML, and DCL statements, don't use endpoints or connection routing techniques that operate in round-robin or random fashion.
- Avoid long-running write queries and long write transactions unless these transactions are guaranteed not to conflict with other write traffic in the cluster.
- Prefer to use autocommitted transactions. Where practical, avoid `autocommit=0` settings at global or session level. When you use a database connector or database framework for your programming language, check that `autocommit` is turned on for applications that use the connector or framework. If needed, add `COMMIT` statements at logical points throughout your code to ensure that transactions are brief.
- When global read consistency or read-after-write guarantee is required, follow recommendations for global read-after-write (GRAW) described in [Consistency model for multi-master clusters \(p. 974\)](#).
- Use the cluster endpoint for DDL and DCL statements where practical. The cluster endpoint helps to minimize the dependency on the hostnames of the individual DB instances. You don't need to divide DDL and DCL statements by table or database, as you do with DML statements.

Consistency model for multi-master clusters

Aurora multi-master clusters support a global read-after-write (GRAW) mode that is configurable at the session level. This setting introduces extra synchronization to create a consistent read view for each query. That way, queries always see the very latest data. By default, the replication lag in a multi-master cluster means that a DB instance might see old data for a few milliseconds after the data was updated. Enable this feature if your application depends on queries seeing the latest data changes made by any other DB instance, even if the query has to wait as a result.

Note

Replication lag doesn't affect your query results if you write and then read the data using the same DB instance. Thus, the GRAW feature applies mainly to applications that issue multiple concurrent write operations through different DB instances.

When using the GRAW mode, don't enable it for all queries by default. Globally consistent reads are noticeably slower than local reads. Therefore, use GRAW selectively for queries that require it.

Be aware of these considerations for using GRAW:

- GRAW involves performance overhead due to the cost of establishing a cluster-wide consistent read view. The transaction must first determine a cluster-wide consistent point in time, then replication must catch up to that time. The total delay depends on the workload, but it's typically in the range of tens of milliseconds.
- You can't change GRAW mode within a transaction.

- When using GRAW without explicit transactions, each individual query incurs the performance overhead of establishing a globally consistent read view.
- With GRAW enabled, the performance penalty applies to both reads and writes.
- When you use GRAW with explicit transactions, the overhead of establishing a globally consistent view applies once for each transaction, when the transaction starts. Queries performed later in the transaction are as fast as if run without GRAW. If multiple successive statements can all use the same read view, you can wrap them in a single transaction for a better overall performance. That way, the penalty is only paid once per transaction instead of per query.

Multi-master clusters and transactions

Standard Aurora MySQL guidance applies to Aurora multi-master clusters. The Aurora MySQL database engine is optimized for short-lived SQL statements. These are the types of statements typically associated with online transaction processing (OLTP) applications.

In particular, make your write transactions as short as possible. Doing so reduces the window of opportunity for write conflicts. The conflict resolution mechanism is *optimistic*, meaning that it performs best when write conflicts are rare. The tradeoff is that when conflicts occur, they incur substantial overhead.

There are certain workloads that benefit from large transactions. For example, bulk data imports are significantly faster when run using multi-megabyte transactions rather than single-statement transactions. If you observe an unacceptable number of conflicts while running such workloads, consider the following options:

- Reduce transaction size.
- Reschedule or rearrange batch jobs so that they don't overlap and don't provoke conflicts with other workloads. If practical, reschedule the batch jobs so that they run during off-peak hours.
- Refactor the batch jobs so that they run on the same writer instance as the other transactions causing conflicts. When conflicting transactions are run on the same instance, the transactional engine manages access to the rows. In that case, storage-level write conflicts don't occur.

Write conflicts and deadlocks in multi-master clusters

One important performance aspect for multi-master clusters is the frequency of write conflicts. When such a problem condition occurs in the Aurora storage subsystem, your application receives a deadlock error and performs the usual error handling for deadlock conditions. Aurora uses a lock-free optimistic algorithm that performs best when such conflicts are rare.

In a multi-master cluster, all the DB instances can write to the shared storage volume. For every data page you modify, Aurora automatically distributes several copies across multiple Availability Zones (AZs). A write conflict can occur when multiple DB instances try to modify the same data page within a very short time. The Aurora storage subsystem detects that the changes overlap and performs conflict resolution before finalizing the write operation.

Aurora detects write conflicts at the level of the physical data pages, which have a fixed size of 16 KiB. Thus, a conflict can occur even for changes that affect different rows, if the rows are both within the same data page.

When conflicts do occur, the cleanup operation requires extra work to undo the changes from one of the DB instances. From the point of view of your application, the transaction that caused the conflict encounters a deadlock and Aurora rolls back that whole transaction. Your application receives error code 1213.

Undoing the transaction might require modifying many other data pages whose changes were already applied to the Aurora storage subsystem. Depending on how much data was changed by the transaction,

undoing it might involve substantial overhead. Therefore, minimizing the potential for write conflicts is a crucial design consideration for an Aurora multi-master cluster.

Some conflicts result from changes that you initiate. These changes include SQL statements, transactions, and transaction rollbacks. You can minimize these kinds of conflicts through your schema design and the connection management logic in your application.

Other conflicts happen because of simultaneous changes from both a SQL statement and an internal server thread. These conflicts are hard to predict because they depend on internal server activity that you might not be aware of. The two major kinds of internal activity that cause these conflicts are garbage collection (known as *purge*), and transaction rollbacks performed automatically by Aurora. For example, Aurora performs rollbacks automatically during crash recovery or if a client connection is lost.

A transaction rollback physically reverts page changes that were already made. A rollback produces page changes just like the original transaction does. A rollback takes time, potentially several times as long as the original transaction. While the rollback is proceeding, the changes it produces can come into conflict with your transactions.

Garbage collection has to do with multi-version concurrency control (MVCC), which is the concurrency control method used by the Aurora MySQL transactional engine. With MVCC, data mutations create new row versions, and the database keeps multiple versions of rows to achieve transaction isolation while permitting concurrent access to data. Row versions are removed (*purged*) when they're no longer needed. Here again, the process of purging produces page changes, which might conflict with your transactions. Depending on the workload, the database can develop a *purge lag*: a queue of changes waiting to be garbage collected. If the lag grows substantially, the database might need a considerable amount of time to complete the purge, even if you stop submitting SQL statements.

If an internal server thread encounters a write conflict, Aurora retries automatically. In contrast, your application must handle the retry logic for any transactions that encounter conflicts.

When multiple transactions from the same DB instance cause these kinds of overlapping changes, Aurora uses the standard transaction concurrency rules. For example, if two transactions on the same DB instance modify the same row, one of them waits. If the wait is longer than the configured timeout (`innodb_lock_wait_timeout`, by default 50 seconds), the waiting transaction aborts with a "Lock wait timeout exceeded" message.

Multi-master clusters and locking reads

Aurora multi-master clusters support locking reads in the following forms.

```
SELECT ... FOR UPDATE
SELECT ... LOCK IN SHARE MODE
```

For more information about locking reads, see the [MySQL reference manual](#).

Locking read operations are supported on all nodes, but the lock scope is local to the node on which the command was run. A locking read performed on one writer doesn't prevent other writers from accessing or modifying the locked rows. Despite this limitation, you can still work with locking reads in use cases that guarantee strict workload scope separation between writers, such as in sharded or multitenant databases.

Consider the following guidelines:

- Remember that a node can always see its own changes immediately and without delay. When possible, you can colocate reads and writes on the same node to eliminate the GRAW requirement.
- If read-only queries must be run with globally consistent results, use GRAW.
- If read-only queries care about data visibility but not global consistency, use GRAW or introduce a timed wait before each read. For example, a single application thread might maintain connections C1 and C2 to two different nodes. The application writes on C1 and reads on C2. In such case, the

application can issue a read immediately using GRAW, or it can sleep before issuing a read. The sleep time should be equal to or longer than the replication lag (usually approximately 20–30 ms).

The read-after-write feature is controlled using the `aurora_mm_session_consistency_level` session variable. The valid values are `INSTANCE_RAW` for local consistency mode (default) and `REGIONAL_RAW` for cluster-wide consistency:

Performing DDL operations on a multi-master cluster

The SQL data definition language (DDL) statements have special considerations for multi-master clusters. These statements sometimes cause substantial reorganization of the underlying data. Such large-scale changes potentially affect many data pages in the shared storage volume. The definitions of tables and other schema objects are held in the `INFORMATION_SCHEMA` tables. Aurora handles changes to those tables specially to avoid write conflicts when multiple DB instances run DDL statements at the same time.

For DDL statements, Aurora automatically delegates the statement processing to a special server process in the cluster. Because Aurora centralizes the changes to the `INFORMATION_SCHEMA` tables, this mechanism avoids the potential for write conflicts between DDL statements.

DDL operations prevent concurrent writes to that table. During a DDL operation on a table, all DB instances in the multi-master cluster are limited to read-only access to that table until the DDL statement finishes.

The following DDL behaviors are the same in Aurora single-master and multi-master clusters:

- A DDL performed on one DB instance causes other instances to terminate any connections actively using the table.
- Session-level temporary tables can be created on any node using the `MyISAM` or `MEMORY` storage engines.
- DDL operations on very large tables might fail if the DB instance doesn't have sufficient local temporary storage.

Note the following DDL performance considerations in multi-master clusters:

- Try to avoid issuing large numbers of short DDL statements in your application. Create databases, tables, partitions, columns, and so on, in advance where practical. Replication overhead can impose significant performance overhead for simple DDL statements that are typically very quick. The statement doesn't finish until the changes are replicated to all DB instances in the cluster. For example, multi-master clusters take longer than other Aurora clusters to create empty tables, drop tables, or drop schemas containing many tables.

If you do need to perform a large set of DDL operations, you can reduce the network and coordination overhead by issuing the statements in parallel through multiple threads.

- Long-running DDL statements are less affected, because the replication delay is only a small fraction of the total time for the DDL statement.
- Performance of DDLs on session-level temporary tables should be roughly equivalent on Aurora single-master and multi-master clusters. Operations on temporary tables happen locally and are not subject to synchronous replication overhead.

Using Percona online schema change with multi-master clusters

The `pt-online-schema-change` tool works with multi-master clusters. You can use it if your priority is to run table modifications in the most nonblocking manner. However, be aware of the write conflict implications of the schema change process.

At a high level, the `pt-online-schema-change` tool works as follows:

1. It creates a new, empty table with the desired structure.
2. It creates `DELETE`, `INSERT` and `UPDATE` triggers on the original table to redo any data changes on the original table on top of the new table.
3. It moves existing rows into the new table using small chunks while ongoing table changes are automatically handled using the triggers.
4. After all the data is moved, it drops the triggers and switches the tables by renaming them.

The potential contention point occurs while the data is being transferred to the new table. When the new table is initially created, it's completely empty and therefore can become a locking hot point. The same is true in other kinds of database systems. Because triggers are synchronous, the impact from the hot point can propagate back to your queries.

In multi-master clusters, the impact can be more visible. This visibility is because the new table not only provokes lock contention, but also increases the likelihood of write conflicts. The table initially has very few pages in it, which means that writes are highly localized and therefore prone to conflicts. After the table grows, writes should spread out and write conflicts should no longer be a problem.

You can use the online schema change tool with multi-master clusters. However, it might require more careful testing and its effects on the ongoing workload might be slightly more visible in the first minutes of the operation.

Using autoincrement columns

Aurora multi-master clusters handle autoincrement columns using the existing configuration parameters `auto_increment_increment` and `auto_increment_offset`. For more information, see the [MySQL reference manual](#).

Parameter values are predetermined and you can't change them. Specifically, the `auto_increment_increment` parameter is hardcoded to 16, which is the maximum number of DB instances in any kind of Aurora cluster.

Due to the hard-coded increment setting, autoincrement values are consumed much more quickly than in single-master clusters. This is true even if a given table is only ever modified by a single DB instance. For best results, always use a `BIGINT` data type instead of `INT` for your autoincrement columns.

In a multi-master cluster, your application logic must be prepared to tolerate autoincrement columns that have the following properties:

- The values are noncontiguous.
- The values might not start from 1 on an empty table.
- The values increase by increments greater than 1.
- The values are consumed significantly more quickly than in a single-master cluster.

The following example shows how the sequence of autoincrement values in a multi-master cluster can be different from what you might expect.

```
mysql> create table autoinc (id bigint not null auto_increment, s varchar(64), primary key (id));
mysql> insert into autoinc (s) values ('row 1'), ('row 2'), ('row 3');
Query OK, 3 rows affected (0.02 sec)

mysql> select * from autoinc order by id;
```

```
+----+-----+
| id | s   |
+----+-----+
|  2 | row 1 |
| 18 | row 2 |
| 34 | row 3 |
+----+-----+
3 rows in set (0.00 sec)
```

You can change the `AUTO_INCREMENT` table property. Using a nondefault value only works reliably if that value is larger than any of the primary key values already in the table. You can't use smaller values to fill in an empty interval in the table. If you do, the change takes effect either temporarily or not at all. This behavior is inherited from MySQL 5.6 and is not specific to the Aurora implementation.

Multi-master clusters feature reference

Following, you can find a quick reference of the commands, procedures, and status variables specific to Aurora multi-master clusters.

Using read-after-write

The read-after-write feature is controlled using the `aurora_mm_session_consistency_level` session variable. The valid values are `INSTANCE_RAW` for local consistency mode (default) and `REGIONAL_RAW` for cluster-wide consistency.

An example follows.

```
mysql> select @@aurora_mm_session_consistency_level;
+-----+
| @@aurora_mm_session_consistency_level |
+-----+
| INSTANCE_RAW                            |
+-----+
1 row in set (0.01 sec)

mysql> set session aurora_mm_session_consistency_level = 'REGIONAL_RAW';
Query OK, 0 rows affected (0.00 sec)

mysql> select @@aurora_mm_session_consistency_level;
+-----+
| @@aurora_mm_session_consistency_level |
+-----+
| REGIONAL_RAW                           |
+-----+
1 row in set (0.03 sec)
```

Checking DB instance read/write mode

In multi-master clusters, all nodes operate in read/write mode. The `innodb_read_only` variable always returns zero. The following example shows that when you connect to any DB instance in a multi-master cluster, the DB instance reports that it has read/write capability.

```
$ mysql -h mysql -A -h multi-master-instance-1.example123.us-east-1.rds.amazonaws.com
mysql> select @@innodb_read_only;
+-----+
| @@innodb_read_only |
+-----+
|          0         |
+-----+
mysql> quit;
Bye
```

```
$ mysql -h mysql -A -h multi-master-instance-2.example123.us-east-1.rds.amazonaws.com
mysql> select @@innodb_read_only;
+-----+
| @@innodb_read_only |
+-----+
|          0 |
+-----+
```

Checking the node name and role

You can check the name of the DB instance you're currently connected to by using the `aurora_server_id` status variable. The following example shows how.

```
mysql> select @@aurora_server_id;
+-----+
| @@aurora_server_id |
+-----+
| mmr-demo-test-mm-3-1 |
+-----+
1 row in set (0.00 sec)
```

To find this information for all the DB instances in a multi-master cluster, see [Describing cluster topology \(p. 980\)](#).

Describing cluster topology

You can describe multi-master cluster topology by selecting from the `information_schema.replica_host_status` table. Multi-master clusters have the following differences from single-master clusters:

- The `has_primary` column identifies the role of the node. For multi-master clusters, this value is true for the DB instance that handles all DDL and DCL statements. Aurora forwards such requests to one of the DB instances in a multi-master cluster.
- The `replica_lag_in_milliseconds` column reports replication lag on all DB instances.
- The `last_reported_status` column reports the status of the DB instance. It can be `Online`, `Recovery`, or `Offline`.

An example follows.

```
mysql> select server_id, has_primary, replica_lag_in_milliseconds, last_reported_status
-> from information_schema.replica_host_status;
+-----+-----+-----+-----+
| server_id | has_primary | replica_lag_in_milliseconds | last_reported_status |
+-----+-----+-----+-----+
| mmr-demo-test-mm-3-1 | true | 37.302 | Online |
| mmr-demo-test-mm-3-2 | false | 39.907 | Online |
+-----+-----+-----+-----+
```

Using instance read-only mode

In Aurora multi-master clusters, you usually issue `SELECT` statements to the specific DB instance that performs write operations on the associated tables. Doing so avoids consistency issues due to replication lag and maximizes reuse for table and index data from the buffer pool.

If you need to run a query-intensive workload across multiple tables, you might designate one or more DB instances within a multi-master cluster as read-only.

To put an entire DB instance into read-only mode at runtime, call the `mysql.rds_set_read_only` stored procedure.

```
mysql> select @@read_only;
+-----+
| @@read_only |
+-----+
|          0 |
+-----+
1 row in set (0.00 sec)
mysql> call mysql.rds_set_read_only(1);
Query OK, 0 rows affected (0.00 sec)
mysql> select @@read_only;
+-----+
| @@read_only |
+-----+
|          1 |
+-----+
1 row in set (0.00 sec)
mysql> call mysql.rds_set_read_only(0);
Query OK, 0 rows affected (0.00 sec)
mysql> select @@read_only;
+-----+
| @@read_only |
+-----+
|          0 |
+-----+
1 row in set (0.00 sec)
```

Calling the stored procedure is equivalent to running `SET GLOBAL read_only = 0|1`. That setting is runtime only and doesn't survive an engine restart. You can permanently set the DB instance to read-only by setting the `read_only` parameter to `true` in the parameter group for your DB instance.

Performance considerations for Aurora multi-master clusters

For both single-master and multi-master clusters, the Aurora engine is optimized for OLTP workloads. OLTP applications consist mostly of short-lived transactions with highly selective, random-access queries. You get the most advantage from Aurora with workloads that run many such operations concurrently.

Avoid running all the time at 100 percent utilization. Doing so lets Aurora keep up with internal maintenance work. To learn how to measure how busy a multi-master cluster is and how much maintenance work is needed, see [Monitoring an Aurora multi-master cluster \(p. 969\)](#).

Topics

- [Query performance for multi-master clusters \(p. 981\)](#)
- [Conflict resolution for multi-master clusters \(p. 982\)](#)
- [Optimizing buffer pool and dictionary cache usage \(p. 982\)](#)

Query performance for multi-master clusters

Multi-master clusters don't provide dedicated read-only nodes or read-only DNS endpoints, but it's possible to create groups of read-only DB instances and use them for the intended purpose. For more information, see [Using instance read-only mode \(p. 980\)](#).

You can use the following approaches to optimize query performance for a multi-master cluster:

- Perform `SELECT` statements on the DB instance that handles the shard containing the associated table, database, or other schema objects involved in the query. This technique maximizes reuse of data in the buffer pool. It also avoids the same data being cached on more than one DB instance. For more details about this technique, see [Optimizing buffer pool and dictionary cache usage \(p. 982\)](#).
- If you need read/write workload isolation, designate one or more DB instances as read-only, as described in [Using instance read-only mode \(p. 980\)](#). You can direct read-only sessions to those DB instances by connecting to the corresponding instance endpoints, or by defining a custom endpoint that is associated with all the read-only instances.
- Spread read-only queries across all DB instances. This approach is the least efficient. Use one of the other approaches where practical, especially as you move from the development and test phase towards production.

Conflict resolution for multi-master clusters

Many best practices for multi-master clusters focus on reducing the chance of write conflicts. Resolving write conflicts involves network overhead. Your applications must also handle error conditions and retry transactions. Wherever possible, try to minimize these unwanted consequences:

- Wherever practical, make all changes to a particular table and its associated indexes using the same DB instance. If only one DB instance ever modifies a data page, changing that page cannot trigger any write conflicts. This access pattern is common in sharded or multitenant database deployments. Thus, it's relatively easy to switch such deployments to use multi-master clusters.
- A multi-master cluster doesn't have a reader endpoint. The reader endpoint load-balances incoming connections, freeing you from knowing which DB instance is handling a particular connection. In a multi-master cluster, managing connections involves being aware which DB instance is used for each connection. That way, modifications to a particular database or table can always be routed to the same DB instance.
- A write conflict for a small amount of data (one 16-KB page) can trigger a substantial amount of work to roll back the entire transaction. Thus, ideally you keep the transactions for a multi-master cluster relatively brief and small. This best practice for OLTP applications is especially important for Aurora multi-master clusters.

Conflicts are detected at page level. A conflict could occur because proposed changes from different DB instances modify different rows within the page. All page changes introduced in the system are subject to conflict detection. This rule applies regardless of whether the source is a user transaction or a server background process. It also applies whether the data page is from a table, secondary index, undo space, and so on.

You can divide the write operations so that each DB instance handles all write operations for a set of schema objects. In this case, all the changes to each data page are made by one specific instance.

Optimizing buffer pool and dictionary cache usage

Each DB instance in a multi-master cluster maintains separate in-memory buffers and caches such as the buffer pool, table handler cache, and table dictionary cache. For each DB instance, the contents and amount of turnover for the buffers and caches depends on the SQL statements processed by that instance.

Using memory efficiently can help the performance of multi-master clusters and reduce I/O cost. Use a sharded design to physically separate the data and write to each shard from a particular DB instance. Doing so makes the most efficient use of the buffer cache on each DB instance. Try to assign `SELECT` statements for a table to the same DB instance that performs write operations for that table. Doing so helps those queries to reuse the cached data on that DB instance. If you have a large number of tables or partitions, this technique also reduces the number of unique table handlers and dictionary objects held in memory by each DB instance.

Approaches to Aurora multi-master clusters

In the following sections, you can find approaches to take for particular deployments that are suitable for multi-master clusters. These approaches involve ways to divide the workload so that the DB instances perform write operations for portions of the data that don't overlap. Doing so minimizes the chances of write conflicts. Write conflicts are the main focus of performance tuning and troubleshooting for a multi-master cluster.

Topics

- [Using a multi-master cluster for a sharded database \(p. 983\)](#)
- [Using a multi-master cluster without sharding \(p. 983\)](#)
- [Using a multi-master cluster as an active standby \(p. 984\)](#)

Using a multi-master cluster for a sharded database

Sharding is a popular type of schema design that works well with Aurora multi-master clusters. In a sharded architecture, each DB instance is assigned to update a specific group of schema objects. That way, multiple DB instances can write to the same shared storage volume without conflicts from concurrent changes. Each DB instance can handle write operations for multiple shards. You can change the mapping of DB instances to shards at any time by updating your application configuration. You don't need to reorganize your database storage or reconfigure DB instances when you do so.

Applications that use a sharded schema design are good candidates to use with Aurora multi-master clusters. The way the data is physically divided in a sharded system helps to avoid write conflicts. You map each shard to a schema object such as a partition, a table, or a database. Your application directs all write operations for a particular shard to the appropriate DB instance.

Bring-your-own-shard (BYOS) describes a use case where you already have a sharded/partitioned database and an application capable of accessing it. The shards are already physically separated. Thus, you can easily move the workload to Aurora multi-master clusters without changing your schema design. The same simple migration path applies to multitenant databases, where each tenant uses a dedicated table, a set of tables, or an entire database.

You map shards or tenants to DB instances in a one-to-one or many-to-one fashion. Each DB instance handles one or more shards. The sharded design primarily applies to write operations. You can issue `SELECT` queries for any shard from any DB instance with equivalent performance.

Suppose you used a multi-master cluster for a sharded gaming application. You might distribute the work so that database updates are performed by specific DB instances, depending on the player's user name. Your application handles the logic of mapping each player to the appropriate DB instance and connecting to the endpoint for that instance. Each DB instance can handle write operations for many different shards. You can submit queries to any DB instance, because conflicts can only arise during write operations. You might designate one DB instance to perform all `SELECT` queries to minimize the overhead on the DB instances that perform write operations.

Suppose that as time goes on, one of the shards becomes much more active. To rebalance the workload, you can switch which DB instance is responsible for that shard. In a non-Aurora system, you might have to physically move the data to a different server. With an Aurora multi-master cluster, you can reshuffle like this by directing all write operations for the shard to some other DB instance that has unused compute capacity. The Aurora shared storage model avoids the need to physically reorganize the data.

Using a multi-master cluster without sharding

If your schema design doesn't subdivide the data into physically separate containers such as databases, tables, or partitions, you can still divide write operations such as DML statements among the DB instances in a multi-master cluster.

You might see some performance overhead, and your application might have to deal with occasional transaction rollbacks when write conflicts are treated as deadlock conditions. Write conflicts are more likely during write operations for small tables. If a table contains few data pages, rows from different parts of the primary key range might be in the same data page. This overlap might lead to write conflicts if those rows are changed simultaneously by different DB instances.

You should also minimize the number of secondary indexes in this case. When you make a change to indexed columns in a table, Aurora makes corresponding changes in the associated secondary indexes. A change to an index could cause a write conflict because the order and grouping of rows is different between a secondary index and the associated table.

Because you might still experience some write conflicts when using this technique, Amazon recommends using a different approach if practical. See if you can use an alternative database design that subdivides the data into different schema objects.

Using a multi-master cluster as an active standby

An *active standby* is a DB instance that is kept synchronized with another DB instance, and is ready to take over for it very quickly. This configuration helps with high availability in situations where a single DB instance can handle the full workload.

You can use multi-master clusters in an active standby configuration by directing all traffic, both read/write and read-only, to a single DB instance. If that DB instance becomes unavailable, your application must detect the problem and switch all connections to a different DB instance. In this case, Aurora doesn't perform any failover because the other DB instance is already available to accept read/write connections. By only writing to a single DB instance at any one time, you avoid write conflicts. Thus, you don't need to have a sharded database schema to use multi-master clusters in this way.

Tip

If your application can tolerate a brief pause, you can wait several seconds after a DB instance becomes unavailable before redirecting write traffic to another instance. When an instance becomes unavailable because of a restart, it becomes available again after approximately 10–20 seconds. If the instance can't restart quickly, Aurora might initiate recovery for that instance. When an instance is shut down, it performs some additional cleanup activities as part of the shutdown. If you begin writing to a different instance while the instance is restarting, undergoing recovery, or being shut down, you can encounter write conflicts. The conflicts can occur between SQL statements on the new instance, and recovery operations such as rollback and purge on the instance that was restarted or shut down.

Integrating Amazon Aurora MySQL with other AWS services

Amazon Aurora MySQL integrates with other AWS services so that you can extend your Aurora MySQL DB cluster to use additional capabilities in the AWS Cloud. Your Aurora MySQL DB cluster can use AWS services to do the following:

- Synchronously or asynchronously invoke an AWS Lambda function using the native functions `lambda_sync` or `lambda_async`. For more information, see [Invoking a Lambda function from an Amazon Aurora MySQL DB cluster \(p. 1010\)](#).
- Load data from text or XML files stored in an Amazon Simple Storage Service (Amazon S3) bucket into your DB cluster using the `LOAD DATA FROM S3` or `LOAD XML FROM S3` command. For more information, see [Loading data into an Amazon Aurora MySQL DB cluster from text files in an Amazon S3 bucket \(p. 997\)](#).

- Save data to text files stored in an Amazon S3 bucket from your DB cluster using the `SELECT INTO OUTFILE S3` command. For more information, see [Saving data from an Amazon Aurora MySQL DB cluster into text files in an Amazon S3 bucket \(p. 1004\)](#).
- Automatically add or remove Aurora Replicas with Application Auto Scaling. For more information, see [Using Amazon Aurora Auto Scaling with Aurora replicas \(p. 427\)](#).
- Perform sentiment analysis with Amazon Comprehend, or a wide variety of machine learning algorithms with SageMaker. For more information, see [Using machine learning \(ML\) capabilities with Amazon Aurora \(p. 442\)](#).

Aurora secures the ability to access other AWS services by using AWS Identity and Access Management (IAM). You grant permission to access other AWS services by creating an IAM role with the necessary permissions, and then associating the role with your DB cluster. For details and instructions on how to permit your Aurora MySQL DB cluster to access other AWS services on your behalf, see [Authorizing Amazon Aurora MySQL to access other AWS services on your behalf \(p. 985\)](#).

Authorizing Amazon Aurora MySQL to access other AWS services on your behalf

Note

Integration with other AWS services is available for Amazon Aurora MySQL version 1.8 and later. Some integration features are only available for later versions of Aurora MySQL. For more information on Aurora versions, see [Database engine updates for Amazon Aurora MySQL \(p. 1082\)](#).

For your Aurora MySQL DB cluster to access other services on your behalf, create and configure an AWS Identity and Access Management (IAM) role. This role authorizes database users in your DB cluster to access other AWS services. For more information, see [Setting up IAM roles to access AWS services \(p. 985\)](#).

You must also configure your Aurora DB cluster to allow outbound connections to the target AWS service. For more information, see [Enabling network communication from Amazon Aurora MySQL to other AWS services \(p. 996\)](#).

If you do so, your database users can perform these actions using other AWS services:

- Synchronously or asynchronously invoke an AWS Lambda function using the native functions `lambda_sync` or `lambda_async`. Or, asynchronously invoke an AWS Lambda function using the `mysql.lambda_async` procedure. For more information, see [Invoking a Lambda function with an Aurora MySQL native function \(p. 1012\)](#).
- Load data from text or XML files stored in an Amazon S3 bucket into your DB cluster by using the `LOAD DATA FROM S3` or `LOAD XML FROM S3` statement. For more information, see [Loading data into an Amazon Aurora MySQL DB cluster from text files in an Amazon S3 bucket \(p. 997\)](#).
- Save data from your DB cluster into text files stored in an Amazon S3 bucket by using the `SELECT INTO OUTFILE S3` statement. For more information, see [Saving data from an Amazon Aurora MySQL DB cluster into text files in an Amazon S3 bucket \(p. 1004\)](#).
- Export log data to Amazon CloudWatch Logs MySQL. For more information, see [Publishing Amazon Aurora MySQL logs to Amazon CloudWatch Logs \(p. 1017\)](#).
- Automatically add or remove Aurora Replicas with Application Auto Scaling. For more information, see [Using Amazon Aurora Auto Scaling with Aurora replicas \(p. 427\)](#).

Setting up IAM roles to access AWS services

To permit your Aurora DB cluster to access another AWS service, do the following:

1. Create an IAM policy that grants permission to the AWS service. For more information, see:
 - [Creating an IAM policy to access Amazon S3 resources \(p. 986\)](#)
 - [Creating an IAM policy to access AWS Lambda resources \(p. 988\)](#)
 - [Creating an IAM policy to access CloudWatch Logs resources \(p. 989\)](#)
 - [Creating an IAM policy to access AWS KMS resources \(p. 990\)](#)
2. Create an IAM role and attach the policy that you created. For more information, see [Creating an IAM role to allow Amazon Aurora to access AWS services \(p. 991\)](#).
3. Associate that IAM role with your Aurora DB cluster. For more information, see [Associating an IAM role with an Amazon Aurora MySQL DB cluster \(p. 992\)](#).

Creating an IAM policy to access Amazon S3 resources

Aurora can access Amazon S3 resources to either load data to or save data from an Aurora DB cluster. However, you must first create an IAM policy that provides the bucket and object permissions that allow Aurora to access Amazon S3.

The following table lists the Aurora features that can access an Amazon S3 bucket on your behalf, and the minimum required bucket and object permissions required by each feature.

Feature	Bucket permissions	Object permissions
LOAD DATA FROM S3	ListBucket	GetObject GetObjectVersion
LOAD XML FROM S3	ListBucket	GetObject GetObjectVersion
SELECT INTO OUTFILE S3	ListBucket	AbortMultipartUpload DeleteObject GetObject ListMultipartUploadParts PutObject

The following policy adds the permissions that might be required by Aurora to access an Amazon S3 bucket on your behalf.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowAuroraToExampleBucket",
      "Effect": "Allow",
      "Action": [
        "s3:PutObject",
        "s3:GetObject",
        "s3:AbortMultipartUpload",
        "s3>ListBucket",
        "s3>DeleteObject",
        "s3:GetObjectVersion",
        "s3>ListMultipartUploadParts"
      ],
    }
  ]
}
```

```
        "Resource": [
            "arn:aws:s3::::example-bucket/*",
            "arn:aws:s3::::example-bucket"
        ]
    }
}
```

Note

Make sure to include both entries for the **Resource** value. Aurora needs the permissions on both the bucket itself and all the objects inside the bucket.

Based on your use case, you might not need to add all of the permissions in the sample policy. Also, other permissions might be required. For example, if your Amazon S3 bucket is encrypted, you need to add `kms:Decrypt` permissions.

You can use the following steps to create an IAM policy that provides the minimum required permissions for Aurora to access an Amazon S3 bucket on your behalf. To allow Aurora to access all of your Amazon S3 buckets, you can skip these steps and use either the `AmazonS3ReadOnlyAccess` or `AmazonS3FullAccess` predefined IAM policy instead of creating your own.

To create an IAM policy to grant access to your Amazon S3 resources

1. Open the [IAM Management Console](#).
2. In the navigation pane, choose **Policies**.
3. Choose **Create policy**.
4. On the **Visual editor** tab, choose **Choose a service**, and then choose **S3**.
5. For **Actions**, choose **Expand all**, and then choose the bucket permissions and object permissions needed for the IAM policy.

Object permissions are permissions for object operations in Amazon S3, and need to be granted for objects in a bucket, not the bucket itself. For more information about permissions for object operations in Amazon S3, see [Permissions for object operations](#).

6. Choose **Resources**, and choose **Add ARN for bucket**.
7. In the **Add ARN(s)** dialog box, provide the details about your resource, and choose **Add**.

Specify the Amazon S3 bucket to allow access to. For instance, if you want to allow Aurora to access the Amazon S3 bucket named `example-bucket`, then set the Amazon Resource Name (ARN) value to `arn:aws:s3::::example-bucket`.

8. If the **object** resource is listed, choose **Add ARN for object**.
9. In the **Add ARN(s)** dialog box, provide the details about your resource.

For the Amazon S3 bucket, specify the Amazon S3 bucket to allow access to. For the object, you can choose **Any** to grant permissions to any object in the bucket.

Note

You can set **Amazon Resource Name (ARN)** to a more specific ARN value in order to allow Aurora to access only specific files or folders in an Amazon S3 bucket. For more information about how to define an access policy for Amazon S3, see [Managing access permissions to your Amazon S3 resources](#).

10. (Optional) Choose **Add ARN for bucket** to add another Amazon S3 bucket to the policy, and repeat the previous steps for the bucket.

Note

You can repeat this to add corresponding bucket permission statements to your policy for each Amazon S3 bucket that you want Aurora to access. Optionally, you can also grant access to all buckets and objects in Amazon S3.

11. Choose **Review policy**.

12. For **Name**, enter a name for your IAM policy, for example `AllowAuroraToExampleBucket`. You use this name when you create an IAM role to associate with your Aurora DB cluster. You can also add an optional **Description** value.
13. Choose **Create policy**.
14. Complete the steps in [Creating an IAM role to allow Amazon Aurora to access AWS services \(p. 991\)](#).

Creating an IAM policy to access AWS Lambda resources

You can create an IAM policy that provides the minimum required permissions for Aurora to invoke an AWS Lambda function on your behalf.

The following policy adds the permissions required by Aurora to invoke an AWS Lambda function on your behalf.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "AllowAuroraToExampleFunction",  
            "Effect": "Allow",  
            "Action": "lambda:InvokeFunction",  
            "Resource": "  
arn:aws:lambda:<region>:<123456789012>:function:<example_function>"  
        }  
    ]  
}
```

You can use the following steps to create an IAM policy that provides the minimum required permissions for Aurora to invoke an AWS Lambda function on your behalf. To allow Aurora to invoke all of your AWS Lambda functions, you can skip these steps and use the predefined `AWSLambdaRole` policy instead of creating your own.

To create an IAM policy to grant invoke to your AWS Lambda functions

1. Open the [IAM console](#).
2. In the navigation pane, choose **Policies**.
3. Choose **Create policy**.
4. On the **Visual editor** tab, choose **Choose a service**, and then choose **Lambda**.
5. For **Actions**, choose **Expand all**, and then choose the AWS Lambda permissions needed for the IAM policy.

Ensure that `InvokeFunction` is selected. It is the minimum required permission to enable Amazon Aurora to invoke an AWS Lambda function.

6. Choose **Resources** and choose **Add ARN for function**.
7. In the **Add ARN(s)** dialog box, provide the details about your resource.

Specify the Lambda function to allow access to. For instance, if you want to allow Aurora to access a Lambda function named `example_function`, then set the ARN value to `arn:aws:lambda:::function:example_function`.

For more information on how to define an access policy for AWS Lambda, see [Authentication and access control for AWS Lambda](#).

8. Optionally, choose **Add additional permissions** to add another AWS Lambda function to the policy, and repeat the previous steps for the function.

Note

You can repeat this to add corresponding function permission statements to your policy for each AWS Lambda function that you want Aurora to access.

9. Choose **Review policy**.
10. Set **Name** to a name for your IAM policy, for example `AllowAuroraToExampleFunction`. You use this name when you create an IAM role to associate with your Aurora DB cluster. You can also add an optional **Description** value.
11. Choose **Create policy**.
12. Complete the steps in [Creating an IAM role to allow Amazon Aurora to access AWS services \(p. 991\)](#).

Creating an IAM policy to access CloudWatch Logs resources

Aurora can access CloudWatch Logs to export audit log data from an Aurora DB cluster. However, you must first create an IAM policy that provides the log group and log stream permissions that allow Aurora to access CloudWatch Logs.

The following policy adds the permissions required by Aurora to access Amazon CloudWatch Logs on your behalf, and the minimum required permissions to create log groups and export data.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "EnableCreationAndManagementOfRDSCloudwatchLogEvents",  
            "Effect": "Allow",  
            "Action": [  
                "logs:GetLogEvents",  
                "logs:PutLogEvents"  
            ],  
            "Resource": "arn:aws:logs:*:log-group:/aws/rds/*:log-stream:*"  
        },  
        {  
            "Sid": "EnableCreationAndManagementOfRDSCloudwatchLogGroupsAndStreams",  
            "Effect": "Allow",  
            "Action": [  
                "logs>CreateLogStream",  
                "logs:DescribeLogStreams",  
                "logs:PutRetentionPolicy",  
                "logs>CreateLogGroup"  
            ],  
            "Resource": "arn:aws:logs:*:log-group:/aws/rds/*"  
        }  
    ]  
}
```

You can modify the ARNs in the policy to restrict access to a specific AWS Region and account.

You can use the following steps to create an IAM policy that provides the minimum required permissions for Aurora to access CloudWatch Logs on your behalf. To allow Aurora full access to CloudWatch Logs, you can skip these steps and use the `CloudWatchLogsFullAccess` predefined IAM policy instead of creating your own. For more information, see [Using identity-based policies \(IAM policies\) for CloudWatch Logs](#) in the *Amazon CloudWatch User Guide*.

To create an IAM policy to grant access to your CloudWatch Logs resources

1. Open the [IAM console](#).

2. In the navigation pane, choose **Policies**.
3. Choose **Create policy**.
4. On the **Visual editor** tab, choose **Choose a service**, and then choose **CloudWatch Logs**.
5. For **Actions**, choose **Expand all** (on the right), and then choose the Amazon CloudWatch Logs permissions needed for the IAM policy.

Ensure that the following permissions are selected:

- `CreateLogGroup`
- `CreateLogStream`
- `DescribeLogStreams`
- `GetLogEvents`
- `PutLogEvents`
- `PutRetentionPolicy`

6. Choose **Resources** and choose **Add ARN for log-group**.
7. In the **Add ARN(s)** dialog box, enter the following values:
 - **Region** – An AWS Region or *
 - **Account** – An account number or *
 - **Log Group Name** – `/aws/rds/*`
8. In the **Add ARN(s)** dialog box, choose **Add**.
9. Choose **Add ARN for log-stream**.
10. In the **Add ARN(s)** dialog box, enter the following values:
 - **Region** – An AWS Region or *
 - **Account** – An account number or *
 - **Log Group Name** – `/aws/rds/*`
 - **Log Stream Name** – *
11. In the **Add ARN(s)** dialog box, choose **Add**.
12. Choose **Review policy**.
13. Set **Name** to a name for your IAM policy, for example `AmazonRDSCloudWatchLogs`. You use this name when you create an IAM role to associate with your Aurora DB cluster. You can also add an optional **Description** value.
14. Choose **Create policy**.
15. Complete the steps in [Creating an IAM role to allow Amazon Aurora to access AWS services \(p. 991\)](#).

Creating an IAM policy to access AWS KMS resources

Aurora can access the AWS KMS keys used for encrypting their database backups. However, you must first create an IAM policy that provides the permissions that allow Aurora to access KMS keys.

The following policy adds the permissions required by Aurora to access KMS keys on your behalf.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": [  
                "kms:Decrypt"  
            ]  
        }  
    ]  
}
```

```
        ],
      "Resource": "arn:aws:kms:<region>:<123456789012>:key/<key-ID>"  
    }  
}
```

You can use the following steps to create an IAM policy that provides the minimum required permissions for Aurora to access KMS keys on your behalf.

To create an IAM policy to grant access to your KMS keys

1. Open the [IAM console](#).
2. In the navigation pane, choose **Policies**.
3. Choose **Create policy**.
4. On the **Visual editor** tab, choose **Choose a service**, and then choose **KMS**.
5. For **Actions**, choose **Write**, and then choose **Decrypt**.
6. Choose **Resources**, and choose **Add ARN**.
7. In the **Add ARN(s)** dialog box, enter the following values:
 - **Region** – Type the AWS Region, such as `us-west-2`.
 - **Account** – Type the user account number.
 - **Log Stream Name** – Type the KMS key identifier.
8. In the **Add ARN(s)** dialog box, choose **Add**
9. Choose **Review policy**.
10. Set **Name** to a name for your IAM policy, for example `AmazonRDSKMSKey`. You use this name when you create an IAM role to associate with your Aurora DB cluster. You can also add an optional **Description** value.
11. Choose **Create policy**.
12. Complete the steps in [Creating an IAM role to allow Amazon Aurora to access AWS services \(p. 991\)](#).

Creating an IAM role to allow Amazon Aurora to access AWS services

After creating an IAM policy to allow Aurora to access AWS resources, you must create an IAM role and attach the IAM policy to the new IAM role.

To create an IAM role to permit your Amazon RDS cluster to communicate with other AWS services on your behalf, take the following steps.

To create an IAM role to allow Amazon RDS to access AWS services

1. Open the [IAM console](#).
2. In the navigation pane, choose **Roles**.
3. Choose **Create role**.
4. Under **AWS service**, choose **RDS**.
5. Under **Select your use case**, choose **RDS – Add Role to Database**.
6. Choose **Next: Permissions**.
7. On the **Attach permissions policies** page, enter the name of your policy in the **Search** field.
8. When it appears in the list, select the policy that you defined earlier using the instructions in one of the following sections:

- [Creating an IAM policy to access Amazon S3 resources \(p. 986\)](#)
 - [Creating an IAM policy to access AWS Lambda resources \(p. 988\)](#)
 - [Creating an IAM policy to access CloudWatch Logs resources \(p. 989\)](#)
 - [Creating an IAM policy to access AWS KMS resources \(p. 990\)](#)
9. Choose **Next: Tags**, and then choose **Next: Review**.
 10. In **Role name**, enter a name for your IAM role, for example `RDSLoadFromS3`. You can also add an optional **Role description** value.
 11. Choose **Create Role**.
 12. Complete the steps in [Associating an IAM role with an Amazon Aurora MySQL DB cluster \(p. 992\)](#).

Associating an IAM role with an Amazon Aurora MySQL DB cluster

To permit database users in an Amazon Aurora DB cluster to access other AWS services, you associate the role that you created in [Creating an IAM role to allow Amazon Aurora to access AWS services \(p. 991\)](#) with that DB cluster.

Note

You can't associate an IAM role with an Aurora Serverless DB cluster. For more information, see [Using Amazon Aurora Serverless v1 \(p. 147\)](#).

To associate an IAM role with a DB cluster you do two things:

- Add the role to the list of associated roles for a DB cluster by using the RDS console, the `add-role-to-db-cluster` AWS CLI command, or the `AddRoleToDBCluster` RDS API operation.

You can add a maximum of five IAM roles for each Aurora DB cluster.
- Set the cluster-level parameter for the related AWS service to the ARN for the associated IAM role.

The following table describes the cluster-level parameter names for the IAM roles used to access other AWS services.

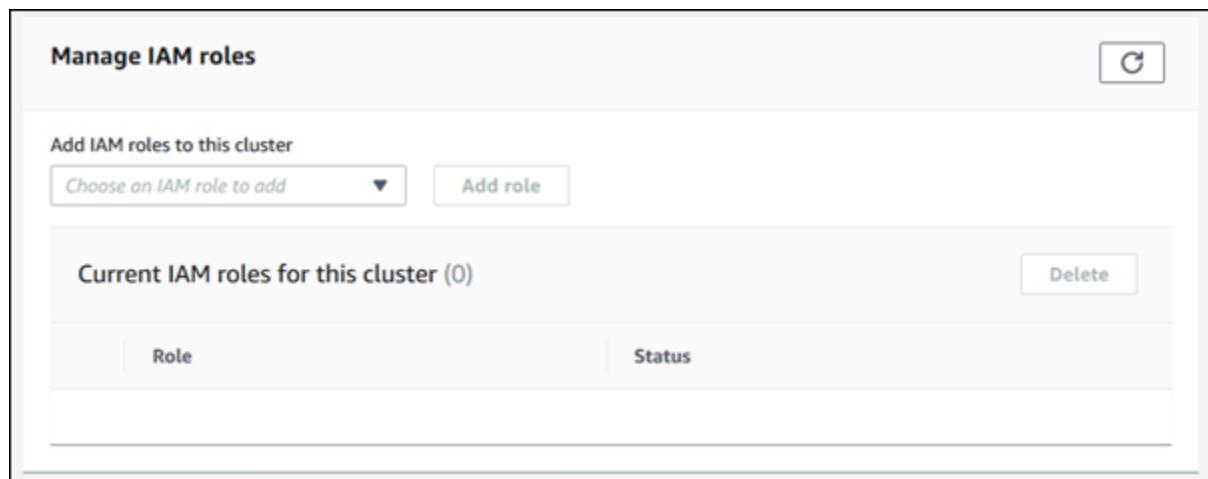
Cluster-level parameter	Description
<code>aws_default_lambda_role</code>	Used when invoking a Lambda function from your DB cluster.
<code>aws_default_logs_role</code>	This parameter is no longer required for exporting log data from your DB cluster to Amazon CloudWatch Logs. Aurora MySQL now uses a service-linked role for the required permissions. For more information about service-linked roles, see Using service-linked roles for Amazon Aurora (p. 1783) .
<code>aws_default_s3_role</code>	Used when invoking the <code>LOAD DATA FROM S3</code> , <code>LOAD XML FROM S3</code> , or <code>SELECT INTO OUTFILE S3</code> statement from your DB cluster. <p>The IAM role specified in this parameter is used only if an IAM role isn't specified for <code>aurora_load_from_s3_role</code> or <code>aurora_select_into_s3_role</code> for the appropriate statement.</p>

Cluster-level parameter	Description	
	For earlier versions of Aurora, the IAM role specified for this parameter is always used.	
aurora_load_from_s3	<p><small>Used when invoking the LOAD DATA FROM S3 or LOAD XML FROM S3 statement from your DB cluster. If an IAM role is not specified for this parameter, the IAM role specified in aws_default_s3_role is used.</small></p> <p>For earlier versions of Aurora, this parameter is not available.</p>	
aurora_select_into_s3	<p><small>Used when invoking the SELECT INTO OUTFILE S3 statement from your DB cluster. If an IAM role is not specified for this parameter, the IAM role specified in aws_default_s3_role is used.</small></p> <p>For earlier versions of Aurora, this parameter is not available.</p>	

To associate an IAM role to permit your Amazon RDS cluster to communicate with other AWS services on your behalf, take the following steps.

To associate an IAM role with an Aurora DB cluster using the console

1. Open the RDS console at <https://console.aws.amazon.com/rds/>.
2. Choose **Databases**.
3. Choose the name of the Aurora DB cluster that you want to associate an IAM role with to show its details.
4. On the **Connectivity & security** tab, in the **Manage IAM roles** section, choose the role to add under **Add IAM roles to this cluster**.



5. Choose **Add role**.
6. (Optional) To stop associating an IAM role with a DB cluster and remove the related permission, choose the role and choose **Delete**.

7. In the RDS console, choose **Parameter groups** in the navigation pane.
8. If you are already using a custom DB parameter group, you can select that group to use instead of creating a new DB cluster parameter group. If you are using the default DB cluster parameter group, create a new DB cluster parameter group, as described in the following steps:
 - a. Choose **Create parameter group**.
 - b. For **Parameter group family**, choose `aurora-mysql8.0` for an Aurora MySQL 8.0-compatible DB cluster, choose `aurora-mysql5.7` for an Aurora MySQL 5.7-compatible DB cluster, or choose `aurora5.6` for an Aurora MySQL 5.6-compatible DB cluster.
 - c. For **Type**, choose **DB Cluster Parameter Group**.
 - d. For **Group name**, type the name of your new DB cluster parameter group.
 - e. For **Description**, type a description for your new DB cluster parameter group.

The screenshot shows the 'Create parameter group' dialog box. The 'Parameter group details' section contains the following fields:

- Parameter group family:** aurora5.6
- Type:** DB Cluster Parameter Group
- Group name:** AllowAWSAccess
- Description:** Allow access to S3

At the bottom right of the dialog box are two buttons: 'Cancel' and 'Create'.

- f. Choose **Create**.
9. On the **Parameter groups** page, select your DB cluster parameter group and choose **Edit** for **Parameter group actions**.
10. Set the appropriate cluster-level parameters to the related IAM role ARN values. For example, you can set just the `aws_default_s3_role` parameter to `arn:aws:iam::123456789012:role/AllowAuroraS3Role`.
11. Choose **Save changes**.
12. To change the DB cluster parameter group for your DB cluster, complete the following steps:
 - a. Choose **Databases**, and then choose your Aurora DB cluster.
 - b. Choose **Modify**.
 - c. Scroll to **Database options** and set **DB cluster parameter group** to the DB cluster parameter group.
 - d. Choose **Continue**.
 - e. Verify your changes and then choose **Apply immediately**.
 - f. Choose **Modify cluster**.

- g. Choose **Databases**, and then choose the primary instance for your DB cluster.
- h. For **Actions**, choose **Reboot**.

When the instance has rebooted, your IAM role is associated with your DB cluster.

For more information about cluster parameter groups, see [Aurora MySQL configuration parameters \(p. 1042\)](#).

To associate an IAM role with a DB cluster by using the AWS CLI

1. Call the `add-role-to-db-cluster` command from the AWS CLI to add the ARNs for your IAM roles to the DB cluster, as shown following.

```
PROMPT> aws rds add-role-to-db-cluster --db-cluster-identifier my-cluster --role-arn arn:aws:iam::123456789012:role/AllowAuroraS3Role
PROMPT> aws rds add-role-to-db-cluster --db-cluster-identifier my-cluster --role-arn arn:aws:iam::123456789012:role/AllowAuroraLambdaRole
```

2. If you are using the default DB cluster parameter group, create a new DB cluster parameter group. If you are already using a custom DB parameter group, you can use that group instead of creating a new DB cluster parameter group.

To create a new DB cluster parameter group, call the `create-db-cluster-parameter-group` command from the AWS CLI, as shown following.

```
PROMPT> aws rds create-db-cluster-parameter-group --db-cluster-parameter-group-name AllowAWSAccess \
    --db-parameter-group-family aurora5.6 --description "Allow access to Amazon S3 and
AWS Lambda"
```

For an Aurora MySQL 5.7-compatible DB cluster, specify `aurora-mysql5.7` for `--db-parameter-group-family`. For an Aurora MySQL 8.0-compatible DB cluster, specify `aurora-mysql8.0` for `--db-parameter-group-family`.

3. Set the appropriate cluster-level parameter or parameters and the related IAM role ARN values in your DB cluster parameter group, as shown following.

```
PROMPT> aws rds modify-db-cluster-parameter-group --db-cluster-parameter-group-name
    AllowAWSAccess \
    --parameters
    "ParameterName=aws_default_s3_role,ParameterValue=arn:aws:iam::123456789012:role/
    AllowAuroraS3Role,method=pending-reboot" \
    --parameters
    "ParameterName=aws_default_lambda_role,ParameterValue=arn:aws:iam::123456789012:role/
    AllowAuroraLambdaRole,method=pending-reboot"
```

4. Modify the DB cluster to use the new DB cluster parameter group and then reboot the cluster, as shown following.

```
PROMPT> aws rds modify-db-cluster --db-cluster-identifier my-cluster --db-cluster-
parameter-group-name AllowAWSAccess
PROMPT> aws rds reboot-db-instance --db-instance-identifier my-cluster-primary
```

When the instance has rebooted, your IAM roles are associated with your DB cluster.

For more information about cluster parameter groups, see [Aurora MySQL configuration parameters \(p. 1042\)](#).

Enabling network communication from Amazon Aurora MySQL to other AWS services

To use certain other AWS services with Amazon Aurora, the network configuration of your Aurora DB cluster must allow outbound connections to endpoints for those services. The following operations require this network configuration.

- Invoking AWS Lambda functions. To learn about this feature, see [Invoking a Lambda function with an Aurora MySQL native function \(p. 1012\)](#).
- Accessing files from Amazon S3. To learn about this feature, see [Loading data into an Amazon Aurora MySQL DB cluster from text files in an Amazon S3 bucket \(p. 997\)](#) and [Saving data from an Amazon Aurora MySQL DB cluster into text files in an Amazon S3 bucket \(p. 1004\)](#).
- Accessing AWS KMS endpoints. AWS KMS access is required to use database activity streams with Aurora MySQL. To learn about this feature, see [Monitoring Amazon Aurora with Database Activity Streams \(p. 714\)](#).
- Accessing SageMaker endpoints. SageMaker access is required to use SageMaker machine learning with Aurora MySQL. To learn about this feature, see [Using machine learning \(ML\) with Aurora MySQL \(p. 1020\)](#).

Aurora returns the following error messages if it can't connect to a service endpoint.

```
ERROR 1871 (HY000): S3 API returned error: Network Connection
```

```
ERROR 1873 (HY000): Lambda API returned error: Network Connection. Unable to connect to endpoint
```

```
ERROR 1815 (HY000): Internal error: Unable to initialize S3Stream
```

For database activity streams using Aurora MySQL, the activity stream stops functioning if the DB cluster can't access the AWS KMS endpoint. Aurora notifies you about this issue using RDS Events.

If you encounter these messages while using the corresponding AWS services, check if your Aurora DB cluster is public or private. If your Aurora DB cluster is private, you must configure it to enable connections.

For an Aurora DB cluster to be public, it must be marked as publicly accessible. If you look at the details for the DB cluster in the AWS Management Console, **Publicly Accessible** is **Yes** if this is the case. The DB cluster must also be in an Amazon VPC public subnet. For more information about publicly accessible DB instances, see [Working with a DB instance in a VPC \(p. 1787\)](#). For more information about public Amazon VPC subnets, see [Your VPC and subnets](#).

If your Aurora DB cluster isn't publicly accessible and in a VPC public subnet, it is private. You might have a DB cluster that is private and want to use one of the features that requires this network configuration. If so, configure the cluster so that it can connect to Internet addresses through Network Address Translation (NAT). As an alternative for Amazon S3, Amazon SageMaker, and AWS Lambda, you can instead configure the VPC to have a VPC endpoint for the other service associated with the DB cluster's route table. For more information about configuring NAT in your VPC, see [NAT gateways](#). For more information about configuring VPC endpoints, see [VPC endpoints](#).

Related topics

- [Integrating Aurora with other AWS services \(p. 426\)](#)

- Managing an Amazon Aurora DB cluster (p. 367)

Loading data into an Amazon Aurora MySQL DB cluster from text files in an Amazon S3 bucket

You can use the `LOAD DATA FROM S3` or `LOAD XML FROM S3` statement to load data from files stored in an Amazon S3 bucket.

If you are using encryption, the Amazon S3 bucket must be encrypted with an AWS managed key. Currently, you can't load data from a bucket that is encrypted with a customer managed key.

Note

Loading data into a table from text files in an Amazon S3 bucket is available for Amazon Aurora MySQL version 1.8 and later. For more information about Aurora MySQL versions, see [Database engine updates for Amazon Aurora MySQL \(p. 1082\)](#).

This feature currently isn't available for Aurora Serverless clusters.

Giving Aurora access to Amazon S3

Before you can load data from an Amazon S3 bucket, you must first give your Aurora MySQL DB cluster permission to access Amazon S3.

To give Aurora MySQL access to Amazon S3

1. Create an AWS Identity and Access Management (IAM) policy that provides the bucket and object permissions that allow your Aurora MySQL DB cluster to access Amazon S3. For instructions, see [Creating an IAM policy to access Amazon S3 resources \(p. 986\)](#).
2. Create an IAM role, and attach the IAM policy you created in [Creating an IAM policy to access Amazon S3 resources \(p. 986\)](#) to the new IAM role. For instructions, see [Creating an IAM role to allow Amazon Aurora to access AWS services \(p. 991\)](#).
3. Make sure the DB cluster is using a custom DB cluster parameter group.

For more information about creating a custom DB cluster parameter group, see [Creating a DB cluster parameter group \(p. 343\)](#).

4. Set either the `aurora_load_from_s3_role` or `aws_default_s3_role` DB cluster parameter to the Amazon Resource Name (ARN) of the new IAM role. If an IAM role isn't specified for `aurora_load_from_s3_role`, Aurora uses the IAM role specified in `aws_default_s3_role`.

If the cluster is part of an Aurora global database, set this parameter for each Aurora cluster in the global database. Although only the primary cluster in an Aurora global database can load data, another cluster might be promoted by the failover mechanism and become the primary cluster.

For more information about DB cluster parameters, see [Amazon Aurora DB cluster and DB instance parameters \(p. 341\)](#).

5. To permit database users in an Aurora MySQL DB cluster to access Amazon S3, associate the role that you created in [Creating an IAM role to allow Amazon Aurora to access AWS services \(p. 991\)](#) with the DB cluster. For an Aurora global database, associate the role with each Aurora cluster in the global database. For information about associating an IAM role with a DB cluster, see [Associating an IAM role with an Amazon Aurora MySQL DB cluster \(p. 992\)](#).
6. Configure your Aurora MySQL DB cluster to allow outbound connections to Amazon S3. For instructions, see [Enabling network communication from Amazon Aurora MySQL to other AWS services \(p. 996\)](#).

For an Aurora global database, enable outbound connections for each Aurora cluster in the global database.

Granting privileges to load data in Amazon Aurora MySQL

The database user that issues the `LOAD DATA FROM S3` or `LOAD XML FROM S3` statement must have a specific role or privilege to issue either statement. In Aurora MySQL version 3, you grant the `AWS_LOAD_S3_ACCESS` role. In Aurora MySQL version 1 or 2, you grant the `LOAD FROM S3` privilege. The administrative user for a DB cluster is granted the appropriate role or privilege by default. You can grant the privilege to another user by using one of the following statements.

Use the following statement for Aurora MySQL version 3:

```
GRANT AWS_LOAD_S3_ACCESS TO 'user'@'domain-or-ip-address'
```

Tip

When you use the role technique in Aurora MySQL version 3, you also activate the role by using the `SET ROLE role_name` or `SET ROLE ALL` statement. If you aren't familiar with the MySQL 8.0 role system, you can learn more in [Role-based privilege model \(p. 757\)](#). You can also find more details in [Using Roles in the MySQL Reference Manual](#).

Use the following statement for Aurora MySQL version 1 or 2:

```
GRANT LOAD FROM S3 ON *.* TO 'user'@'domain-or-ip-address'
```

The `AWS_LOAD_S3_ACCESS` role and `LOAD FROM S3` privilege are specific to Amazon Aurora and are not available for MySQL databases or RDS for MySQL DB instances. If you have set up replication between an Aurora DB cluster as the replication master and a MySQL database as the replication client, then the `GRANT` statement for the role or privilege causes replication to stop with an error. You can safely skip the error to resume replication. To skip the error on an RDS for MySQL DB instance, use the `mysql_rds_skip_repl_error` procedure. To skip the error on an external MySQL database, use the `SET GLOBAL sql_slave_skip_counter` statement (Aurora MySQL version 1 and 2) or `SET GLOBAL sql_replica_skip_counter` statement (Aurora MySQL version 3).

Specifying a path to an Amazon S3 bucket

The syntax for specifying a path to files stored on an Amazon S3 bucket is as follows.

```
s3-region://bucket-name/file-name-or-prefix
```

The path includes the following values:

- `region` (optional) – The AWS Region that contains the Amazon S3 bucket to load from. This value is optional. If you don't specify a `region` value, then Aurora loads your file from Amazon S3 in the same region as your DB cluster.
- `bucket-name` – The name of the Amazon S3 bucket that contains the data to load. Object prefixes that identify a virtual folder path are supported.
- `file-name-or-prefix` – The name of the Amazon S3 text file or XML file, or a prefix that identifies one or more text or XML files to load. You can also specify a manifest file that identifies one or more text files to load. For more information about using a manifest file to load text files from Amazon S3, see [Using a manifest to specify data files to load \(p. 1000\)](#).

LOAD DATA FROM S3

You can use the `LOAD DATA FROM S3` statement to load data from any text file format that is supported by the MySQL `LOAD DATA INFILE` statement, such as text data that is comma-delimited. Compressed files are not supported.

Syntax

```
LOAD DATA FROM S3 [FILE | PREFIX | MANIFEST] 'S3-URI'  
    [REPLACE | IGNORE]  
    INTO TABLE tbl_name  
    [PARTITION (partition_name,...)]  
    [CHARACTER SET charset_name]  
    [{FIELDS | COLUMNS}  
        [TERMINATED BY 'string']  
        [[OPTIONALLY] ENCLOSED BY 'char']  
        [ESCAPED BY 'char']  
    ]  
    [LINES  
        [STARTING BY 'string']  
        [TERMINATED BY 'string']  
    ]  
    [IGNORE number {LINES | ROWS}]  
    [(col_name_or_user_var,...)]  
    [SET col_name = expr,...]
```

Parameters

Following, you can find a list of the required and optional parameters used by the `LOAD DATA FROM S3` statement. You can find more details about some of these parameters in [LOAD DATA INFILE syntax](#) in the MySQL documentation.

- **FILE | PREFIX | MANIFEST** – Identifies whether to load the data from a single file, from all files that match a given prefix, or from all files in a specified manifest. `FILE` is the default.
- **S3-URI** – Specifies the URI for a text or manifest file to load, or an Amazon S3 prefix to use. Specify the URI using the syntax described in [Specifying a path to an Amazon S3 bucket \(p. 998\)](#).
- **REPLACE | IGNORE** – Determines what action to take if an input row as the same unique key values as an existing row in the database table.
 - Specify `REPLACE` if you want the input row to replace the existing row in the table.
 - Specify `IGNORE` if you want to discard the input row.
- **INTO TABLE** – Identifies the name of the database table to load the input rows into.
- **PARTITION** – Requires that all input rows be inserted into the partitions identified by the specified list of comma-separated partition names. If an input row cannot be inserted into one of the specified partitions, then the statement fails and an error is returned.
- **CHARACTER SET** – Identifies the character set of the data in the input file.
- **FIELDS | COLUMNS** – Identifies how the fields or columns in the input file are delimited. Fields are tab-delimited by default.
- **LINES** – Identifies how the lines in the input file are delimited. Lines are delimited by a newline character ('\n') by default.
- **IGNORE *number* LINES | ROWS** – Specifies to ignore a certain number of lines or rows at the start of the input file. For example, you can use `IGNORE 1 LINES` to skip over an initial header line containing column names, or `IGNORE 2 ROWS` to skip over the first two rows of data in the input file. If you also use `PREFIX`, `IGNORE` skips a certain number of lines or rows at the start of the first input file.
- ***col_name_or_user_var*, ...** – Specifies a comma-separated list of one or more column names or user variables that identify which columns to load by name. The name of a user variable used for this purpose must match the name of an element from the text file, prefixed with `@`. You can employ user variables to store the corresponding field values for subsequent reuse.

For example, the following statement loads the first column from the input file into the first column of `table1`, and sets the value of the `table_column2` column in `table1` to the input value of the second column divided by 100.

```
LOAD DATA FROM S3 's3://mybucket/data.txt'
    INTO TABLE table1
    (column1, @var1)
    SET table_column2 = @var1/100;
```

- **SET** – Specifies a comma-separated list of assignment operations that set the values of columns in the table to values not included in the input file.

For example, the following statement sets the first two columns of `table1` to the values in the first two columns from the input file, and then sets the value of the `column3` in `table1` to the current time stamp.

```
LOAD DATA FROM S3 's3://mybucket/data.txt'
    INTO TABLE table1
    (column1, column2)
    SET column3 = CURRENT_TIMESTAMP;
```

You can use subqueries in the right side of `SET` assignments. For a subquery that returns a value to be assigned to a column, you can use only a scalar subquery. Also, you cannot use a subquery to select from the table that is being loaded.

You cannot use the `LOCAL` keyword of the `LOAD DATA FROM S3` statement if you are loading data from an Amazon S3 bucket.

Using a manifest to specify data files to load

You can use the `LOAD DATA FROM S3` statement with the `MANIFEST` keyword to specify a manifest file in JSON format that lists the text files to be loaded into a table in your DB cluster. You must be using Aurora 1.11 or greater to use the `MANIFEST` keyword with the `LOAD DATA FROM S3` statement.

The following JSON schema describes the format and content of a manifest file.

```
{
    "$schema": "http://json-schema.org/draft-04/schema#",
    "additionalProperties": false,
    "definitions": {},
    "id": "Aurora_LoadFromS3_Manifest",
    "properties": {
        "entries": {
            "additionalItems": false,
            "id": "/properties/entries",
            "items": {
                "additionalProperties": false,
                "id": "/properties/entries/items",
                "properties": {
                    "mandatory": {
                        "default": "false"
                        "id": "/properties/entries/items/properties/mandatory",
                        "type": "boolean"
                    },
                    "url": {
                        "id": "/properties/entries/items/properties/url",
                        "maxLength": 1024,
                        "minLength": 1,
                        "type": "string"
                    }
                },
                "required": [
                    "url"
                ]
            }
        }
    }
}
```

```

        ],
        "type": "object"
    },
    "type": "array",
    "uniqueItems": true
}
},
"required": [
    "entries"
],
"type": "object"
}

```

Each `url` in the manifest must specify a URL with the bucket name and full object path for the file, not just a prefix. You can use a manifest to load files from different buckets, different regions, or files that do not share the same prefix. If a region is not specified in the URL, the region of the target Aurora DB cluster is used. The following example shows a manifest file that loads four files from different buckets.

```

{
  "entries": [
    {
      "url": "s3://aurora-bucket/2013-10-04-customerdata",
      "mandatory": true
    },
    {
      "url": "s3-us-west-2://aurora-bucket-usw2/2013-10-05-customerdata",
      "mandatory": true
    },
    {
      "url": "s3://aurora-bucket/2013-10-04-customerdata",
      "mandatory": false
    },
    {
      "url": "s3://aurora-bucket/2013-10-05-customerdata"
    }
  ]
}

```

The optional `mandatory` flag specifies whether `LOAD DATA FROM S3` should return an error if the file is not found. The `mandatory` flag defaults to `false`. Regardless of how `mandatory` is set, `LOAD DATA FROM S3` terminates if no files are found.

Manifest files can have any extension. The following example runs the `LOAD DATA FROM S3` statement with the manifest in the previous example, which is named **`customer.manifest`**.

```

LOAD DATA FROM S3 MANIFEST 's3-us-west-2://aurora-bucket/customer.manifest'
INTO TABLE CUSTOMER
FIELDS TERMINATED BY ','
LINES TERMINATED BY '\n'
(ID, FIRSTNAME, LASTNAME, EMAIL);

```

After the statement completes, an entry for each successfully loaded file is written to the `aurora_s3_load_history` table.

Verifying loaded files using the `aurora_s3_load_history` table

Every successful `LOAD DATA FROM S3` statement updates the `aurora_s3_load_history` table in the `mysql` schema with an entry for each file that was loaded.

After you run the `LOAD DATA FROM S3` statement, you can verify which files were loaded by querying the `aurora_s3_load_history` table. To see the files that were loaded from one iteration of the

statement, use the `WHERE` clause to filter the records on the Amazon S3 URI for the manifest file used in the statement. If you have used the same manifest file before, filter the results using the `timestamp` field.

```
select * from mysql.aurora_s3_load_history where load_prefix = 'S3_URI';
```

The following table describes the fields in the `aurora_s3_load_history` table.

Field	Description
<code>load_prefix</code>	The URI that was specified in the load statement. This URI can map to any of the following: <ul style="list-style-type: none"> A single data file for a <code>LOAD DATA FROM S3 FILE</code> statement An Amazon S3 prefix that maps to multiple data files for a <code>LOAD DATA FROM S3 PREFIX</code> statement A single manifest file that contains the names of files to be loaded for a <code>LOAD DATA FROM S3 MANIFEST</code> statement
<code>file_name</code>	The name of a file that was loaded into Aurora from Amazon S3 using the URI identified in the <code>load_prefix</code> field.
<code>version_number</code>	The version number of the file identified by the <code>file_name</code> field that was loaded, if the Amazon S3 bucket has a version number.
<code>bytes_loaded</code>	The size of the file loaded, in bytes.
<code>load_timestamp</code>	The timestamp when the <code>LOAD DATA FROM S3</code> statement completed.

Examples

The following statement loads data from an Amazon S3 bucket that is in the same region as the Aurora DB cluster. The statement reads the comma-delimited data in the file `customerdata.txt` that is in the `dbbucket` Amazon S3 bucket, and then loads the data into the table `store-schema.customer-table`.

```
LOAD DATA FROM S3 's3://dbbucket/customerdata.csv'
  INTO TABLE store-schema.customer-table
  FIELDS TERMINATED BY ','
  LINES TERMINATED BY '\n'
  (ID, FIRSTNAME, LASTNAME, ADDRESS, EMAIL, PHONE);
```

The following statement loads data from an Amazon S3 bucket that is in a different region from the Aurora DB cluster. The statement reads the comma-delimited data from all files that match the `employee-data` object prefix in the `my-data` Amazon S3 bucket in the `us-west-2` region, and then loads the data into the `employees` table.

```
LOAD DATA FROM S3 PREFIX 's3-us-west-2://my-data/employee_data'
  INTO TABLE employees
  FIELDS TERMINATED BY ','
  LINES TERMINATED BY '\n'
  (ID, FIRSTNAME, LASTNAME, EMAIL, SALARY);
```

The following statement loads data from the files specified in a JSON manifest file named `q1_sales.json` into the `sales` table.

```
LOAD DATA FROM S3 MANIFEST 's3-us-west-2://aurora-bucket/q1_sales.json'  
INTO TABLE sales  
FIELDS TERMINATED BY ','  
LINES TERMINATED BY '\n'  
(MONTH, STORE, GROSS, NET);
```

LOAD XML FROM S3

You can use the `LOAD XML FROM S3` statement to load data from XML files stored on an Amazon S3 bucket in one of three different XML formats:

- Column names as attributes of a `<row>` element. The attribute value identifies the contents of the table field.

```
<row column1="value1" column2="value2" .../>
```

- Column names as child elements of a `<row>` element. The value of the child element identifies the contents of the table field.

```
<row>  
  <column1>value1</column1>  
  <column2>value2</column2>  
</row>
```

- Column names in the `name` attribute of `<field>` elements in a `<row>` element. The value of the `<field>` element identifies the contents of the table field.

```
<row>  
  <field name='column1'>value1</field>  
  <field name='column2'>value2</field>  
</row>
```

Syntax

```
LOAD XML FROM S3 'S3-URI'  
[REPLACE | IGNORE]  
INTO TABLE tbl_name  
[PARTITION (partition_name,...)]  
[CHARACTER SET charset_name]  
[ROWS IDENTIFIED BY '<element-name>']  
[IGNORE number {LINES | ROWS}]  
[(field_name_or_user_var,...)]  
[SET col_name = expr,...]
```

Parameters

Following, you can find a list of the required and optional parameters used by the `LOAD DATA FROM S3` statement. You can find more details about some of these parameters in [LOAD XML syntax](#) in the MySQL documentation.

- **FILE | PREFIX** – Identifies whether to load the data from a single file, or from all files that match a given prefix. `FILE` is the default.
- **REPLACE | IGNORE** – Determines what action to take if an input row has the same unique key values as an existing row in the database table.
 - Specify `REPLACE` if you want the input row to replace the existing row in the table.

- Specify `IGNORE` if you want to discard the input row. `IGNORE` is the default.
- **INTO TABLE** – Identifies the name of the database table to load the input rows into.
- **PARTITION** – Requires that all input rows be inserted into the partitions identified by the specified list of comma-separated partition names. If an input row cannot be inserted into one of the specified partitions, then the statement fails and an error is returned.
- **CHARACTER SET** – Identifies the character set of the data in the input file.
- **ROWS IDENTIFIED BY** – Identifies the element name that identifies a row in the input file. The default is `<row>`.
- **IGNORE *number* LINES | ROWS** – Specifies to ignore a certain number of lines or rows at the start of the input file. For example, you can use `IGNORE 1 LINES` to skip over the first line in the text file, or `IGNORE 2 ROWS` to skip over the first two rows of data in the input XML.
- **field_name_or_user_var, ...** – Specifies a comma-separated list of one or more XML element names or user variables that identify which elements to load by name. The name of a user variable used for this purpose must match the name of an element from the XML file, prefixed with `@`. You can employ user variables to store the corresponding field values for subsequent reuse.

For example, the following statement loads the first column from the input file into the first column of `table1`, and sets the value of the `table_column2` column in `table1` to the input value of the second column divided by 100.

```
LOAD XML FROM S3 's3://mybucket/data.xml'  
    INTO TABLE table1  
        (column1, @var1)  
    SET table_column2 = @var1/100;
```

- **SET** – Specifies a comma-separated list of assignment operations that set the values of columns in the table to values not included in the input file.

For example, the following statement sets the first two columns of `table1` to the values in the first two columns from the input file, and then sets the value of the `column3` in `table1` to the current time stamp.

```
LOAD XML FROM S3 's3://mybucket/data.xml'  
    INTO TABLE table1  
        (column1, column2)  
    SET column3 = CURRENT_TIMESTAMP;
```

You can use subqueries in the right side of `SET` assignments. For a subquery that returns a value to be assigned to a column, you can use only a scalar subquery. Also, you cannot use a subquery to select from the table that is being loaded.

Related topics

- [Integrating Amazon Aurora MySQL with other AWS services \(p. 984\)](#)
- [Saving data from an Amazon Aurora MySQL DB cluster into text files in an Amazon S3 bucket \(p. 1004\)](#)
- [Managing an Amazon Aurora DB cluster \(p. 367\)](#)
- [Migrating data to an Amazon Aurora DB cluster \(p. 366\)](#)

Saving data from an Amazon Aurora MySQL DB cluster into text files in an Amazon S3 bucket

You can use the `SELECT INTO OUTFILE S3` statement to query data from an Amazon Aurora MySQL DB cluster and save it directly into text files stored in an Amazon S3 bucket. You can use this functionality to skip bringing the data down to the client first, and then copying it from the client to Amazon S3. The `LOAD DATA FROM S3` statement can use the files created by this statement to load data into an Aurora DB cluster. For more information, see [Loading data into an Amazon Aurora MySQL DB cluster from text files in an Amazon S3 bucket \(p. 997\)](#).

If you are using encryption, the Amazon S3 bucket must be encrypted with an AWS managed key. Currently, you can't save data to a bucket that is encrypted with a customer managed key.

This feature currently isn't available for Aurora Serverless clusters.

Note

You can save DB cluster snapshot data to Amazon S3 using the AWS Management Console, AWS CLI, or Amazon RDS API. For more information, see [Exporting DB snapshot data to Amazon S3 \(p. 518\)](#).

Giving Aurora MySQL access to Amazon S3

Before you can save data into an Amazon S3 bucket, you must first give your Aurora MySQL DB cluster permission to access Amazon S3.

To give Aurora MySQL access to Amazon S3

1. Create an AWS Identity and Access Management (IAM) policy that provides the bucket and object permissions that allow your Aurora MySQL DB cluster to access Amazon S3. For instructions, see [Creating an IAM policy to access Amazon S3 resources \(p. 986\)](#).
2. Create an IAM role, and attach the IAM policy you created in [Creating an IAM policy to access Amazon S3 resources \(p. 986\)](#) to the new IAM role. For instructions, see [Creating an IAM role to allow Amazon Aurora to access AWS services \(p. 991\)](#).
3. Set either the `aurora_select_into_s3_role` or `aws_default_s3_role` DB cluster parameter to the Amazon Resource Name (ARN) of the new IAM role. If an IAM role isn't specified for `aurora_select_into_s3_role`, Aurora uses the IAM role specified in `aws_default_s3_role`.

If the cluster is part of an Aurora global database, set this parameter for each Aurora cluster in the global database.

For more information about DB cluster parameters, see [Amazon Aurora DB cluster and DB instance parameters \(p. 341\)](#).

4. To permit database users in an Aurora MySQL DB cluster to access Amazon S3, associate the role that you created in [Creating an IAM role to allow Amazon Aurora to access AWS services \(p. 991\)](#) with the DB cluster.

For an Aurora global database, associate the role with each Aurora cluster in the global database.

For information about associating an IAM role with a DB cluster, see [Associating an IAM role with an Amazon Aurora MySQL DB cluster \(p. 992\)](#).

5. Configure your Aurora MySQL DB cluster to allow outbound connections to Amazon S3. For instructions, see [Enabling network communication from Amazon Aurora MySQL to other AWS services \(p. 996\)](#).

For an Aurora global database, enable outbound connections for each Aurora cluster in the global database.

Granting privileges to save data in Aurora MySQL

The database user that issues the `SELECT INTO OUTFILE S3` statement must have a specific role or privilege. In Aurora MySQL version 3, you grant the `AWS_SELECT_S3_ACCESS` role. In Aurora MySQL version 1 or 2, you grant the `SELECT INTO S3` privilege. The administrative user for a DB cluster is granted the appropriate role or privilege by default. You can grant the privilege to another user by using one of the following statements.

Use the following statement for Aurora MySQL version 3:

```
GRANT AWS_SELECT_S3_ACCESS TO 'user'@'domain-or-ip-address'
```

Tip

When you use the role technique in Aurora MySQL version 3, you also activate the role by using the `SET ROLE role_name` or `SET ROLE ALL` statement. If you aren't familiar with the MySQL 8.0 role system, you can learn more in [Role-based privilege model \(p. 757\)](#). You can also find more details in [Using Roles](#) in the *MySQL Reference Manual*.

Use the following statement for Aurora MySQL version 1 or 2:

```
GRANT SELECT INTO S3 ON *.* TO 'user'@'domain-or-ip-address'
```

The `AWS_SELECT_S3_ACCESS` role and `SELECT INTO S3` privilege are specific to Amazon Aurora MySQL and are not available for MySQL databases or RDS for MySQL DB instances. If you have set up replication between an Aurora MySQL DB cluster as the replication master and a MySQL database as the replication client, then the `GRANT` statement for the role or privilege causes replication to stop with an error. You can safely skip the error to resume replication. To skip the error on an RDS for MySQL DB instance, use the `mysql_rds_skip_repl_error` procedure. To skip the error on an external MySQL database, use the `SET GLOBAL sql_slave_skip_counter` statement (Aurora MySQL version 1 and 2) or `SET GLOBAL sql_replica_skip_counter` statement (Aurora MySQL version 3).

Specifying a path to an Amazon S3 bucket

The syntax for specifying a path to store the data and manifest files on an Amazon S3 bucket is similar to that used in the `LOAD DATA FROM S3 PREFIX` statement, as shown following.

```
s3-region://bucket-name/file-prefix
```

The path includes the following values:

- `region` (optional) – The AWS Region that contains the Amazon S3 bucket to save the data into. This value is optional. If you don't specify a `region` value, then Aurora saves your files into Amazon S3 in the same region as your DB cluster.
- `bucket-name` – The name of the Amazon S3 bucket to save the data into. Object prefixes that identify a virtual folder path are supported.
- `file-prefix` – The Amazon S3 object prefix that identifies the files to be saved in Amazon S3.

The data files created by the `SELECT INTO OUTFILE S3` statement use the following path, in which `00000` represents a 5-digit, zero-based integer number.

```
s3-region://bucket-name/file-prefix.part_00000
```

For example, suppose that a `SELECT INTO OUTFILE S3` statement specifies `s3-us-west-2://bucket/prefix` as the path in which to store data files and creates three data files. The specified Amazon S3 bucket contains the following data files.

- s3-us-west-2://bucket/prefix.part_00000
- s3-us-west-2://bucket/prefix.part_00001
- s3-us-west-2://bucket/prefix.part_00002

Creating a manifest to list data files

You can use the `SELECT INTO OUTFILE S3` statement with the `MANIFEST ON` option to create a manifest file in JSON format that lists the text files created by the statement. The `LOAD DATA FROM S3` statement can use the manifest file to load the data files back into an Aurora MySQL DB cluster. For more information about using a manifest to load data files from Amazon S3 into an Aurora MySQL DB cluster, see [Using a manifest to specify data files to load \(p. 1000\)](#).

The data files included in the manifest created by the `SELECT INTO OUTFILE S3` statement are listed in the order that they're created by the statement. For example, suppose that a `SELECT INTO OUTFILE S3` statement specified `s3-us-west-2://bucket/prefix` as the path in which to store data files and creates three data files and a manifest file. The specified Amazon S3 bucket contains a manifest file named `s3-us-west-2://bucket/prefix.manifest`, that contains the following information.

```
{
  "entries": [
    {
      "url": "s3-us-west-2://bucket/prefix.part_00000"
    },
    {
      "url": "s3-us-west-2://bucket/prefix.part_00001"
    },
    {
      "url": "s3-us-west-2://bucket/prefix.part_00002"
    }
  ]
}
```

SELECT INTO OUTFILE S3

You can use the `SELECT INTO OUTFILE S3` statement to query data from a DB cluster and save it directly into delimited text files stored in an Amazon S3 bucket. Compressed files are not supported. Encrypted files are supported starting in Aurora MySQL 2.09.0.

Syntax

```
SELECT
  [ALL | DISTINCT | DISTINCTROW ]
  [HIGH_PRIORITY]
  [STRAIGHT_JOIN]
  [SQL_SMALL_RESULT] [SQL_BIG_RESULT] [SQL_BUFFER_RESULT]
  [SQL_CACHE | SQL_NO_CACHE] [SQL_CALC_FOUND_ROWS]
  select_expr [, select_expr ...]
  [FROM table_references
    [PARTITION partition_list]
  [WHERE where_condition]
  [GROUP BY {col_name | expr | position}
    [ASC | DESC], ... [WITH ROLLUP]]
  [HAVING where_condition]
  [ORDER BY {col_name | expr | position}
    [ASC | DESC], ...]
  [LIMIT {[offset,] row_count | row_count OFFSET offset}]
  [PROCEDURE procedure_name(argument_list)]
  INTO OUTFILE S3 's3_uri'
  [CHARACTER SET charset_name]
```

```
[export_options]
[MANIFEST {ON | OFF}]
[OVERWRITE {ON | OFF}]

export_options:
[FORMAT {CSV|TEXT} [HEADER]]
[{:FIELDS | COLUMNS}
    [TERMINATED BY 'string']
    [[OPTIONALLY] ENCLOSED BY 'char']
    [ESCAPED BY 'char']]
]
[LINES
    [STARTING BY 'string']
    [TERMINATED BY 'string']]
]
```

Parameters

Following, you can find a list of the required and optional parameters used by the `SELECT INTO OUTFILE S3` statement that are specific to Aurora.

- **s3-uri** – Specifies the URI for an Amazon S3 prefix to use. Specify the URI using the syntax described in [Specifying a path to an Amazon S3 bucket \(p. 1006\)](#).
- **FORMAT {CSV|TEXT} [HEADER]** – Optionally saves the data in CSV format. This syntax is available in Aurora MySQL version 2.07.0 and later.

The `TEXT` option is the default and produces the existing MySQL export format.

The `CSV` option produces comma-separated data values. The CSV format follows the specification in [RFC-4180](#). If you specify the optional keyword `HEADER`, the output file contains one header line. The labels in the header line correspond to the column names from the `SELECT` statement. You can use the CSV files for training data models for use with AWS ML services. For more information about using exported Aurora data with AWS ML services, see [Exporting data to Amazon S3 for SageMaker model training \(p. 1026\)](#).

- **MANIFEST {ON | OFF}** – Indicates whether a manifest file is created in Amazon S3. The manifest file is a JavaScript Object Notation (JSON) file that can be used to load data into an Aurora DB cluster with the `LOAD DATA FROM S3 MANIFEST` statement. For more information about `LOAD DATA FROM S3 MANIFEST`, see [Loading data into an Amazon Aurora MySQL DB cluster from text files in an Amazon S3 bucket \(p. 997\)](#).

If `MANIFEST ON` is specified in the query, the manifest file is created in Amazon S3 after all data files have been created and uploaded. The manifest file is created using the following path:

```
s3-region://bucket-name/file-prefix.manifest
```

For more information about the format of the manifest file's contents, see [Creating a manifest to list data files \(p. 1007\)](#).

- **OVERWRITE {ON | OFF}** – Indicates whether existing files in the specified Amazon S3 bucket are overwritten. If `OVERWRITE ON` is specified, existing files that match the file prefix in the URI specified in `s3-uri` are overwritten. Otherwise, an error occurs.

You can find more details about other parameters in [SELECT syntax](#) and [LOAD DATA INFILE syntax](#), in the MySQL documentation.

Considerations

The number of files written to the Amazon S3 bucket depends on the amount of data selected by the `SELECT INTO OUTFILE S3` statement and the file size threshold for Aurora MySQL. The default

file size threshold is 6 gigabytes (GB). If the data selected by the statement is less than the file size threshold, a single file is created; otherwise, multiple files are created. Other considerations for files created by this statement include the following:

- Aurora MySQL guarantees that rows in data files are not split across file boundaries. For multiple files, the size of every data file except the last is typically close to the file size threshold. However, occasionally staying under the file size threshold results in a row being split across two data files. In this case, Aurora MySQL creates a data file that keeps the row intact, but might be larger than the file size threshold.
- Because each `SELECT` statement in Aurora MySQL runs as an atomic transaction, a `SELECT INTO OUTFILE S3` statement that selects a large data set might run for some time. If the statement fails for any reason, you might need to start over and issue the statement again. If the statement fails, however, files already uploaded to Amazon S3 remain in the specified Amazon S3 bucket. You can use another statement to upload the remaining data instead of starting over again.
- If the amount of data to be selected is large (more than 25 GB), we recommend that you use multiple `SELECT INTO OUTFILE S3` statements to save the data to Amazon S3. Each statement should select a different portion of the data to be saved, and also specify a different `file_prefix` in the `s3-uri` parameter to use when saving the data files. Partitioning the data to be selected with multiple statements makes it easier to recover from an error in one statement. If an error occurs for one statement, only a portion of data needs to be re-selected and uploaded to Amazon S3. Using multiple statements also helps to avoid a single long-running transaction, which can improve performance.
- If multiple `SELECT INTO OUTFILE S3` statements that use the same `file_prefix` in the `s3-uri` parameter run in parallel to select data into Amazon S3, the behavior is undefined.
- Metadata, such as table schema or file metadata, is not uploaded by Aurora MySQL to Amazon S3.
- In some cases, you might re-run a `SELECT INTO OUTFILE S3` query, such as to recover from a failure. In these cases, you must either remove any existing data files in the Amazon S3 bucket with the same file prefix specified in `s3-uri`, or include `OVERWRITE ON` in the `SELECT INTO OUTFILE S3` query.

The `SELECT INTO OUTFILE S3` statement returns a typical MySQL error number and response on success or failure. If you don't have access to the MySQL error number and response, the easiest way to determine when it's done is by specifying `MANIFEST ON` in the statement. The manifest file is the last file written by the statement. In other words, if you have a manifest file, the statement has completed.

Currently, there's no way to directly monitor the progress of the `SELECT INTO OUTFILE S3` statement while it runs. However, suppose that you're writing a large amount of data from Aurora MySQL to Amazon S3 using this statement, and you know the size of the data selected by the statement. In this case, you can estimate progress by monitoring the creation of data files in Amazon S3.

To do so, you can use the fact that a data file is created in the specified Amazon S3 bucket for about every 6 GB of data selected by the statement. Divide the size of the data selected by 6 GB to get the estimated number of data files to create. You can then estimate the progress of the statement by monitoring the number of files uploaded to Amazon S3 while the statement runs.

Examples

The following statement selects all of the data in the `employees` table and saves the data into an Amazon S3 bucket that is in a different region from the Aurora MySQL DB cluster. The statement creates data files in which each field is terminated by a comma (,) character and each row is terminated by a newline (\n) character. The statement returns an error if files that match the `sample_employee_data` file prefix exist in the specified Amazon S3 bucket.

```
SELECT * FROM employees INTO OUTFILE S3 's3-us-west-2://aurora-select-into-s3-pdx/
sample_employee_data'
FIELDS TERMINATED BY ','
```

```
LINES TERMINATED BY '\n';
```

The following statement selects all of the data in the `employees` table and saves the data into an Amazon S3 bucket that is in the same region as the Aurora MySQL DB cluster. The statement creates data files in which each field is terminated by a comma (,) character and each row is terminated by a newline (\n) character, and also a manifest file. The statement returns an error if files that match the `sample_employee_data` file prefix exist in the specified Amazon S3 bucket.

```
SELECT * FROM employees INTO OUTFILE S3 's3://aurora-select-into-s3-pdx/
sample_employee_data'
FIELDS TERMINATED BY ','
LINES TERMINATED BY '\n'
MANIFEST ON;
```

The following statement selects all of the data in the `employees` table and saves the data into an Amazon S3 bucket that is in a different region from the Aurora DB cluster. The statement creates data files in which each field is terminated by a comma (,) character and each row is terminated by a newline (\n) character. The statement overwrites any existing files that match the `sample_employee_data` file prefix in the specified Amazon S3 bucket.

```
SELECT * FROM employees INTO OUTFILE S3 's3-us-west-2://aurora-select-into-s3-pdx/
sample_employee_data'
FIELDS TERMINATED BY ','
LINES TERMINATED BY '\n'
OVERWRITE ON;
```

The following statement selects all of the data in the `employees` table and saves the data into an Amazon S3 bucket that is in the same region as the Aurora MySQL DB cluster. The statement creates data files in which each field is terminated by a comma (,) character and each row is terminated by a newline (\n) character, and also a manifest file. The statement overwrites any existing files that match the `sample_employee_data` file prefix in the specified Amazon S3 bucket.

```
SELECT * FROM employees INTO OUTFILE S3 's3://aurora-select-into-s3-pdx/
sample_employee_data'
FIELDS TERMINATED BY ','
LINES TERMINATED BY '\n'
MANIFEST ON
OVERWRITE ON;
```

Related topics

- [Integrating Aurora with other AWS services \(p. 426\)](#)
- [Loading data into an Amazon Aurora MySQL DB cluster from text files in an Amazon S3 bucket \(p. 997\)](#)
- [Managing an Amazon Aurora DB cluster \(p. 367\)](#)
- [Migrating data to an Amazon Aurora DB cluster \(p. 366\)](#)

Invoking a Lambda function from an Amazon Aurora MySQL DB cluster

You can invoke an AWS Lambda function from an Amazon Aurora MySQL-Compatible Edition DB cluster with the native function `lambda_sync` or `lambda_async`. Before invoking a Lambda function from an

Aurora MySQL, the Aurora DB cluster must have access to Lambda. For details about granting access to Aurora MySQL, see [Giving Aurora access to Lambda \(p. 1011\)](#). For information about the `lambda_sync` and `lambda_async` stored functions, see [Invoking a Lambda function with an Aurora MySQL native function \(p. 1012\)](#).

You can also call an AWS Lambda function by using a stored procedure. However, using a stored procedure is deprecated. We strongly recommend using an Aurora MySQL native function if you are using one of the following Aurora MySQL versions:

- Aurora MySQL version 1.16 and later, for MySQL 5.6-compatible clusters.
- Aurora MySQL version 2.06 and later, for MySQL 5.7-compatible clusters.
- Aurora MySQL version 3.01 and higher, for MySQL 8.0-compatible clusters. The stored procedure is not available in Aurora MySQL version 3.

Topics

- [Giving Aurora access to Lambda \(p. 1011\)](#)
- [Invoking a Lambda function with an Aurora MySQL native function \(p. 1012\)](#)
- [Invoking a Lambda function with an Aurora MySQL stored procedure \(deprecated\) \(p. 1014\)](#)

Giving Aurora access to Lambda

Before you can invoke Lambda functions from an Aurora MySQL DB cluster, make sure to first give your cluster permission to access Lambda.

To give Aurora MySQL access to Lambda

1. Create an AWS Identity and Access Management (IAM) policy that provides the permissions that allow your Aurora MySQL DB cluster to invoke Lambda functions. For instructions, see [Creating an IAM policy to access AWS Lambda resources \(p. 988\)](#).
2. Create an IAM role, and attach the IAM policy you created in [Creating an IAM policy to access AWS Lambda resources \(p. 988\)](#) to the new IAM role. For instructions, see [Creating an IAM role to allow Amazon Aurora to access AWS services \(p. 991\)](#).
3. Set the `aws_default_lambda_role` DB cluster parameter to the Amazon Resource Name (ARN) of the new IAM role.

If the cluster is part of an Aurora global database, apply the same setting for each Aurora cluster in the global database.

For more information about DB cluster parameters, see [Amazon Aurora DB cluster and DB instance parameters \(p. 341\)](#).

4. To permit database users in an Aurora MySQL DB cluster to invoke Lambda functions, associate the role that you created in [Creating an IAM role to allow Amazon Aurora to access AWS services \(p. 991\)](#) with the DB cluster. For information about associating an IAM role with a DB cluster, see [Associating an IAM role with an Amazon Aurora MySQL DB cluster \(p. 992\)](#).

If the cluster is part of an Aurora global database, associate the role with each Aurora cluster in the global database.

5. Configure your Aurora MySQL DB cluster to allow outbound connections to Lambda. For instructions, see [Enabling network communication from Amazon Aurora MySQL to other AWS services \(p. 996\)](#).

If the cluster is part of an Aurora global database, enable outbound connections for each Aurora cluster in the global database.

Invoking a Lambda function with an Aurora MySQL native function

Note

You can call the native functions `lambda_sync` and `lambda_async` when you use Aurora MySQL version 1.16 and later, Aurora MySQL 2.06 and later, or Aurora MySQL version 3.01 and higher. For more information about Aurora MySQL versions, see [Database engine updates for Amazon Aurora MySQL \(p. 1082\)](#).

You can invoke an AWS Lambda function from an Aurora MySQL DB cluster by calling the native functions `lambda_sync` and `lambda_async`. This approach can be useful when you want to integrate your database running on Aurora MySQL with other AWS services. For example, you might want to send a notification using Amazon Simple Notification Service (Amazon SNS) whenever a row is inserted into a specific table in your database.

Working with native functions to invoke a Lambda function

The `lambda_sync` and `lambda_async` functions are built-in, native functions that invoke a Lambda function synchronously or asynchronously. When you must know the result of the Lambda function before moving on to another action, use the synchronous function `lambda_sync`. When you don't need to know the result of the Lambda function before moving on to another action, use the asynchronous function `lambda_async`.

In Aurora MySQL version 3, the user invoking a native function must be granted the `AWS_LAMBDA_ACCESS` role. To grant this role to a user, connect to the DB instance as the administrative user, and run the following statement.

```
GRANT AWS_LAMBDA_ACCESS TO user@domain-or-ip-address
```

You can revoke this role by running the following statement.

```
REVOKE AWS_LAMBDA_ACCESS FROM user@domain-or-ip-address
```

Tip

When you use the role technique in Aurora MySQL version 3, you also activate the role by using the `SET ROLE role_name` or `SET ROLE ALL` statement. If you aren't familiar with the MySQL 8.0 role system, you can learn more in [Role-based privilege model \(p. 757\)](#). You can also find more details in [Using Roles in the MySQL Reference Manual](#).

In Aurora MySQL version 1 and 2, the user invoking a native function must be granted the `INVOKELAMBDA` privilege. To grant this privilege to a user, connect to the DB instance as the administrative user, and run the following statement.

```
GRANT INVOKELAMBDA ON *.* TO user@domain-or-ip-address
```

You can revoke this privilege by running the following statement.

```
REVOKE INVOKELAMBDA ON *.* FROM user@domain-or-ip-address
```

Syntax for the `lambda_sync` function

You invoke the `lambda_sync` function synchronously with the `RequestResponse` invocation type. The function returns the result of the Lambda invocation in a JSON payload. The function has the following syntax.

```
lambda_sync (
```

```
lambda_function_ARN,  
JSON_payload  
)
```

Note

You can use triggers to call Lambda on data-modifying statements. Remember that triggers are not run once per SQL statement, but once per row modified, one row at a time. When a trigger runs, the process is synchronous. The data-modifying statement only returns when the trigger completes.

Be careful when invoking an AWS Lambda function from triggers on tables that experience high write traffic. `INSERT`, `UPDATE`, and `DELETE` triggers are activated per row. A write-heavy workload on a table with `INSERT`, `UPDATE`, or `DELETE` triggers results in a large number of calls to your AWS Lambda function.

Parameters for the `lambda_sync` function

The `lambda_sync` function has the following parameters.

lambda_function_ARN

The Amazon Resource Name (ARN) of the Lambda function to invoke.

JSON_payload

The payload for the invoked Lambda function, in JSON format.

Note

Aurora MySQL version 3 supports the JSON parsing functions from MySQL 8.0. However, Aurora MySQL versions 1 and 2 don't include those functions. JSON parsing isn't required when a Lambda function returns an atomic value, such as a number or a string.

Example for the `lambda_sync` function

The following query based on `lambda_sync` invokes the Lambda function `BasicTestLambda` synchronously using the function ARN. The payload for the function is `{"operation": "ping"}`.

```
SELECT lambda_sync(  
    'arn:aws:lambda:us-east-1:868710585169:function:BasicTestLambda',  
    '{"operation": "ping"}');
```

Syntax for the `lambda_async` function

You invoke the `lambda_async` function asynchronously with the `Event` invocation type. The function returns the result of the Lambda invocation in a JSON payload. The function has the following syntax.

```
lambda_async (  
    lambda_function_ARN,  
    JSON_payload  
)
```

Parameters for the `lambda_async` function

The `lambda_async` function has the following parameters.

lambda_function_ARN

The Amazon Resource Name (ARN) of the Lambda function to invoke.

JSON_payload

The payload for the invoked Lambda function, in JSON format.

Note

Aurora MySQL version 3 supports the JSON parsing functions from MySQL 8.0. However, Aurora MySQL versions 1 and 2 don't include those functions. JSON parsing isn't required when a Lambda function returns an atomic value, such as a number or a string.

Example for the `lambda_async` function

The following query based on `lambda_async` invokes the Lambda function `BasicTestLambda` asynchronously using the function ARN. The payload for the function is `{"operation": "ping"}`.

```
SELECT lambda_async(  
    'arn:aws:lambda:us-east-1:868710585169:function:BasicTestLambda',  
    '{"operation": "ping"}');
```

Invoking a Lambda function with an Aurora MySQL stored procedure (deprecated)

You can invoke an AWS Lambda function from an Aurora MySQL DB cluster by calling the `mysql.lambda_async` procedure. This approach can be useful when you want to integrate your database running on Aurora MySQL with other AWS services. For example, you might want to send a notification using Amazon Simple Notification Service (Amazon SNS) whenever a row is inserted into a specific table in your database.

Aurora MySQL version considerations

Starting in Aurora MySQL version 1.8 and Aurora MySQL version 2.06, you can use the native function method instead of these stored procedures to invoke a Lambda function. For more information about the native functions, see [Working with native functions to invoke a Lambda function \(p. 1012\)](#).

Starting with Amazon Aurora version 1.16 and 2.06, the stored procedure `mysql.lambda_async` is no longer supported. If you are using an Aurora version that's higher than 1.16 or 2.06, we strongly recommend that you work with native Lambda functions instead. In Aurora MySQL version 3, the stored procedure isn't available.

Working with the `mysql.lambda_async` procedure to invoke a Lambda function (deprecated)

The `mysql.lambda_async` procedure is a built-in stored procedure that invokes a Lambda function asynchronously. To use this procedure, your database user must have `EXECUTE` privilege on the `mysql.lambda_async` stored procedure.

Syntax

The `mysql.lambda_async` procedure has the following syntax.

```
CALL mysql.lambda_async (  
    lambda_function_ARN,  
    lambda_function_input  
)
```

Parameters

The `mysql.lambda_async` procedure has the following parameters.

lambda_function_ARN

The Amazon Resource Name (ARN) of the Lambda function to invoke.

lambda_function_input

The input string, in JSON format, for the invoked Lambda function.

Examples

As a best practice, we recommend that you wrap calls to the `mysql.lambda_async` procedure in a stored procedure that can be called from different sources such as triggers or client code. This approach can help to avoid impedance mismatch issues and make it easier to invoke Lambda functions.

Note

Be careful when invoking an AWS Lambda function from triggers on tables that experience high write traffic. `INSERT`, `UPDATE`, and `DELETE` triggers are activated per row. A write-heavy workload on a table with `INSERT`, `UPDATE`, or `DELETE` triggers results in a large number of calls to your AWS Lambda function.

Although calls to the `mysql.lambda_async` procedure are asynchronous, triggers are synchronous. A statement that results in a large number of trigger activations doesn't wait for the call to the AWS Lambda function to complete, but it does wait for the triggers to complete before returning control to the client.

Example Example: Invoke an AWS Lambda function to send email

The following example creates a stored procedure that you can call in your database code to send an email using a Lambda function.

AWS Lambda Function

```
import boto3

ses = boto3.client('ses')

def SES_send_email(event, context):

    return ses.send_email(
        Source=event['email_from'],
        Destination={
            'ToAddresses': [
                event['email_to'],
            ]
        },
        Message={
            'Subject': {
                'Data': event['email_subject']
            },
            'Body': {
                'Text': {
                    'Data': event['email_body']
                }
            }
        }
    )
```

Stored Procedure

```
DROP PROCEDURE IF EXISTS SES_send_email;
DELIMITER ;;
CREATE PROCEDURE SES_send_email(IN email_from VARCHAR(255),
                                IN email_to VARCHAR(255),
                                IN subject VARCHAR(255),
                                IN body TEXT) LANGUAGE SQL
BEGIN
```

```
CALL mysql.lambda_async(
    'arn:aws:lambda:us-west-2:123456789012:function:SES_send_email',
    CONCAT('{"email_to" : "', email_to,
           '", "email_from" : "", email_from,
           '", "email_subject" : "", subject,
           '", "email_body" : "", body, "")')
);
END
;;
DELIMITER ;
```

Call the Stored Procedure to Invoke the AWS Lambda Function

```
mysql> call SES_send_email('example_from@amazon.com', 'example_to@amazon.com', 'Email subject', 'Email content');
```

Example Example: Invoke an AWS Lambda function to publish an event from a trigger

The following example creates a stored procedure that publishes an event by using Amazon SNS. The code calls the procedure from a trigger when a row is added to a table.

AWS Lambda Function

```
import boto3

sns = boto3.client('sns')

def SNS_publish_message(event, context):

    return sns.publish(
        TopicArn='arn:aws:sns:us-west-2:123456789012:Sample_Topic',
        Message=event['message'],
        Subject=event['subject'],
        MessageStructure='string'
    )
```

Stored Procedure

```
DROP PROCEDURE IF EXISTS SNS_Publish_Message;
DELIMITER ;;
CREATE PROCEDURE SNS_Publish_Message (IN subject VARCHAR(255),
                                         IN message TEXT) LANGUAGE SQL
BEGIN
    CALL mysql.lambda_async('arn:aws:lambda:us-
west-2:123456789012:function:SNS_publish_message',
                           CONCAT('{ "subject" : "", subject,
                                   "", "message" : "", message, "" }'))
END
;;
DELIMITER ;
```

Table

```
CREATE TABLE 'Customer_Feedback' (
    'id' int(11) NOT NULL AUTO_INCREMENT,
    'customer_name' varchar(255) NOT NULL,
    'customer_feedback' varchar(1024) NOT NULL,
    PRIMARY KEY ('id')
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

Trigger

```
DELIMITER ;;
CREATE TRIGGER TR_Customer_Feedback_AI
    AFTER INSERT ON Customer_Feedback
    FOR EACH ROW
BEGIN
    SELECT CONCAT('New customer feedback from ', NEW.customer_name), NEW.customer_feedback
    INTO @subject, @feedback;
    CALL SNS_Publish_Message(@subject, @feedback);
END
;;
DELIMITER ;
```

Insert a Row into the Table to Trigger the Notification

```
mysql> insert into Customer_Feedback (customer_name, customer_feedback) VALUES ('Sample
Customer', 'Good job guys!');
```

Publishing Amazon Aurora MySQL logs to Amazon CloudWatch Logs

You can configure your Aurora MySQL DB cluster to publish general, slow, audit, and error log data to a log group in Amazon CloudWatch Logs. With CloudWatch Logs, you can perform real-time analysis of the log data, and use CloudWatch to create alarms and view metrics. You can use CloudWatch Logs to store your log records in highly durable storage.

To publish logs to CloudWatch Logs, the respective logs must be enabled. Error logs are enabled by default, but you must enable the other types of logs explicitly. For information about enabling logs in MySQL, see [Selecting general query and slow query log output destinations](#) in the MySQL documentation. For more information about enabling Aurora MySQL audit logs, see [Enabling Advanced Auditing \(p. 915\)](#).

Note

Be aware of the following:

- You can't publish logs to CloudWatch Logs for the China (Ningxia) region.
- If exporting log data is disabled, Aurora doesn't delete existing log groups or log streams. If exporting log data is disabled, existing log data remains available in CloudWatch Logs, depending on log retention, and you still incur charges for stored audit log data. You can delete log streams and log groups using the CloudWatch Logs console, the AWS CLI, or the CloudWatch Logs API.
- An alternative way to publish audit logs to CloudWatch Logs is by enabling advanced auditing and setting the cluster-level DB parameter `server_audit_logs_upload` to 1. The default for the `server_audit_logs_upload` parameter is 0.

If you use this alternative method, you must have an IAM role to access CloudWatch Logs and set the `aws_default_logs_role` cluster-level parameter to the ARN for this role. For information about creating the role, see [Setting up IAM roles to access AWS services \(p. 985\)](#). However, if you have the `AWSServiceRoleForRDS` service-linked role, it provides access to CloudWatch Logs and overrides any custom-defined roles. For information service-linked roles for Amazon RDS, see [Using service-linked roles for Amazon Aurora \(p. 1783\)](#).

- If you don't want to export audit logs to CloudWatch Logs, make sure that all methods of exporting audit logs are disabled. These methods are the AWS Management Console, the AWS CLI, the RDS API, and the `server_audit_logs_upload` parameter.

- The procedure is slightly different for Aurora Serverless clusters than for provisioned clusters. Serverless clusters automatically upload all the kinds of logs that you enable through the configuration parameters. Therefore, you enable or disable log upload for Serverless clusters by turning different log types on and off in the DB cluster parameter group. You don't modify the settings of the cluster itself through the AWS Management Console, AWS CLI, or RDS API. For information about enabling MySQL logs for Serverless clusters, see [Parameter groups and Aurora Serverless v1 \(p. 156\)](#).

Console

You can publish Aurora MySQL logs for provisioned clusters to CloudWatch Logs with the console.

To publish Aurora MySQL logs from the console

1. Open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. In the navigation pane, choose **Databases**.
3. Choose the Aurora MySQL DB cluster that you want to publish the log data for.
4. Choose **Modify**.
5. In the **Log exports** section, choose the logs that you want to start publishing to CloudWatch Logs.
6. Choose **Continue**, and then choose **Modify DB Cluster** on the summary page.

AWS CLI

You can publish Aurora MySQL logs for provisioned clusters with the AWS CLI. To do so, you run the [modify-db-cluster](#) AWS CLI command with the following options:

- `--db-cluster-identifier`—The DB cluster identifier.
- `--cloudwatch-logs-export-configuration`—The configuration setting for the log types to be enabled for export to CloudWatch Logs for the DB cluster.

You can also publish Aurora MySQL logs by running one of the following AWS CLI commands:

- [create-db-cluster](#)
- [restore-db-cluster-from-s3](#)
- [restore-db-cluster-from-snapshot](#)
- [restore-db-cluster-to-point-in-time](#)

Run one of these AWS CLI commands with the following options:

- `--db-cluster-identifier`—The DB cluster identifier.
- `--engine`—The database engine.
- `--enable-cloudwatch-logs-exports`—The configuration setting for the log types to be enabled for export to CloudWatch Logs for the DB cluster.

Other options might be required depending on the AWS CLI command that you run.

Example

The following command modifies an existing Aurora MySQL DB cluster to publish log files to CloudWatch Logs.

For Linux, macOS, or Unix:

```
aws rds modify-db-cluster \
--db-cluster-identifier mydbcluster \
--cloudwatch-logs-export-configuration '{"EnableLogTypes": \
["error", "general", "slowquery", "audit"]}'
```

For Windows:

```
aws rds modify-db-cluster ^
--db-cluster-identifier mydbcluster ^
--cloudwatch-logs-export-configuration '{"EnableLogTypes": \
["error", "general", "slowquery", "audit"]}'
```

Example

The following command creates an Aurora MySQL DB cluster to publish log files to CloudWatch Logs.

For Linux, macOS, or Unix:

```
aws rds create-db-cluster \
--db-cluster-identifier mydbcluster \
--engine aurora \
--enable-cloudwatch-logs-exports '[ "error", "general", "slowquery", "audit" ]'
```

For Windows:

```
aws rds create-db-cluster ^
--db-cluster-identifier mydbcluster ^
--engine aurora ^
--enable-cloudwatch-logs-exports '[ "error", "general", "slowquery", "audit" ]'
```

RDS API

You can publish Aurora MySQL logs for provisioned clusters with the RDS API. To do so, you run the [ModifyDBCluster](#) operation with the following options:

- **DBClusterIdentifier**—The DB cluster identifier.
- **CloudwatchLogsExportConfiguration**—The configuration setting for the log types to be enabled for export to CloudWatch Logs for the DB cluster.

You can also publish Aurora MySQL logs with the RDS API by running one of the following RDS API operations:

- [CreateDBCluster](#)
- [RestoreDBClusterFromS3](#)
- [RestoreDBClusterFromSnapshot](#)
- [RestoreDBClusterToPointInTime](#)

Run the RDS API operation with the following parameters:

- **DBClusterIdentifier**—The DB cluster identifier.

- **Engine**—The database engine.
- **EnableCloudwatchLogsExports**—The configuration setting for the log types to be enabled for export to CloudWatch Logs for the DB cluster.

Other parameters might be required depending on the AWS CLI command that you run.

Monitoring log events in Amazon CloudWatch

After enabling Aurora MySQL log events, you can monitor the events in Amazon CloudWatch Logs. A new log group is automatically created for the Aurora DB cluster under the following prefix, in which *cluster-name* represents the DB cluster name, and *log_type* represents the log type.

```
/aws/rds/cluster/cluster-name/log_type
```

For example, if you configure the export function to include the slow query log for a DB cluster named *mydbcluster*, slow query data is stored in the /aws/rds/cluster/mydbcluster/slowquery log group.

The events from all instances in your cluster are pushed to a log group using different log streams. The behavior depends on which of the following conditions is true:

- A log group with the specified name exists.

Aurora uses the existing log group to export log data for the cluster. To create log groups with predefined log retention periods, metric filters, and customer access, you can use automated configuration, such as AWS CloudFormation.
- A log group with the specified name doesn't exist.

When a matching log entry is detected in the log file for the instance, Aurora MySQL creates a new log group in CloudWatch Logs automatically. The log group uses the default log retention period of **Never Expire**.

To change the log retention period, use the CloudWatch Logs console, the AWS CLI, or the CloudWatch Logs API. For more information about changing log retention periods in CloudWatch Logs, see [Change log data retention in CloudWatch Logs](#).

To search for information within the log events for a DB cluster, use the CloudWatch Logs console, the AWS CLI, or the CloudWatch Logs API. For more information about searching and filtering log data, see [Searching and filtering log data](#).

Using machine learning (ML) with Aurora MySQL

With Aurora machine learning, you can add machine learning-based predictions to database applications using the SQL language. Aurora machine learning uses a highly optimized integration between the Aurora database and the AWS machine learning (ML) services SageMaker and Amazon Comprehend.

Benefits of Aurora machine learning include the following:

- You can add ML-based predictions to your existing database applications. You don't need to build custom integrations or learn separate tools. You can embed machine learning processing directly into your SQL query as calls to stored functions.
- The ML integration is a fast way to enable ML services to work with transactional data. You don't have to move the data out of the database to perform the machine learning operations. You don't have to convert or reimport the results of the machine learning operations to use them in your database application.

- You can use your existing governance policies to control who has access to the underlying data and to the generated insights.

AWS ML Services are managed services that are set up and run in their own production environments. Currently, Aurora machine learning integrates with Amazon Comprehend for sentiment analysis and SageMaker for a wide variety of ML algorithms.

For details about using Aurora and Amazon Comprehend together, see [Using Amazon Comprehend for sentiment detection \(p. 1029\)](#). For general information about Amazon Comprehend, see [Amazon Comprehend](#).

For details about using Aurora and SageMaker together, see [Using SageMaker to run your own ML models \(p. 1027\)](#). For general information about SageMaker, see [SageMaker](#).

Topics

- [Prerequisites for Aurora machine learning \(p. 1021\)](#)
- [Enabling Aurora machine learning \(p. 1021\)](#)
- [Exporting data to Amazon S3 for SageMaker model training \(p. 1026\)](#)
- [Using SageMaker to run your own ML models \(p. 1027\)](#)
- [Using Amazon Comprehend for sentiment detection \(p. 1029\)](#)
- [Performance considerations for Aurora machine learning \(p. 1030\)](#)
- [Monitoring Aurora machine learning \(p. 1031\)](#)
- [Limitations of Aurora machine learning \(p. 1032\)](#)

Prerequisites for Aurora machine learning

Aurora machine learning is available for any Aurora cluster that's running an Aurora MySQL 2.07.0 or higher version in an AWS Region that supports Aurora machine learning. You can upgrade an Aurora cluster that's running a lower version of Aurora MySQL to a supported higher version if you want to use Aurora machine learning with that cluster. For more information, see [Database engine updates for Amazon Aurora MySQL \(p. 1082\)](#).

For more information about Regions and Aurora version availability, see [Aurora machine learning \(p. 23\)](#).

Enabling Aurora machine learning

Enabling the ML capabilities involves the following steps:

- You enable the Aurora cluster to access the Amazon machine learning services SageMaker or Amazon Comprehend, depending the kinds of ML algorithms you want for your application.
- For SageMaker, then you use the Aurora CREATE FUNCTION statement to set up stored functions that access inference features.

Note

Aurora machine learning includes built-in functions that call Amazon Comprehend for sentiment analysis. You don't need to run any CREATE FUNCTION statements if you only use Amazon Comprehend.

Topics

- [Setting up IAM access to Amazon Comprehend and SageMaker \(p. 1022\)](#)
- [Granting SQL privileges for invoking Aurora machine learning services \(p. 1026\)](#)
- [Enabling network communication from Aurora MySQL to other AWS services \(p. 1026\)](#)

Setting up IAM access to Amazon Comprehend and SageMaker

Before you can access SageMaker and Amazon Comprehend services, enable the Aurora MySQL cluster to access AWS ML services. For your Aurora MySQL DB cluster to access AWS ML services on your behalf, create and configure AWS Identity and Access Management (IAM) roles. These roles authorize the users of your Aurora MySQL database to access AWS ML services.

When you use the AWS Management Console, AWS does the IAM setup for you automatically. You can skip the following information and follow the procedure in [Connecting an Aurora DB cluster to Amazon S3, SageMaker, or Amazon Comprehend using the console \(p. 1022\)](#).

Setting up the IAM roles for SageMaker or Amazon Comprehend using the AWS CLI or the RDS API consists of the following steps:

1. Create an IAM policy to specify which SageMaker endpoints can be invoked by your Aurora MySQL cluster or to enable access to Amazon Comprehend.
2. Create an IAM role to permit your Aurora MySQL database cluster to access AWS ML services. The IAM policy created above is attached to the IAM role.
3. To permit the Aurora MySQL database cluster to access AWS ML services, you associate the IAM role that you created above to the database cluster.
4. To permit database applications to invoke AWS ML services, you must also grant privileges to specific database users. For SageMaker, because the calls to the endpoints are wrapped inside a stored function, you also grant `EXECUTE` privileges on the stored functions to any database users that call them.

For general information about how to permit your Aurora MySQL DB cluster to access other AWS services on your behalf, see [Authorizing Amazon Aurora MySQL to access other AWS services on your behalf \(p. 985\)](#).

[Connecting an Aurora DB cluster to Amazon S3, SageMaker, or Amazon Comprehend using the console](#)

Aurora machine learning requires that your DB cluster use some combination of Amazon S3, SageMaker, and Amazon Comprehend. Amazon Comprehend is for sentiment analysis. SageMaker is for a wide variety of machine learning algorithms. For Aurora machine learning, Amazon S3 is only for training SageMaker models. You only need to use Amazon S3 with Aurora machine learning if you don't already have a trained model available and the training is your responsibility. To connect a DB cluster to these services requires that you set up an AWS Identity and Access Management (IAM) role for each Amazon service. The IAM role enables users of your DB cluster to authenticate with the corresponding service.

To generate the IAM roles for Amazon S3, SageMaker, or Amazon Comprehend, repeat the following steps for each service that you need.

To connect a DB cluster to an Amazon service

1. Sign in to the AWS Management Console and open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. In the navigation pane, choose **Databases**, and then choose the Aurora MySQL DB cluster that you want to use.
3. Choose the **Connectivity & security** tab.
4. Choose **Select a service to connect to this cluster** in the **Manage IAM roles** section., and choose the service that you want to connect to:
 - **Amazon S3**
 - **Amazon Comprehend**
 - **SageMaker**

5. Choose **Connect service**.
6. Enter the required information for the specific service on the **Connect cluster** window:
 - For SageMaker, enter the Amazon Resource Name (ARN) of an SageMaker endpoint. For details about what the endpoint represents, see [Deploy a model on Amazon SageMaker hosting services](#).

In the navigation pane of the [SageMaker console](#), choose **Endpoints** and copy the ARN of the endpoint you want to use.

- For Amazon Comprehend, don't specify any additional parameter.
- For Amazon S3, enter the ARN of an Amazon S3 bucket to use.

The format of an Amazon S3 bucket ARN is `arn:aws:s3:::bucket_name`. Ensure that the Amazon S3 bucket that you use is set up with the requirements for training SageMaker models. When you train a model, your Aurora DB cluster requires permission to export data to the Amazon S3 bucket, and also to import data from the bucket.

To learn more about Amazon S3 bucket ARNs, see [Specifying resources in a policy](#) in the *Amazon Simple Storage Service User Guide*. For more about using an Amazon S3 bucket with SageMaker, see [Step 1: Create an Amazon Amazon S3 bucket](#) in the *Amazon SageMaker Developer Guide*.

7. Choose **Connect service**.
8. Aurora creates a new IAM role and adds it to the DB cluster's list of **Current IAM roles for this cluster**. The IAM role's status is initially **In progress**. The IAM role name is autogenerated with the following pattern for each connected service:
 - The Amazon S3 IAM role name pattern is `rds-cluster_ID-S3-policy-timestamp`.
 - The SageMaker IAM role name pattern is `rds-cluster_ID-SageMaker-policy-timestamp`.
 - The Amazon Comprehend IAM role name pattern is `rds-cluster_ID-Comprehend-policy-timestamp`.

Aurora also creates a new IAM policy and attaches it to the role. The policy name follows a similar naming convention and also has a timestamp.

[Creating an IAM policy to access SageMaker \(AWS CLI only\)](#)

Note

When you use the AWS Management Console, Aurora creates the IAM policy automatically. In that case, you can skip this section.

The following policy adds the permissions required by Aurora MySQL to invoke an SageMaker function on your behalf. You can specify all of your SageMaker endpoints that you need your database applications to access from your Aurora MySQL cluster in a single policy. The policy allows you to specify the AWS Region for an SageMaker endpoint. However, an Aurora MySQL cluster can only invoke SageMaker models deployed in the same AWS Region as the cluster.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "AllowAuroraToInvokeRCFEndPoint",  
            "Effect": "Allow",  
            "Action": "sagemaker:InvokeEndpoint",  
            "Resource": "arn:aws:sagemaker:region:123456789012:endpoint/endpointName"  
        }  
    ]  
}
```

The following command performs the same operation through the AWS CLI.

```
aws iam put-role-policy --role-name role_name --policy-name policy_name
--policy-document '{ "Version": "2012-10-17", "Statement": [ { "Sid": "AllowAuroraToInvokeRCFEndpoint", "Effect": "Allow", "Action": "sagemaker:InvokeEndpoint", "Resource": "arn:aws:sagemaker:region:123456789012:endpoint/endpointName } ]}'
```

Creating an IAM policy to access Amazon Comprehend (AWS CLI only)

Note

When you use the AWS Management Console, Aurora creates the IAM policy automatically. In that case, you can skip this section.

The following policy adds the permissions required by Aurora MySQL to invoke Amazon Comprehend on your behalf.

```
{ "Version": "2012-10-17",
"Statement": [
{
  "Sid": "AllowAuroraToInvokeComprehendDetectSentiment",
  "Effect": "Allow",
  "Action": [
    "comprehend:DetectSentiment",
    "comprehend:BatchDetectSentiment"
  ],
  "Resource": "*"
}
]
```

The following command performs the same operation through the AWS CLI.

```
aws iam put-role-policy --role-name role_name --policy-name policy_name
--policy-document '{ "Version": "2012-10-17", "Statement": [ { "Sid": "AllowAuroraToInvokeComprehendDetectSentiment", "Effect": "Allow", "Action": [ "comprehend:DetectSentiment", "comprehend:BatchDetectSentiment" ], "Resource": "*" } ]}'
```

To create an IAM policy to grant access to Amazon Comprehend

1. Open the [IAM Management Console](#).
2. In the navigation pane, choose **Policies**.
3. Choose **Create policy**.
4. On the **Visual editor** tab, choose **Choose a service**, and then choose **Comprehend**.
5. For **Actions**, choose **Detect Sentiment** and **BatchDetectSentiment**.
6. Choose **Review policy**.
7. For **Name**, enter a name for your IAM policy. You use this name when you create an IAM role to associate with your Aurora DB cluster. You can also add an optional **Description** value.
8. Choose **Create policy**.
9. Complete the procedure in [Creating an IAM role to allow Amazon Aurora to access AWS services \(p. 991\)](#).

Creating an IAM role to access SageMaker and Amazon Comprehend

After you create the IAM policies, create an IAM role that the Aurora MySQL cluster can assume on behalf of your database users to access ML services. To create an IAM role, you can use the AWS Management

Console or the AWS CLI. To create an IAM role and attach the preceding policies to the role, follow the steps described in [Creating an IAM role to allow Amazon Aurora to access AWS services \(p. 991\)](#). For more information about IAM roles, see [IAM roles](#) in the *AWS Identity and Access Management User Guide*.

You can only use a global IAM role for authentication. You can't use an IAM role associated with a database user or a session. This requirement is the same as for Aurora integration with the Lambda and Amazon S3 services.

Associating an IAM role with an Aurora MySQL DB cluster (AWS CLI only)

Note

When you use the AWS Management Console, Aurora creates the IAM policy automatically. In that case, you can skip this section.

The last step is to associate the IAM role with the attached IAM policy with your Aurora MySQL DB cluster. To associate an IAM role with an Aurora DB cluster, you do two things:

1. Add the role to the list of associated roles for a DB cluster by using the AWS Management Console, the [add-role-to-db-cluster](#) AWS CLI command, or the [AddRoleToDBCluster](#) RDS API operation.
2. Set the cluster-level parameter for the related AWS ML service to the ARN for the associated IAM role. Use the `aws_default_sagemaker_role`, `aws_default_comprehend_role`, or both parameters depending on which AWS ML services you intend to use with your Aurora cluster.

Cluster-level parameters are grouped into DB cluster parameter groups. To set the preceding cluster parameters, use an existing custom DB cluster group or create a new one. To create a new DB cluster parameter group, call the `create-db-cluster-parameter-group` command from the AWS CLI, as shown following.

```
PROMPT> aws rds create-db-cluster-parameter-group --db-cluster-parameter-group-name AllowAWSAccessToExternalServices \
--db-parameter-group-family aurora-mysql5.7 --description "Allow access to Amazon S3, AWS Lambda, AWS SageMaker, and AWS Comprehend"
```

Set the appropriate cluster-level parameter or parameters and the related IAM role ARN values in your DB cluster parameter group, as shown in the following.

```
PROMPT> aws rds modify-db-cluster-parameter-group \
--db-cluster-parameter-group-name AllowAWSAccessToExternalServices \
--parameters
"ParameterName=aws_default_s3_role,ParameterValue=arn:aws:iam::123456789012:role/AllowAuroraS3Role,ApplyMethod=pending-reboot" \
--parameters
"ParameterName=aws_default_sagemaker_role,ParameterValue=arn:aws:iam::123456789012:role/AllowAuroraSageMakerRole,ApplyMethod=pending-reboot" \
--parameters
"ParameterName=aws_default_comprehend_role,ParameterValue=arn:aws:iam::123456789012:role/AllowAuroraComprehendRole,ApplyMethod=pending-reboot"
```

Modify the DB cluster to use the new DB cluster parameter group. Then, reboot the cluster. The following shows how.

```
PROMPT> aws rds modify-db-cluster --db-cluster-identifier your_cluster_id --db-cluster-parameter-group-nameAllowAWSAccessToExternalServices
PROMPT> aws rds failover-db-cluster --db-cluster-identifier your_cluster_id
```

When the instance has rebooted, your IAM roles are associated with your DB cluster.

Granting SQL privileges for invoking Aurora machine learning services

After you create the required IAM policies and roles and associating the role to the Aurora MySQL DB cluster, you authorize individual database users to invoke the Aurora machine learning stored functions for SageMaker and built-in functions for Amazon Comprehend.

The database user invoking a native function must be granted a corresponding role or privilege. For Aurora MySQL version 3, you grant the `AWS_SAGEMAKER_ACCESS` role or the `AWS_COMPREHEND_ACCESS` role. For Aurora MySQL version 1 or 2, you grant the `INVOKE SAGEMAKER` or `INVOKE COMPREHEND` privilege. To grant this privilege to a user, connect to the DB instance as the administrative user, and run the following statements. Substitute the appropriate details for the database user.

Use the following statements for Aurora MySQL version 3:

```
GRANT AWS_SAGEMAKER_ACCESS TO user@domain-or-ip-address
GRANT AWS_COMPREHEND_ACCESS TO user@domain-or-ip-address
```

Tip

When you use the role technique in Aurora MySQL version 3, you also activate the role by using the `SET ROLE role_name` or `SET ROLE ALL` statement. If you aren't familiar with the MySQL 8.0 role system, you can learn more in [Role-based privilege model \(p. 757\)](#). You can also find more details in [Using Roles in the MySQL Reference Manual](#).

Use the following statements for Aurora MySQL version 1 or 2:

Use the following statements for Aurora MySQL version 3:

```
GRANT AWS_SAGEMAKER_ACCESS TO user@domain-or-ip-address
GRANT AWS_COMPREHEND_ACCESS TO user@domain-or-ip-address
```

Use the following statements for Aurora MySQL version 1 or 2:

```
GRANT INVOKE SAGEMAKER ON *.* TO user@domain-or-ip-address
GRANT INVOKE COMPREHEND ON *.* TO user@domain-or-ip-address
```

For SageMaker, user-defined functions define the parameters to be sent to the model for producing the inference and to configure the endpoint name to be invoked. You grant `EXECUTE` permission to the stored functions configured for SageMaker for each of the database users who intend to invoke the endpoint.

```
GRANT EXECUTE ON FUNCTION db1.anomaly_score TO user1@domain-or-ip-address1
GRANT EXECUTE ON FUNCTION db2.company_forecasts TO user2@domain-or-ip-address2
```

Enabling network communication from Aurora MySQL to other AWS services

Since SageMaker and Amazon Comprehend are external AWS services, you must also configure your Aurora DB cluster to allow outbound connections to the target AWS service. For more information, see [Enabling network communication from Amazon Aurora MySQL to other AWS services \(p. 996\)](#).

You can use VPC endpoints to connect to Amazon S3. AWS PrivateLink can't be used to connect Aurora to AWS machine learning services or Amazon S3 at this time.

Exporting data to Amazon S3 for SageMaker model training

Depending on how your team divides the machine learning tasks, you might not perform this task. If someone else provides the SageMaker model for you, you can skip this section.

To train SageMaker models, you export data to an Amazon S3 bucket. The Amazon S3 bucket is used by a Jupyter SageMaker notebook instance to train your model before it is deployed. You can use the `SELECT INTO OUTFILE S3` statement to query data from an Aurora MySQL DB cluster and save it directly into text files stored in an Amazon S3 bucket. Then the notebook instance consumes the data from the Amazon S3 bucket for training.

Aurora machine learning extends the existing `SELECT INTO OUTFILE` syntax in Aurora MySQL to export data to CSV format. The generated CSV file can be directly consumed by models that need this format for training purposes.

```
SELECT * INTO OUTFILE S3 's3_uri' [FORMAT {CSV|TEXT} [HEADER]] FROM table_name;
```

The extension supports the standard CSV format.

- Format `TEXT` is the same as the existing MySQL export format. This is the default format.
- Format `CSV` is a newly introduced format that follows the specification in [RFC-4180](#).
- If you specify the optional keyword `HEADER`, the output file contains one header line. The labels in the header line correspond to the column names from the `SELECT` statement.
- You can still use the keywords `CSV` and `HEADER` as identifiers.

The extended syntax and grammar of `SELECT INTO` is now as follows:

```
INTO OUTFILE S3 's3_uri'  
[CHARACTER SET charset_name]  
[FORMAT {CSV|TEXT} [HEADER]]  
[{FIELDS | COLUMNS}  
 [TERMINATED BY 'string']  
 [[OPTIONALLY] ENCLOSED BY 'char']  
 [ESCAPED BY 'char']]  
[LINES  
 [STARTING BY 'string']  
 [TERMINATED BY 'string']]
```

Using SageMaker to run your own ML models

SageMaker is a fully managed machine learning service. With SageMaker, data scientists and developers can quickly and easily build and train machine learning models. Then they can directly deploy the models into a production-ready hosted environment. SageMaker provides an integrated Jupyter authoring notebook instance for easy access to your data sources. That way, you can perform exploration and analysis without managing the hardware infrastructure for servers. It also provides common machine learning algorithms that are optimized to run efficiently against extremely large datasets in a distributed environment. With native support for bring-your-own-algorithms and frameworks, SageMaker offers flexible distributed training options that adjust to your specific workflows.

Currently, Aurora machine learning supports any SageMaker endpoint that can read and write comma-separated value format, through a `ContentType` of `text/csv`. The built-in SageMaker algorithms that currently accept this format are Random Cut Forest, Linear Learner, 1P, XGBoost, and 3P. If the algorithms return multiple outputs per item, the Aurora machine learning function returns only the first item. This first item is expected to be a representative result.

Aurora machine learning always invokes SageMaker endpoints in the same AWS Region as your Aurora cluster. Therefore, for a single-region Aurora cluster, always deploy the model in the same AWS Region as your Aurora MySQL cluster.

If you are using an Aurora global database, you set up the same integration between the services for each AWS Region that's part of the global database. In particular, make sure the following conditions are satisfied for all AWS Regions in the global database:

- Configure the appropriate IAM roles for accessing external services such as SageMaker, Amazon Comprehend, or Lambda for the global database cluster in each AWS Region.
- Ensure that all AWS Regions have the same trained SageMaker models deployed with the same endpoint names. Do so before running the `CREATE FUNCTION` statement for your Aurora machine learning function in the primary AWS Region. In a global database, all `CREATE FUNCTION` statements you run in the primary AWS Region are immediately run in all the secondary regions also.

To use models deployed in SageMaker for inference, you create user-defined functions using the familiar MySQL data definition language (DDL) statements for stored functions. Each stored function represents the SageMaker endpoint hosting the model. When you define such a function, you specify the input parameters to the model, the specific SageMaker endpoint to invoke, and the return type. The function returns the inference computed by the SageMaker endpoint after applying the model to the input parameters. All Aurora machine learning stored functions return numeric types or `VARCHAR`. You can use any numeric type except `BIT`. Other types, such as `JSON`, `BLOB`, `TEXT`, and `DATE` are not allowed. Use model input parameters that are the same as the input parameters that you exported to Amazon S3 for model training.

```

CREATE FUNCTION function_name (arg1 type1, arg2 type2, ...) -- variable number of arguments
    [DEFINER = user]                                         -- same as existing MySQL
CREATE FUNCTION
    RETURNS mysql_type          -- For example, INTEGER, REAL, ...
    [SQL SECURITY { DEFINER | INVOKER } ]                      -- same as existing MySQL
CREATE FUNCTION
    ALIAS AWS_SAGEMAKER_INVOKE_ENDPOINT -- ALIAS replaces the stored function body. Only
    AWS_SAGEMAKER_INVOKE_ENDPOINT is supported for now.
    ENDPOINT NAME 'endpoint_name'                                -- default is 10,000
    [MAX_BATCH_SIZE max_batch_size];

```

This is a variation of the existing `CREATE FUNCTION` DDL statement. In the `CREATE FUNCTION` statement that defines the SageMaker function, you don't specify a function body. Instead, you specify the new keyword `ALIAS` where the function body usually goes. Currently, Aurora machine learning only supports `aws_sagemaker_invoke_endpoint` for this extended syntax. You must specify the `endpoint_name` parameter. The optional parameter `max_batch_size` restricts the maximum number of inputs processed in an actual batched request to SageMaker. An SageMaker endpoint can have different characteristics for each model. The `max_batch_size` parameter can help to avoid an error caused by inputs that are too large, or to make SageMaker return a response more quickly. The `max_batch_size` parameter affects the size of an internal buffer used for ML request processing. Specifying too large a value for `max_batch_size` might cause substantial memory overhead on your DB instance.

We recommend leaving the `MANIFEST` setting at its default value of `OFF`. Although you can use the `MANIFEST ON` option, some SageMaker features can't directly use the CSV exported with this option. The manifest format is not compatible with the expected manifest format from SageMaker.

You create a separate stored function for each of your SageMaker models. This mapping of functions to models is required because an endpoint is associated with a specific model, and each model accepts different parameters. Using SQL types for the model inputs and the model output type helps to avoid type conversion errors passing data back and forth between the AWS services. You can control who can apply the model. You can also control the runtime characteristics by specifying a parameter representing the maximum batch size.

Currently, all Aurora machine learning functions have the `NOT DETERMINISTIC` property. If you don't specify that property explicitly, Aurora sets `NOT DETERMINISTIC` automatically. This requirement is because the ML model can be changed without any notification to the database. If that happens, calls

to an Aurora machine learning function might return different results for the same input within a single transaction.

You can't use the characteristics `CONTAINS SQL`, `NO SQL`, `READS SQL DATA`, or `MODIFIES SQL DATA` in your `CREATE FUNCTION` statement.

Following is an example usage of invoking an SageMaker endpoint to detect anomalies. There is an SageMaker endpoint `random-cut-forest-model`. The corresponding model is already trained by the `random-cut-forest` algorithm. For each input, the model returns an anomaly score. This example shows the data points whose score is greater than 3 standard deviations (approximately the 99.9th percentile) from the mean score.

```
create function anomaly_score(value real) returns real
  alias aws_sagemaker_invoke_endpoint endpoint name 'random-cut-forest-model-demo';

set @score_cutoff = (select avg(anomaly_score(value)) + 3 * std(anomaly_score(value)) from
  nyc_taxi);

select *, anomaly_detection(value) score from nyc_taxi
  where anomaly_detection(value) > @score_cutoff;
```

Character set requirement for SageMaker functions that return strings

We recommend specifying a character set of `utf8mb4` as the return type type for your SageMaker functions that return string values. If that isn't practical, use a large enough string length for the return type to hold a value represented in the `utf8mb4` character set. The following example shows how to declare the `utf8mb4` character set for your function.

```
CREATE FUNCTION my_ml_func(...) RETURNS VARCHAR(5) CHARSET utf8mb4 ALIAS ...
```

Currently, each SageMaker function that returns a string uses the character set `utf8mb4` for the return value. The return value uses this character set even if your ML function declares a different character set for its return type implicitly or explicitly. If your ML function declares a different character set for the return value, the returned data might be silently truncated if you store it in a table column that isn't long enough. For example, a query with a `DISTINCT` clause creates a temporary table. Thus, the ML function result might be truncated due to the way strings are handled internally during a query.

Using Amazon Comprehend for sentiment detection

Amazon Comprehend uses machine learning to find insights and relationships in textual data. You can use this AWS machine learning service even if you don't have any machine learning experience or expertise. Aurora machine learning uses Amazon Comprehend for sentiment analysis of text that is stored in your database. For example, using Amazon Comprehend you can analyze contact center call-in documents to detect sentiment and better understand caller-agent dynamics. You can find a further description in the post [Analyzing contact center calls](#) on the AWS Machine Learning blog.

You can also combine sentiment analysis with analysis of other information in your database using a single query. For example, you can detect the average sentiment of call-in center documents for issues that combine the following:

- Open for more than 30 days.
- About a specific product or feature.
- Made by the customers who have the greatest social media influence.

Using Amazon Comprehend from Aurora machine learning is as easy as calling a SQL function. Aurora machine learning provides two built-in Amazon Comprehend functions, `aws_comprehend_detect_sentiment()` and

`aws_comprehend_detect_sentiment()` to perform sentiment analysis through Amazon Comprehend. For each text fragment that you analyze, these functions help you to determine the sentiment and the confidence level.

```
-- Returns one of 'POSITIVE', 'NEGATIVE', 'NEUTRAL', 'MIXED'  
aws_comprehend_detect_sentiment(  
    input_text  
    ,language_code  
    [,max_batch_size] -- default is 25. should be greater than 0  
)  
  
-- Returns a double value that indicates confidence of the result of  
aws_comprehend_detect_sentiment.  
aws_comprehend_detect_sentiment_confidence(  
    input_text  
    ,language_code  
    [,max_batch_size] -- default is 25. should be greater than 0.  
)
```

The `max_batch_size` helps you to tune the performance of the Amazon Comprehend function calls. A large batch size trades off faster performance for greater memory usage on the Aurora cluster. For more information, see [Performance considerations for Aurora machine learning \(p. 1030\)](#).

For information about parameters and return types for the sentiment detection functions in Amazon Comprehend, see [DetectSentiment](#)

A typical Amazon Comprehend query looks for rows where the sentiment is a certain value, with a confidence level greater than a certain number. For example, the following query shows how you can determine the average sentiment of documents in your database. The query considers only documents where the confidence of the assessment is at least 80%.

```
SELECT AVG(CASE aws_comprehend_detect_sentiment(productTable.document, 'en')  
    WHEN 'POSITIVE' THEN 1.0  
    WHEN 'NEGATIVE' THEN -1.0  
    ELSE 0.0 END) AS avg_sentiment, COUNT(*) AS total  
FROM productTable  
WHERE productTable.productCode = 1302 AND  
    aws_comprehend_detect_sentiment_confidence(productTable.document, 'en') >= 0.80;
```

Note

Amazon Comprehend is currently available only in some AWS Regions. To check in which AWS Regions you can use Amazon Comprehend, see [the AWS Region table page](#).

Performance considerations for Aurora machine learning

Most of the work in an Aurora machine learning function call happens within the external ML service. This separation enables you to scale the resources for the machine learning service independent of your Aurora cluster. Within Aurora, you mostly focus on making the function calls themselves as efficient as possible.

Query cache

The Aurora MySQL query cache doesn't work for ML functions. Aurora MySQL doesn't store query results in the query cache for any SQL statements that call ML functions.

Batch optimization for Aurora machine learning function calls

The main Aurora machine learning performance aspect that you can influence from your Aurora cluster is the batch mode setting for calls to the Aurora machine learning stored functions. Machine learning

functions typically require substantial overhead, making it impractical to call an external service separately for each row. Aurora machine learning can minimize this overhead by combining the calls to the external Aurora machine learning service for many rows into a single batch. Aurora machine learning receives the responses for all the input rows, and delivers the responses, one row at a time, to the query as it runs. This optimization improves the throughput and latency of your Aurora queries without changing the results.

When you create an Aurora stored function that's connected to an SageMaker endpoint, you define the batch size parameter. This parameter influences how many rows are transferred for every underlying call to SageMaker. For queries that process large numbers of rows, the overhead to make a separate SageMaker call for each row can be substantial. The larger the data set processed by the stored procedure, the larger you can make the batch size.

If the batch mode optimization can be applied to an SageMaker function, you can tell by checking the query plan produced by the `EXPLAIN PLAN` statement. In this case, the `extra` column in the execution plan includes `Batched machine learning`. The following example shows a call to an SageMaker function that uses batch mode.

```
mysql> create function anomaly_score(val real) returns real alias
    aws_sagemaker_invoke_endpoint endpoint name 'my-rcf-model-20191126';
Query OK, 0 rows affected (0.01 sec)

mysql> explain select timestamp, value, anomaly_score(value) from nyc_taxi;
+-----+-----+-----+-----+-----+-----+
| id | select_type | table      | partitions | type | possible_keys | key   | key_len | ref   |
| rows | filtered | Extra          |           |       |             |        |         |        |
+-----+-----+-----+-----+-----+-----+
| 1 | SIMPLE      | nyc_taxi | NULL      | ALL  | NULL          | NULL  | NULL   | NULL  |
| 48 |    100.00 | Batched machine learning |
+-----+-----+-----+-----+-----+-----+
1 row in set, 1 warning (0.01 sec)
```

When you call one of the built-in Amazon Comprehend functions, you can control the batch size by specifying the optional `max_batch_size` parameter. This parameter restricts the maximum number of `input_text` values processed in each batch. By sending multiple items at once, it reduces the number of round trips between Aurora and Amazon Comprehend. Limiting the batch size is useful in situations such as a query with a `LIMIT` clause. By using a small value for `max_batch_size`, you can avoid invoking Amazon Comprehend more times than you have input texts.

The batch optimization for evaluating Aurora machine learning functions applies in the following cases:

- Function calls within the select list or the `WHERE` clause of `SELECT` statements. There are some exceptions, as described following.
- Function calls in the `VALUES` list of `INSERT` and `REPLACE` statements.
- ML functions in `SET` values in `UPDATE` statements.

```
INSERT INTO MY_TABLE (col1, col2, col3) VALUES
    (ML_FUNC(1), ML_FUNC(2), ML_FUNC(3)),
    (ML_FUNC(4), ML_FUNC(5), ML_FUNC(6));
UPDATE MY_TABLE SET col1 = ML_FUNC(col2), SET col3 = ML_FUNC(col4) WHERE ...;
```

Monitoring Aurora machine learning

To monitor the performance of Aurora machine learning batch operations, Aurora MySQL includes several global variables that you can query as follows.

```
show status like 'Aurora_ml%';
```

You can reset these status variables by using a `FLUSH STATUS` statement. Thus, all of the figures represent totals, averages, and so on, since the last time the variable was reset.

`Aurora_ml_logical_response_cnt`

The aggregate response count that Aurora MySQL receives from the ML services across all queries run by users of the DB instance.

`Aurora_ml_actual_request_cnt`

The aggregate request count that Aurora MySQL receives from the ML services across all queries run by users of the DB instance.

`Aurora_ml_actual_response_cnt`

The aggregate response count that Aurora MySQL receives from the ML services across all queries run by users of the DB instance.

`Aurora_ml_cache_hit_cnt`

The aggregate internal cache hit count that Aurora MySQL receives from the ML services across all queries run by users of the DB instance.

`Aurora_ml_single_request_cnt`

The aggregate count of ML functions that are evaluated by non-batch mode across all queries run by users of the DB instance.

For information about monitoring the performance of the SageMaker operations called from Aurora machine learning functions, see [Monitor Amazon SageMaker](#).

Limitations of Aurora machine learning

The following limitations apply to Aurora machine learning.

You can't use an Aurora machine learning function for a generated-always column. The same limitation applies to any Aurora MySQL stored function. Functions aren't compatible with this binary log (binlog) format. For information about generated columns, see [the MySQL documentation](#).

The setting `--binlog-format=STATEMENT` throws an exception for calls to Aurora machine learning functions. The reason for the error is that Aurora machine learning considers all ML functions to be nondeterministic, and nondeterministic stored functions aren't compatible with this binlog format. For information about this binlog format, see [the MySQL documentation](#).

Amazon Aurora MySQL lab mode

Aurora lab mode is used to enable Aurora features that are available in the current Aurora database version, but are not enabled by default. While Aurora lab mode features are not recommended for use in production DB clusters, you can use Aurora lab mode to enable these features for DB clusters in your development and test environments. For more information about Aurora features available when Aurora lab mode is enabled, see [Aurora lab mode features \(p. 1033\)](#).

The `aurora_lab_mode` parameter is an instance-level parameter that is in the default parameter group. The parameter is set to 0 (disabled) in the default parameter group. To enable Aurora lab

mode, create a custom parameter group, set the `aurora_lab_mode` parameter to 1 (enabled) in the custom parameter group, and modify one or more DB instances in your Aurora cluster to use the custom parameter group. Then connect to the appropriate instance endpoint to try the lab mode features. For information on modifying a DB parameter group, see [Modifying parameters in a DB parameter group \(p. 347\)](#). For information on parameter groups and Amazon Aurora, see [Aurora MySQL configuration parameters \(p. 1042\)](#).

Aurora lab mode features

The following table lists the Aurora features currently available when Aurora lab mode is enabled. You must enable Aurora lab mode before any of these features can be used.

Feature	Description
Scan Batching	Aurora MySQL scan batching speeds up in-memory, scan-oriented queries significantly. The feature boosts the performance of table full scans, index full scans, and index range scans by batch processing.
Hash Joins	This feature can improve query performance when you need to join a large amount of data by using an equijoin. It requires lab mode in Aurora MySQL version 1. You can use this feature without lab mode in Aurora MySQL version 2. For more information about using this feature, see Optimizing large Aurora MySQL join queries with hash joins (p. 1038) .
Fast DDL	This feature allows you to run an <code>ALTER TABLE <i>tbl_name</i> ADD COLUMN <i>col_name</i> <i>column_definition</i></code> operation nearly instantaneously. The operation completes without requiring the table to be copied and without materially impacting other DML statements. Since it does not consume temporary storage for a table copy, it makes DDL statements practical even for large tables on small instance classes. Fast DDL is currently only supported for adding a nullable column, without a default value, at the end of a table. For more information about using this feature, see Altering tables in Amazon Aurora using fast DDL (p. 832) .

Best practices with Amazon Aurora MySQL

This topic includes information on best practices and options for using or migrating data to an Amazon Aurora MySQL DB cluster. The information in this topic summarizes and reiterates some of the guidelines and procedures that you can find in [Managing an Amazon Aurora DB cluster \(p. 367\)](#).

Contents

- [Determining which DB instance you are connected to \(p. 1034\)](#)
- [Best practices for using AWS features with Aurora MySQL \(p. 1034\)](#)

- [Using T instance classes for development and testing \(p. 1034\)](#)
- [Invoking AWS Lambda functions using native functions \(p. 1036\)](#)
- [Best practices for Aurora MySQL performance and scaling \(p. 1036\)](#)
 - [Optimizing Amazon Aurora indexed join queries with asynchronous key prefetch \(p. 1036\)](#)
 - [Enabling asynchronous key prefetch \(p. 1036\)](#)
 - [Optimizing queries for asynchronous key prefetch \(p. 1037\)](#)
 - [Optimizing large Aurora MySQL join queries with hash joins \(p. 1038\)](#)
 - [Enabling hash joins \(p. 1039\)](#)
 - [Optimizing queries for hash joins \(p. 1039\)](#)
 - [Using Amazon Aurora to scale reads for your MySQL database \(p. 1040\)](#)
- [Best practices for Aurora MySQL high availability \(p. 1040\)](#)
 - [Using Amazon Aurora for Disaster Recovery with your MySQL databases \(p. 1040\)](#)
 - [Migrating from MySQL to Amazon Aurora MySQL with reduced downtime \(p. 1041\)](#)
- [Best practices for limiting certain MySQL features with Aurora MySQL \(p. 1041\)](#)
 - [Using multithreaded replication in Aurora MySQL version 3 \(p. 1041\)](#)
 - [Avoiding XA transactions with Amazon Aurora MySQL \(p. 1042\)](#)
 - [Keeping foreign keys turned on during DML statements \(p. 1042\)](#)

Determining which DB instance you are connected to

To determine which DB instance in an Aurora MySQL DB cluster a connection is connected to, check the `innodb_read_only` global variable, as shown in the following example.

```
SHOW GLOBAL VARIABLES LIKE 'innodb_read_only';
```

The `innodb_read_only` variable is set to `ON` if you are connected to a reader DB instance. This setting is `OFF` if you are connected to a writer DB instance, such as primary instance in a provisioned cluster.

This approach can be helpful if you want to add logic to your application code to balance the workload or to ensure that a write operation is using the correct connection. This technique only applies to Aurora clusters using single-master replication. For multi-master clusters, all the DB instances have the setting `innodb_read_only=OFF`.

Best practices for using AWS features with Aurora MySQL

You can apply the following best practices to use Aurora MySQL in combination with AWS aspects such as instance classes and other AWS services.

Topics

- [Using T instance classes for development and testing \(p. 1034\)](#)
- [Invoking AWS Lambda functions using native functions \(p. 1036\)](#)

Using T instance classes for development and testing

Amazon Aurora MySQL instances that use the `db.t2`, `db.t3`, or `db.t4g` DB instance classes are best suited for applications that do not support a high workload for an extended amount of time. The T instances are designed to provide moderate baseline performance and the capability to burst to

significantly higher performance as required by your workload. They are intended for workloads that don't use the full CPU often or consistently, but occasionally need to burst. We recommend only using the T DB instance classes for development and test servers, or other non-production servers. For more details on the T instance classes, see [Burstable performance instances](#).

If your Aurora cluster is larger than 40 TB, don't use the T instance classes. When your database has a large volume of data, the memory overhead for managing schema objects can exceed the capacity of a T instance.

Don't enable the MySQL Performance Schema on Amazon Aurora MySQL T instances. If the Performance Schema is enabled, the instance might run out of memory.

When you use a T instance as a DB instance in an Aurora MySQL DB cluster, we recommend the following:

- If you use a T instance as a DB instance class in your DB cluster, use the same DB instance class for all instances in your DB cluster. For example, if you use db.t2.medium for your writer instance, then we recommend that you use db.t2.medium for your reader instances also.
- Don't adjust any memory-related configuration settings, such as `innodb_buffer_pool_size`. Aurora uses a highly tuned set of default values for memory buffers on T instances. These special defaults are needed for Aurora to run on memory-constrained instances. If you change any memory-related settings on a T instance, you are much more likely to encounter out-of-memory conditions, even if your change is intended to increase buffer sizes.
- Monitor your CPU Credit Balance (`CPUCreditBalance`) to ensure that it is at a sustainable level. That is, CPU credits are being accumulated at the same rate as they are being used.

When you have exhausted the CPU credits for an instance, you see an immediate drop in the available CPU and an increase in the read and write latency for the instance. This situation results in a severe decrease in the overall performance of the instance.

If your CPU credit balance is not at a sustainable level, then we recommend that you modify your DB instance to use a one of the supported R DB instance classes (scale compute).

For more information on monitoring metrics, see [Viewing metrics in the Amazon RDS console \(p. 563\)](#).

- For your Aurora MySQL DB clusters using single-master replication, monitor the replica lag (`AuroraReplicaLag`) between the writer instance and the reader instances.

If a reader instance runs out of CPU credits before the writer instance does, the resulting lag can cause the reader instance to restart frequently. This result is common when an application has a heavy load of read operations distributed among reader instances, at the same time that the writer instance has a minimal load of write operations.

If you see a sustained increase in replica lag, make sure that your CPU credit balance for the reader instances in your DB cluster is not being exhausted.

If your CPU credit balance is not at a sustainable level, then we recommend that you modify your DB instance to use one of the supported R DB instance classes (scale compute).

- Keep the number of inserts per transaction below 1 million for DB clusters that have binary logging enabled.

If the DB cluster parameter group for your DB cluster has the `binlog_format` parameter set to a value other than OFF, then your DB cluster might experience out-of-memory conditions if the DB cluster receives transactions that contain over 1 million rows to insert. You can monitor the freeable memory (`FreeableMemory`) metric to determine if your DB cluster is running out of available memory. You then check the write operations (`VolumeWriteIOPS`) metric to see if a writer instance is receiving a heavy load of write operations. If this is the case, then we recommend that you update your application to limit the number of inserts in a transaction to less than 1 million. Alternatively, you can modify your instance to use one of the supported R DB instance classes (scale compute).

Invoking AWS Lambda functions using native functions

If you are using Amazon Aurora version 1.16 or later, we recommend using the native functions `lambda_sync` and `lambda_async` to invoke Lambda functions.

If you are using the deprecated `mysql.lambda_async` procedure, we recommend that you wrap calls to the `mysql.lambda_async` procedure in a stored procedure. You can call this stored procedure from different sources, such as triggers or client code. This approach can help to avoid impedance mismatch issues and make it easier for your database programmers to invoke Lambda functions.

For more information on invoking Lambda functions from Amazon Aurora, see [Invoking a Lambda function from an Amazon Aurora MySQL DB cluster \(p. 1010\)](#).

Best practices for Aurora MySQL performance and scaling

You can apply the following best practices to improve the performance and scalability of your Aurora MySQL clusters.

Topics

- [Optimizing Amazon Aurora indexed join queries with asynchronous key prefetch \(p. 1036\)](#)
- [Optimizing large Aurora MySQL join queries with hash joins \(p. 1038\)](#)
- [Using Amazon Aurora to scale reads for your MySQL database \(p. 1040\)](#)

Optimizing Amazon Aurora indexed join queries with asynchronous key prefetch

Note

The asynchronous key prefetch (AKP) feature is available for Amazon Aurora MySQL version 1.15 and later. For more information about Aurora MySQL versions, see [Database engine updates for Amazon Aurora MySQL \(p. 1082\)](#).

Amazon Aurora can use AKP to improve the performance of queries that join tables across indexes. This feature improves performance by anticipating the rows needed to run queries in which a JOIN query requires use of the Batched Key Access (BKA) Join algorithm and Multi-Range Read (MRR) optimization features. For more information about BKA and MRR, see [Block nested-loop and batched key access joins](#) and [Multi-range read optimization](#) in the MySQL documentation.

To take advantage of the AKP feature, a query must use both BKA and MRR. Typically, such a query occurs when the JOIN clause of a query uses a secondary index, but also needs some columns from the primary index. For example, you can use AKP when a JOIN clause represents an equijoin on index values between a small outer and large inner table, and the index is highly selective on the larger table. AKP works in concert with BKA and MRR to perform a secondary to primary index lookup during the evaluation of the JOIN clause. AKP identifies the rows required to run the query during the evaluation of the JOIN clause. It then uses a background thread to asynchronously load the pages containing those rows into memory before running the query.

Enabling asynchronous key prefetch

You can enable the AKP feature by setting `aurora_use_key_prefetch`, a MySQL server variable, to `on`. By default, this value is set to `on`. However, AKP cannot be enabled until you also enable the BKA Join algorithm and disable cost-based MRR functionality. To do so, you must set the following values for `optimizer_switch`, a MySQL server variable:

- Set `batched_key_access` to `on`. This value controls the use of the BKA Join algorithm. By default, this value is set to `off`.
- Set `mrr_cost_based` to `off`. This value controls the use of cost-based MRR functionality. By default, this value is set to `on`.

Currently, you can set these values only at the session level. The following example illustrates how to set these values to enable AKP for the current session by executing SET statements.

```
mysql> set @@session.aurora_use_key_prefetch=on;
mysql> set @@session.optimizer_switch='batched_key_access=on,mrr_cost_based=off';
```

Similarly, you can use SET statements to disable AKP and the BKA Join algorithm and re-enable cost-based MRR functionality for the current session, as shown in the following example.

```
mysql> set @@session.aurora_use_key_prefetch=off;
mysql> set @@session.optimizer_switch='batched_key_access=off,mrr_cost_based=on';
```

For more information about the `batched_key_access` and `mrr_cost_based` optimizer switches, see [Switchable optimizations](#) in the MySQL documentation.

Optimizing queries for asynchronous key prefetch

You can confirm whether a query can take advantage of the AKP feature. To do so, use the EXPLAIN statement with the EXTENDED keyword to profile the query before running it. The *EXPLAIN statement* provides information about the execution plan to use for a specified query.

In the output for the EXPLAIN statement, the Extra column describes additional information included with the execution plan. If the AKP feature applies to a table used in the query, this column includes one of the following values:

- Using Key Prefetching
- Using join buffer (Batched Key Access with Key Prefetching)

The following example shows use of EXPLAIN with EXTENDED to view the execution plan for a query that can take advantage of AKP.

```
mysql> explain extended select sql_no_cache
      ->     ps_partkey,
      ->     sum(ps_supplycost * ps_availqty) as value
-> from
->     partsupp,
->     supplier,
->     nation
-> where
->     ps_suppkey = s_suppkey
->     and s_nationkey = n_nationkey
->     and n_name = 'ETHIOPIA'
-> group by
->     ps_partkey having
->         sum(ps_supplycost * ps_availqty) > (
->             select
->                 sum(ps_supplycost * ps_availqty) * 0.0000003333
->             from
->                 partsupp,
->                 supplier,
```

```

->          nation
->      where
->          ps_suppkey = s_suppkey
->          and s_nationkey = n_nationkey
->          and n_name = 'ETHIOPIA'
->      )
-> order by
->     value desc;
+-----+-----+-----+-----+
+-----+-----+-----+
+-----+
| id | select_type | table      | type   | possible_keys | key           | key_len |
| ref          |             |           |         | rows        | filtered    | Extra      |
|       |             |           |         |              |             |            |
+-----+-----+-----+-----+
+-----+
| 1 | PRIMARY      | nation     | ALL    | PRIMARY      | NULL         | NULL        |
| NULL          |             |           |         | 25          | 100.00      | Using where; Using temporary; Using
| filesort      |             |           |         |              |             |            |
| 1 | PRIMARY      | supplier   | ref    | PRIMARY,i_s_nationkey | i_s_nationkey | 5          |
| dbt3_scale_10.nation.n_nationkey | 2057 | 100.00 | Using index
|             |             |           |         |              |             |            |
| 1 | PRIMARY      | partsupp   | ref    | i_ps_suppkey | i_ps_suppkey | 4          |
| dbt3_scale_10.supplier.s_suppkey | 42 | 100.00 | Using join buffer (Batched Key Access
| with Key Prefetching) |
| 2 | SUBQUERY     | nation     | ALL    | PRIMARY      | NULL         | NULL        |
| NULL          |             |           |         | 25          | 100.00      | Using where
|             |             |           |         |              |             |            |
| 2 | SUBQUERY     | supplier   | ref    | PRIMARY,i_s_nationkey | i_s_nationkey | 5          |
| dbt3_scale_10.nation.n_nationkey | 2057 | 100.00 | Using index
|             |             |           |         |              |             |            |
| 2 | SUBQUERY     | partsupp   | ref    | i_ps_suppkey | i_ps_suppkey | 4          |
| dbt3_scale_10.supplier.s_suppkey | 42 | 100.00 | Using join buffer (Batched Key Access
| with Key Prefetching) |
+-----+-----+-----+-----+
+-----+
+-----+
6 rows in set, 1 warning (0.00 sec)

```

For more information about the extended EXPLAIN output format, see [Extended EXPLAIN output format](#) in the MySQL product documentation.

Optimizing large Aurora MySQL join queries with hash joins

When you need to join a large amount of data by using an equijoin, a hash join can improve query performance. You can enable hash joins for Aurora MySQL.

A hash join column can be any complex expression. In a hash join column, you can compare across data types in the following ways:

- You can compare anything across the category of precise numeric data types, such as `int`, `bigint`, `numeric`, and `bit`.
- You can compare anything across the category of approximate numeric data types, such as `float` and `double`.
- You can compare items across string types if the string types have the same character set and collation.
- You can compare items with date and timestamp data types if the types are the same.

Note

Data types in different categories cannot compare.

The following restrictions apply to hash joins for Aurora MySQL:

- Left-right outer joins are not supported.
- Semijoins such as subqueries are not supported, unless the subqueries are materialized first.
- Multiple-table updates or deletes are not supported.

Note

Single-table updates or deletes are supported.

- BLOB and spatial data type columns cannot be join columns in a hash join.

Enabling hash joins

To enable hash joins, set the MySQL server variable `optimizer_switch` to `hash_join=on` (Aurora MySQL version 1 and 2) or `block_nested_loop=on` (Aurora MySQL version 3). Hash joins are turned on by default in Aurora MySQL version 3. This optimization is turned off by default in Aurora MySQL version 1 and 2. The following example illustrates how to enable hash joins. You can issue the statement `select @@optimizer_switch` first to see what other settings are present in the `SET` parameter string. Updating one setting in the `optimizer_switch` parameter doesn't erase or modify the other settings.

```
For Aurora MySQL version 1 and 2:  
mysql> SET optimizer_switch='hash_join=on';  
  
For Aurora MySQL version 3:  
mysql> SET optimizer_switch='block_nested_loop=on';
```

With this setting, the optimizer chooses to use a hash join based on cost, query characteristics, and resource availability. If the cost estimation is incorrect, you can force the optimizer to choose a hash join. You do so by setting `hash_join_cost_based`, a MySQL server variable, to `off`. The following example illustrates how to force the optimizer to choose a hash join.

```
mysql> SET optimizer_switch='hash_join_cost_based=off';
```

Note

For Aurora MySQL version 3, hash join support is available in all minor versions and is turned on by default.

For Aurora MySQL version 2, hash join support is available in version 2.06 and higher. In Aurora MySQL version 2, the hash join feature is always controlled by the `optimizer_switch` value. Prior to Aurora MySQL version 1.22, the way to enable hash joins in Aurora MySQL version 1 is by enabling the `aurora_lab_mode` session-level setting. In those Aurora MySQL versions, the `optimizer_switch` setting for hash joins is enabled by default and you only need to enable `aurora_lab_mode`.

Optimizing queries for hash joins

To find out whether a query can take advantage of a hash join, use the `EXPLAIN` statement to profile the query first. The `EXPLAIN` statement provides information about the execution plan to use for a specified query.

In the output for the `EXPLAIN` statement, the `Extra` column describes additional information included with the execution plan. If a hash join applies to the tables used in the query, this column includes values similar to the following:

- Using `where`; Using join buffer (Hash Join Outer table `table1_name`)

- Using where; Using join buffer (Hash Join Inner table `table2_name`)

The following example shows the use of EXPLAIN to view the execution plan for a hash join query.

```
mysql> explain SELECT sql_no_cache * FROM hj_small, hj_big, hj_big2
      ->      WHERE hj_small.col1 = hj_big.col1 and hj_big.col1=hj_big2.col1 ORDER BY 1;
+-----+-----+-----+-----+-----+-----+
| id | select_type | table     | type   | possible_keys | key    | key_len | ref   | rows | Extra
+-----+-----+-----+-----+-----+-----+
| 1  | SIMPLE      | hj_small | ALL    | NULL        | NULL   | NULL    | NULL  | 6   | Using temporary; Using filesort
| 1  | SIMPLE      | hj_big   | ALL    | NULL        | NULL   | NULL    | NULL  | 10  | Using where; Using join buffer (Hash Join Outer table hj_big)
| 1  | SIMPLE      | hj_big2  | ALL    | NULL        | NULL   | NULL    | NULL  | 15  | Using where; Using join buffer (Hash Join Inner table hj_big2)
+-----+-----+-----+-----+-----+-----+
3 rows in set (0.04 sec)
```

In the output, the Hash Join Inner table is the table used to build hash table, and the Hash Join Outer table is the table that is used to probe the hash table.

For more information about the extended EXPLAIN output format, see [Extended EXPLAIN output format](#) in the MySQL product documentation.

In Aurora MySQL 2.08 and higher, you can use SQL hints to influence whether a query uses hash join or not, and which tables to use for the build and probe sides of the join. For details, see [Aurora MySQL hints \(p. 1074\)](#).

Using Amazon Aurora to scale reads for your MySQL database

You can use Amazon Aurora with your MySQL DB instance to take advantage of the read scaling capabilities of Amazon Aurora and expand the read workload for your MySQL DB instance. To use Aurora to read scale your MySQL DB instance, create an Aurora MySQL DB cluster and make it a read replica of your MySQL DB instance. Then connect to the Aurora MySQL cluster to process the read queries. The source database can be an RDS for MySQL DB instance, or a MySQL database running external to Amazon RDS. For more information, see [Using Amazon Aurora to scale reads for your MySQL database \(p. 945\)](#).

Best practices for Aurora MySQL high availability

You can apply the following best practices to improve the availability of your Aurora MySQL clusters.

Topics

- [Using Amazon Aurora for Disaster Recovery with your MySQL databases \(p. 1040\)](#)
- [Migrating from MySQL to Amazon Aurora MySQL with reduced downtime \(p. 1041\)](#)

Using Amazon Aurora for Disaster Recovery with your MySQL databases

You can use Amazon Aurora with your MySQL DB instance to create an offsite backup for disaster recovery. To use Aurora for disaster recovery of your MySQL DB instance, create an Amazon Aurora DB

cluster and make it a read replica of your MySQL DB instance. This applies to an RDS for MySQL DB instance, or a MySQL database running external to Amazon RDS.

Important

When you set up replication between a MySQL DB instance and an Amazon Aurora MySQL DB cluster, you should monitor the replication to ensure that it remains healthy and repair it if necessary.

For instructions on how to create an Amazon Aurora MySQL DB cluster and make it a read replica of your MySQL DB instance, follow the procedure in [Using Amazon Aurora to scale reads for your MySQL database \(p. 1040\)](#).

Migrating from MySQL to Amazon Aurora MySQL with reduced downtime

When importing data from a MySQL database that supports a live application to an Amazon Aurora MySQL DB cluster, you might want to reduce the time that service is interrupted while you migrate. To do so, you can use the procedure documented in [Importing data to a MySQL or MariaDB DB instance with reduced downtime](#) in the *Amazon Relational Database Service User Guide*. This procedure can especially help if you are working with a very large database. You can use the procedure to reduce the cost of the import by minimizing the amount of data that is passed across the network to AWS.

The procedure lists steps to transfer a copy of your database data to an Amazon EC2 instance and import the data into a new RDS for MySQL DB instance. Because Amazon Aurora is compatible with MySQL, you can instead use an Amazon Aurora DB cluster for the target Amazon RDS MySQL DB instance.

Best practices for limiting certain MySQL features with Aurora MySQL

The following features are available in Aurora MySQL for MySQL compatibility. However, they have performance, scalability, or stability issues in the Aurora environment. Thus, we recommend that you limit your use of these features. For example, we recommend that you don't use certain features for production Aurora deployments.

Topics

- [Using multithreaded replication in Aurora MySQL version 3 \(p. 1041\)](#)
- [Avoiding XA transactions with Amazon Aurora MySQL \(p. 1042\)](#)
- [Keeping foreign keys turned on during DML statements \(p. 1042\)](#)

Using multithreaded replication in Aurora MySQL version 3

By default, Aurora uses single-threaded replication when an Aurora MySQL DB cluster is used as a read replica for binary log replication.

Although Aurora MySQL doesn't prohibit multithreaded replication, this feature is only supported in Aurora MySQL version 3 and higher.

Aurora MySQL version 1 and 2 inherited several issues regarding multithreaded replication from MySQL. For those versions, we recommend that you don't use multithreaded replication in production.

If you do use multithreaded replication, we recommend that you test any use thoroughly.

For more information about using replication in Amazon Aurora, see [Replication with Amazon Aurora \(p. 70\)](#). For information about multithreaded replication in Aurora MySQL version 3, see [Multithreaded binary log replication \(Aurora MySQL version 3 and higher\) \(p. 948\)](#).

Avoiding XA transactions with Amazon Aurora MySQL

We recommend that you don't use eXtended Architecture (XA) transactions with Aurora MySQL, because they can cause long recovery times if the XA was in the `PREPARED` state. If you must use XA transactions with Aurora MySQL, follow these best practices:

- Don't leave an XA transaction open in the `PREPARED` state.
- Keep XA transactions as small as possible.

For more information about using XA transactions with MySQL, see [XA transactions](#) in the MySQL documentation.

Keeping foreign keys turned on during DML statements

We strongly recommend that you don't run any data definition language (DDL) statements when the `foreign_key_checks` variable is set to 0 (off).

If you need to insert or update rows that require a transient violation of foreign keys, follow these steps:

1. Set `foreign_key_checks` to 0.
2. Make your data manipulation language (DML) changes.
3. Make sure that your completed changes don't violate any foreign key constraints.
4. Set `foreign_key_checks` to 1 (on).

In addition, follow these other best practices for foreign key constraints:

- Make sure that your client applications don't set the `foreign_key_checks` variable to 0 as a part of the `init_connect` variable.
- If a restore from a logical backup such as `mysqldump` fails or is incomplete, make sure that `foreign_key_checks` is set to 1 before starting any other operations in the same session. A logical backup sets `foreign_key_checks` to 0 when it starts.

Amazon Aurora MySQL reference

This reference includes information about Aurora MySQL parameters, status variables, and general SQL extensions or differences from the community MySQL database engine.

Topics

- [Aurora MySQL configuration parameters \(p. 1042\)](#)
- [MySQL parameters that don't apply to Aurora MySQL \(p. 1061\)](#)
- [MySQL status variables that don't apply to Aurora MySQL \(p. 1062\)](#)
- [Aurora MySQL wait events \(p. 1063\)](#)
- [Aurora MySQL thread states \(p. 1067\)](#)
- [Aurora MySQL isolation levels \(p. 1070\)](#)
- [Aurora MySQL hints \(p. 1074\)](#)
- [Aurora MySQL stored procedures \(p. 1076\)](#)

Aurora MySQL configuration parameters

You manage your Amazon Aurora MySQL DB cluster in the same way that you manage other Amazon RDS DB instances, by using parameters in a DB parameter group. Amazon Aurora differs from other DB engines in that you have a DB cluster that contains multiple DB instances. As a result, some of the parameters that you use to manage your Aurora MySQL DB cluster apply to the entire cluster. Other parameters apply only to a particular DB instance in the DB cluster.

To manage cluster-level parameters, you use DB cluster parameter groups. To manage instance-level parameters, you use DB parameter groups. Each DB instance in an Aurora MySQL DB cluster is compatible with the MySQL database engine. However, you apply some of the MySQL database engine parameters at the cluster level, and you manage these parameters using DB cluster parameter groups. You can't find cluster-level parameters in the DB parameter group for an instance in an Aurora DB cluster. A list of cluster-level parameters appears later in this topic.

You can manage both cluster-level and instance-level parameters using the AWS Management Console, the AWS CLI, or the Amazon RDS API. You use separate commands for managing cluster-level parameters and instance-level parameters. For example, you can use the [modify-db-cluster-parameter-group](#) CLI command to manage cluster-level parameters in a DB cluster parameter group. You can use the [modify-db-parameter-group](#) CLI command to manage instance-level parameters in a DB parameter group for a DB instance in a DB cluster.

You can view both cluster-level and instance-level parameters in the console, or by using the CLI or RDS API. For example, you can use the [describe-db-cluster-parameters](#) AWS CLI command to view cluster-level parameters in a DB cluster parameter group. You can use the [describe-db-parameters](#) CLI command to view instance-level parameters in a DB parameter group for a DB instance in a DB cluster.

Note

Each [default parameter group \(p. 339\)](#) contains the default values for all parameters in the parameter group. If the parameter has "engine default" for this value, see the version-specific MySQL or PostgreSQL documentation for the actual default value.

For more information on DB parameter groups, see [Working with DB parameter groups and DB cluster parameter groups \(p. 339\)](#). For rules and restrictions for Aurora Serverless clusters, see [Parameter groups and Aurora Serverless v1 \(p. 156\)](#).

Topics

- [Cluster-level parameters \(p. 1043\)](#)
- [Instance-level parameters \(p. 1049\)](#)

Cluster-level parameters

The following table shows all of the parameters that apply to the entire Aurora MySQL DB cluster.

Parameter name	Modifiable	Notes
aurora_binlog_read_buffer_size	Yes	Only affects clusters that use binary log (binlog) replication. For information about binlog replication, see Replication between Aurora and MySQL or between Aurora and another Aurora DB cluster (binary log replication) (p. 932) . Removed from Aurora MySQL version 3.
aurora_binlog_replication_max_yield_time	Yes	Only affects clusters that use binary log (binlog) replication. For information about binlog replication, see Replication between Aurora and MySQL or between Aurora

Parameter name	Modifiable	Notes
		and another Aurora DB cluster (binary log replication) (p. 932) .
aurora_binlog_use_large_read_buffer	Yes	Only affects clusters that use binary log (binlog) replication. For information about binlog replication, see Replication between Aurora and MySQL or between Aurora and another Aurora DB cluster (binary log replication) (p. 932) . Removed from Aurora MySQL version 3.
aurora_enable_replica_log_compression	Yes	For more information, see Performance considerations for Amazon Aurora MySQL replication (p. 920) . Doesn't apply to clusters that are part of an Aurora global database. Removed from Aurora MySQL version 3.
aurora_enable_repl_bin_log_filtering	Yes	For more information, see Performance considerations for Amazon Aurora MySQL replication (p. 920) . Doesn't apply to clusters that are part of an Aurora global database. Removed from Aurora MySQL version 3.
aurora_enable_zdr	Yes	This setting is turned on by default in Aurora MySQL 2.10 and higher. For more information, see Zero-downtime restart (ZDR) for Amazon Aurora MySQL (p. 920) .
aurora_load_from_s3_role	Yes	For more information, see Loading data into an Amazon Aurora MySQL DB cluster from text files in an Amazon S3 bucket (p. 997) . Currently not available in Aurora MySQL version 3.
aurora_select_into_s3_role	Yes	For more information, see Saving data from an Amazon Aurora MySQL DB cluster into text files in an Amazon S3 bucket (p. 1004) . Currently not available in Aurora MySQL version 3.
auto_increment_increment	Yes	
auto_increment_offset	Yes	
aws_default_lambda_role	Yes	For more information, see Invoking a Lambda function from an Amazon Aurora MySQL DB cluster (p. 1010) .
aws_default_s3_role	Yes	

Parameter name	Modifiable	Notes
binlog_checksum	Yes	The AWS CLI and RDS API report a value of <code>None</code> if this parameter isn't set. In that case, Aurora MySQL uses the engine default value, which is <code>CRC32</code> . This is different than the explicit setting of <code>NONE</code> , which turns off the checksum. For a bug fix related to this parameter, see Aurora MySQL database engine updates 2020-09-02 (version 1.23.0) (p. 1200) and Aurora MySQL database engine updates 2020-03-05 (version 1.22.2) (p. 1206) .
binlog-do-db	Yes	This parameter applies to Aurora MySQL version 3 and higher.
binlog_format	Yes	For more information, see Replication between Aurora and MySQL or between Aurora and another Aurora DB cluster (binary log replication) (p. 932) .
binlog_group_commit_sync_delay	Yes	This parameter applies to Aurora MySQL version 3 and higher.
binlog_group_commit_sync_no_delay_Yes_nt	Yes	This parameter applies to Aurora MySQL version 3 and higher.
binlog-ignore-db	Yes	This parameter applies to Aurora MySQL version 3 and higher.
binlog_row_image	No	
binlog_row_metadata	Yes	This parameter applies to Aurora MySQL version 3 and higher.
binlog_row_value_options	Yes	This parameter applies to Aurora MySQL version 3 and higher.
binlog_rows_query_log_events	Yes	
binlog_transaction_compression	Yes	This parameter applies to Aurora MySQL version 3 and higher.
binlog_transaction_compression_level_Yes_zstd	Yes	This parameter applies to Aurora MySQL version 3 and higher.
binlog_transaction_dependency_hist_Yes_size	Yes	This parameter applies to Aurora MySQL version 3 and higher.
binlog_transaction_dependency_track_Yes_ng	Yes	This parameter applies to Aurora MySQL version 3 and higher.
character-set-client-handshake	Yes	
character_set_client	Yes	
character_set_connection	Yes	
character_set_database	Yes	

Parameter name	Modifiable	Notes
<code>character_set_filesystem</code>	Yes	
<code>character_set_results</code>	Yes	
<code>character_set_server</code>	Yes	
<code>collation_connection</code>	Yes	
<code>collation_server</code>	Yes	
<code>completion_type</code>	Yes	
<code>default_storage_engine</code>	No	Aurora MySQL clusters use the InnoDB storage engine for all of your data.
<code>enforce_gtid_consistency</code>	Sometimes	Modifiable in Aurora MySQL version 2.04 and later.
<code>gtid-mode</code>	Sometimes	Modifiable in Aurora MySQL version 2.04 and later.
<code>innodb_autoinc_lock_mode</code>	Yes	
<code>innodb_checksums</code>	No	Removed from Aurora MySQL version 3.
<code>innodb_cmp_per_index_enabled</code>	Yes	
<code>innodb_commit_concurrency</code>	Yes	
<code>innodb_data_home_dir</code>	No	Aurora MySQL uses managed instances where you don't access the file system directly.
<code>innodb_file_per_table</code>	Yes	
<code>innodb_flush_log_at_trx_commit</code>	Yes (Aurora MySQL version 1 and 2), No (Aurora MySQL version 3)	For Aurora MySQL version 3, Aurora always uses the default value of 1.
<code>innodb_ft_max_token_size</code>	Yes	
<code>innodb_ft_min_token_size</code>	Yes	
<code>innodb_ft_num_word_optimize</code>	Yes	
<code>innodb_ft_sort_pll_degree</code>	Yes	
<code>innodb_online_alter_log_max_size</code>	Yes	
<code>innodb_optimize_fulltext_only</code>	Yes	
<code>innodb_page_size</code>	No	
<code>innodb_purge_batch_size</code>	Yes	

Parameter name	Modifiable	Notes
innodb_purge_threads	Yes	
innodb_rollback_on_timeout	Yes	
innodb_rollback_segments	Yes	
innodb_spin_wait_delay	Yes	
innodb_strict_mode	Yes	
innodb_support_xa	Yes	Removed from Aurora MySQL version 3.
innodb_sync_array_size	Yes	
innodb_sync_spin_loops	Yes	
innodb_table_locks	Yes	
innodb_undo_directory	No	Aurora MySQL uses managed instances where you don't access the file system directly.
innodb_undo_logs	Yes	Removed from Aurora MySQL version 3.
innodb_undo_tablespaces	No	Removed from Aurora MySQL version 3.
internal_tmp_mem_storage_engine	Yes	This parameter applies to Aurora MySQL version 3 and higher.
lc_time_names	Yes	
lower_case_table_names	Yes (Aurora MySQL version 1 and 2), only at cluster creation time (Aurora MySQL version 3)	In Aurora MySQL version 2.10 and higher 2.x versions, make sure to reboot all reader instances after changing this setting and rebooting the writer instance. For details, see Rebooting an Aurora MySQL cluster (version 2.10 and higher) (p. 452) . In Aurora MySQL version 3, the value of this parameter is set permanently at the time the cluster is created. If you use a nondefault value for this option, set up your Aurora MySQL version 3 custom parameter group before upgrading, and specify the parameter group during the snapshot restore operation that creates the version 3 cluster.
master-info-repository	Yes	Removed from Aurora MySQL version 3.
master_verify_checksum	Yes	Aurora MySQL version 1 and 2. Use <code>source_verify_checksum</code> in Aurora MySQL version 3.
partial_revokes	No	This parameter applies to Aurora MySQL version 3 and higher.

Parameter name	Modifiable	Notes
<code>relay-log-space-limit</code>	Yes	This parameter applies to Aurora MySQL version 3 and higher.
<code>replica_preserve_commit_order</code>	Yes	This parameter applies to Aurora MySQL version 3 and higher.
<code>replica_transaction_retries</code>	Yes	This parameter applies to Aurora MySQL version 3 and higher.
<code>replicate-do-db</code>	Yes	This parameter applies to Aurora MySQL version 3 and higher.
<code>replicate-do-table</code>	Yes	This parameter applies to Aurora MySQL version 3 and higher.
<code>replicate-ignore-db</code>	Yes	This parameter applies to Aurora MySQL version 3 and higher.
<code>replicate-ignore-table</code>	Yes	This parameter applies to Aurora MySQL version 3 and higher.
<code>replicate-wild-do-table</code>	Yes	This parameter applies to Aurora MySQL version 3 and higher.
<code>replicate-wild-ignore-table</code>	Yes	This parameter applies to Aurora MySQL version 3 and higher.
<code>require_secure_transport</code>	Yes	For more information, see Using SSL/TLS with Aurora MySQL DB clusters (p. 775) .
<code>rpl_read_size</code>	Yes	This parameter applies to Aurora MySQL version 3 and higher.
<code>server_audit_events</code>	Yes	
<code>server_audit_excl_users</code>	Yes	
<code>server_audit_incl_users</code>	Yes	
<code>server_audit_logging</code>	Yes	For instructions on uploading the logs to Amazon CloudWatch Logs, see Publishing Amazon Aurora MySQL logs to Amazon CloudWatch Logs (p. 1017) .
<code>server_id</code>	No	
<code>skip-character-set-client-handshake</code>	Yes	
<code>skip_name_resolve</code>	No	
<code>slave-skip-errors</code>	Yes	Only applies to Aurora MySQL version 2 clusters, with MySQL 5.7 compatibility.
<code>source_verify_checksum</code>	Yes	Aurora MySQL version 3 and higher
<code>sync_frm</code>	Yes	Removed from Aurora MySQL version 3.

Parameter name	Modifiable	Notes
time_zone	Yes	
tls_version	Yes	For more information, see TLS versions for Aurora MySQL (p. 776) .

Instance-level parameters

The following table shows all of the parameters that apply to a specific DB instance in an Aurora MySQL DB cluster.

Parameter name	Modifiable	Notes
activate_all_roles_on_login	Yes	This parameter applies to Aurora MySQL version 3 and higher.
allow-suspicious-udfs	No	
aurora_lab_mode	Yes	For more information, see Amazon Aurora MySQL lab mode (p. 1032) . Removed from Aurora MySQL version 3.
aurora_oom_response	Yes	This parameter only applies to Aurora MySQL version 1.18 and higher. It isn't used in Aurora MySQL version 2 or 3. For more information, see Amazon Aurora MySQL out of memory issues (p. 1817) .
aurora_parallel_query	Yes	Set to ON to turn on parallel query in Aurora MySQL versions 1.23 and 2.09 or higher. The old aurora_pq parameter isn't used in these versions. For more information, see Working with parallel query for Amazon Aurora MySQL (p. 881) .
aurora_pq	Yes	Set to OFF to turn off parallel query for specific DB instances in Aurora MySQL versions before 1.23 and 2.09. In 1.23 and 2.09 or higher, turn parallel query on and off with aurora_parallel_query instead. For more information, see Working with parallel query for Amazon Aurora MySQL (p. 881) .
autocommit	Yes	
automatic_sp_privileges	Yes	
back_log	Yes	
basedir	No	Aurora MySQL uses managed instances where you don't access the file system directly.
binlog_cache_size	Yes	

Parameter name	Modifiable	Notes
binlog_max_flush_queue_time	Yes	
binlog_order_commits	Yes	
binlog_stmt_cache_size	Yes	
binlog_transaction_compression	Yes	This parameter applies to Aurora MySQL version 3 and higher.
binlog_transaction_compression_level	Yes	This parameter applies to Aurora MySQL version 3 and higher.
bulk_insert_buffer_size	Yes	
concurrent_insert	Yes	
connect_timeout	Yes	
core-file	No	Aurora MySQL uses managed instances where you don't access the file system directly.
datadir	No	Aurora MySQL uses managed instances where you don't access the file system directly.
default_authentication_plugin	No	This parameter applies to Aurora MySQL version 3 and higher.
default_time_zone	No	
default_tmp_storage_engine	Yes	
default_week_format	Yes	
delay_key_write	Yes	
delayed_insert_limit	Yes	
delayed_insert_timeout	Yes	
delayed_queue_size	Yes	
div_precision_increment	Yes	
end_markers_in_json	Yes	
eq_range_index_dive_limit	Yes	
event_scheduler	Yes	
explicit_defaults_for_timestamp	Yes	
flush	No	
flush_time	Yes	
ft_boolean_syntax	No	
ft_max_word_len	Yes	

Parameter name	Modifiable	Notes
<code>ft_min_word_len</code>	Yes	
<code>ft_query_expansion_limit</code>	Yes	
<code>ft_stopword_file</code>	Yes	
<code>general_log</code>	Yes	For instructions on uploading the logs to CloudWatch Logs, see Publishing Amazon Aurora MySQL logs to Amazon CloudWatch Logs (p. 1017) .
<code>general_log_file</code>	No	Aurora MySQL uses managed instances where you don't access the file system directly.
<code>group_concat_max_len</code>	Yes	
<code>host_cache_size</code>	Yes	
<code>init_connect</code>	Yes	
<code>innodb_adaptive_hash_index</code>	Yes	
<code>innodb_adaptive_max_sleep_delay</code>	Yes	Modifying this parameter has no effect, because <code>innodb_thread_concurrency</code> is always 0 for Aurora.
<code>innodb_autoextend_increment</code>	Yes	
<code>innodb_buffer_pool_dump_at_shutdown</code>	No	
<code>innodb_buffer_pool_dump_now</code>	No	
<code>innodb_buffer_pool_filename</code>	No	
<code>innodb_buffer_pool_load_abort</code>	No	
<code>innodb_buffer_pool_load_at_startup</code>	No	
<code>innodb_buffer_pool_load_now</code>	No	
<code>innodb_buffer_pool_size</code>	Yes	The default value is represented by a formula. For details about how the <code>DBInstanceClassMemory</code> value in the formula is calculated, see DB parameter formula variables (p. 363) .
<code>innodb_change_buffer_max_size</code>	No	Aurora MySQL doesn't use the InnoDB change buffer at all.
<code>innodb_compression_failure_threshold_pct</code>	Yes	
<code>innodb_compression_level</code>	Yes	
<code>innodb_compression_pad_pct_max</code>	Yes	
<code>innodb_concurrency_tickets</code>	Yes	Modifying this parameter has no effect, because <code>innodb_thread_concurrency</code> is always 0 for Aurora.

Parameter name	Modifiable	Notes
innodb_file_format	Yes	Removed from Aurora MySQL version 3.
innodb_flush_log_at_timeout	No	
innodb_flushing_avg_loops	No	
innodb_force_load_corrupted	No	
innodb_ft_aux_table	Yes	
innodb_ft_cache_size	Yes	
innodb_ft_enable_stopword	Yes	
innodb_ft_server_stopword_table	Yes	
innodb_ft_user_stopword_table	Yes	
innodb_large_prefix	Yes	Removed from Aurora MySQL version 3.
innodb_lock_wait_timeout	Yes	
innodb_log_compressed_pages	No	
innodb_lru_scan_depth	Yes	
innodb_max_purge_lag	Yes	
innodb_max_purge_lag_delay	Yes	
innodb_monitor_disable	Yes	
innodb_monitor_enable	Yes	
innodb_monitor_reset	Yes	
innodb_monitor_reset_all	Yes	
innodb_old_blocks_pct	Yes	
innodb_old_blocks_time	Yes	
innodb_open_files	Yes	
innodb_print_all_deadlocks	Yes	
innodb_random_read_ahead	Yes	
innodb_read_ahead_threshold	Yes	
innodb_read_io_threads	No	
innodb_read_only	No	Aurora MySQL manages the read-only and read/write state of DB instances based on the type of cluster. For example, a provisioned cluster has one read/write DB instance (the <i>primary instance</i>) and any other instances in the cluster are read-only (the Aurora Replicas).
innodb_replication_delay	Yes	

Parameter name	Modifiable	Notes
innodb_sort_buffer_size	Yes	
innodb_stats_auto_recalc	Yes	
innodb_stats_method	Yes	
innodb_stats_on_metadata	Yes	
innodb_stats_persistent	Yes	
innodb_stats_persistent_sample_pages	Yes	
innodb_stats_transient_sample_pages	Yes	
innodb_thread_concurrency	No	
innodb_thread_sleep_delay	Yes	Modifying this parameter has no effect, because <code>innodb_thread_concurrency</code> is always 0 for Aurora.
interactive_timeout	Yes	Aurora evaluates the minimum value of <code>interactive_timeout</code> and <code>wait_timeout</code> . It then uses that minimum as the timeout to end all idle sessions, both interactive and noninteractive.
internal_tmp_mem_storage_engine	Yes	This parameter applies to Aurora MySQL version 3 and higher.
join_buffer_size	Yes	
keep_files_on_create	Yes	
key_buffer_size	Yes	
key_cache_age_threshold	Yes	
key_cache_block_size	Yes	
key_cache_division_limit	Yes	
local_infile	Yes	
lock_wait_timeout	Yes	
log-bin	No	Setting <code>binlog_format</code> to STATEMENT, MIXED, or ROW automatically sets <code>log-bin</code> to ON. Setting <code>binlog_format</code> to OFF automatically sets <code>log-bin</code> to OFF. For more information, see Replication between Aurora and MySQL or between Aurora and another Aurora DB cluster (binary log replication) (p. 932) .
log_bin_trust_function_creators	Yes	
log_bin_use_v1_row_events	Yes	Removed from Aurora MySQL version 3.

Parameter name	Modifiable	Notes
log_error	No	
log_output	Yes	
log_queries_not_using_indexes	Yes	
log_slave_updates	No	Aurora MySQL version 1 and 2. Use log_replica_updates in Aurora MySQL version 3.
log_replica_updates	No	Aurora MySQL version 3 and higher
log_throttle_queries_not_using_indexes	Yes	
log_warnings	Yes	Removed from Aurora MySQL version 3.
long_query_time	Yes	
low_priority_updates	Yes	
max_allowed_packet	Yes	
max_binlog_cache_size	Yes	
max_binlog_size	No	
max_binlog_stmt_cache_size	Yes	
max_connect_errors	Yes	
max_connections	Yes	The default value is represented by a formula. For details about how the DBInstanceClassMemory value in the formula is calculated, see DB parameter formula variables (p. 363) . For the default values depending on the instance class, see Maximum connections to an Aurora MySQL DB instance (p. 813) .
max_delayed_threads	Yes	
max_error_count	Yes	
max_heap_table_size	Yes	
max_insert_delayed_threads	Yes	
max_join_size	Yes	
max_length_for_sort_data	Yes	Removed from Aurora MySQL version 3.
max_prepared_stmt_count	Yes	
max_seeks_for_key	Yes	
max_sort_length	Yes	
max_sp_recursion_depth	Yes	
max_tmp_tables	Yes	Removed from Aurora MySQL version 3.

Parameter name	Modifiable	Notes
max_user_connections	Yes	
max_write_lock_count	Yes	
metadata_locks_cache_size	Yes	Removed from Aurora MySQL version 3.
min_examined_row_limit	Yes	
myisam_data_pointer_size	Yes	
myisam_max_sort_file_size	Yes	
myisam mmap_size	Yes	
myisam_sort_buffer_size	Yes	
myisam_stats_method	Yes	
myisam_use_mmap	Yes	
net_buffer_length	Yes	
net_read_timeout	Yes	
net_retry_count	Yes	
net_write_timeout	Yes	
old-style-user-limits	Yes	
old_passwords	Yes	Removed from Aurora MySQL version 3.
optimizer_prune_level	Yes	
optimizer_search_depth	Yes	
optimizer_switch	Yes	For information about Aurora MySQL features that use this switch, see Best practices with Amazon Aurora MySQL (p. 1033) .
optimizer_trace	Yes	
optimizer_trace_features	Yes	
optimizer_trace_limit	Yes	
optimizer_trace_max_mem_size	Yes	
optimizer_trace_offset	Yes	
performance-schema-consumer-events-waits-current	Yes	
performance-schema-instrument	Yes	
performance_schema	Yes	
performance_schema_accounts_size	Yes	
performance_schema_consumer_global_instrumentation	Yes	

Parameter name	Modifiable	Notes
performance_schema_consumer_thread_instrumentation	Yes	
performance_schema_consumer_events_stages_current	Yes	
performance_schema_consumer_events_stages_history	Yes	
performance_schema_consumer_events_stages_history_long	Yes	
performance_schema_consumer_events_statements_current	Yes	
performance_schema_consumer_events_statements_history	Yes	
performance_schema_consumer_events_statements_history_long	Yes	
performance_schema_consumer_events_waits_history	Yes	
performance_schema_consumer_events_waits_history_long	Yes	
performance_schema_consumer_statements_digest	Yes	
performance_schema_digests_size	Yes	
performance_schema_events_stages_history_long_size	Yes	
performance_schema_events_stages_history_size	Yes	
performance_schema_events_statements_history_long_size	Yes	
performance_schema_events_statements_history_size	Yes	
performance_schema_events_transactions_history	Yes	Aurora MySQL 2.x only
performance_schema_events_transactions_history	Yes	Aurora MySQL 2.x only
performance_schema_events_waits_history_long_size	Yes	
performance_schema_events_waits_history_size	Yes	
performance_schema_hosts_size	Yes	
performance_schema_max_cond_classes	Yes	
performance_schema_max_cond_instances	Yes	
performance_schema_max_digest_length	Yes	
performance_schema_max_file_classes	Yes	
performance_schema_max_file_handles	Yes	
performance_schema_max_file_instances	Yes	
performance_schema_max_index_stat	Yes	Aurora MySQL 2.x only
performance_schema_max_memory_classes	Yes	Aurora MySQL 2.x only
performance_schema_max_metadata_locks	Yes	Aurora MySQL 2.x only
performance_schema_max_mutex_classes	Yes	
performance_schema_max_mutex_instances	Yes	

Parameter name	Modifiable	Notes
performance_schema_max_prepared_statements_instances	Yes	Aurora MySQL 2.x only
performance_schema_max_program_instances	Yes	Aurora MySQL 2.x only
performance_schema_max_rwlock_classes	Yes	
performance_schema_max_rwlock_instances	Yes	
performance_schema_max_socket_classes	Yes	
performance_schema_max_socket_instances	Yes	
performance_schema_max_sql_text_length	Yes	Aurora MySQL 2.x only
performance_schema_max_stage_classes	Yes	
performance_schema_max_statement_classes	Yes	
performance_schema_max_statement_instances	Yes	Aurora MySQL 2.x only
performance_schema_max_table_handles	Yes	
performance_schema_max_table_instances	Yes	
performance_schema_max_table_lock_instances	Yes	Aurora MySQL 2.x only
performance_schema_max_thread_classes	Yes	
performance_schema_max_thread_instances	Yes	
performance_schema_session_connect_attrs_size	Yes	
performance_schema_setup_actors_size	Yes	
performance_schema_setup_objects_size	Yes	
performance_schema_users_size	Yes	
pid_file	No	
plugin_dir	No	Aurora MySQL uses managed instances where you don't access the file system directly.
port	No	Aurora MySQL manages the connection properties and enforces consistent settings for all DB instances in a cluster.
preload_buffer_size	Yes	
profiling_history_size	Yes	
query_alloc_block_size	Yes	
query_cache_limit	Yes	Removed from Aurora MySQL version 3.
query_cache_min_res_unit	Yes	Removed from Aurora MySQL version 3.

Parameter name	Modifiable	Notes
query_cache_size	Yes	The default value is represented by a formula. For details about how the DBInstanceClassMemory value in the formula is calculated, see DB parameter formula variables (p. 363) . Removed from Aurora MySQL version 3.
query_cache_type	Yes	Removed from Aurora MySQL version 3.
query_cache_wlock_invalidate	Yes	Removed from Aurora MySQL version 3.
query_prealloc_size	Yes	
range_alloc_block_size	Yes	
read_buffer_size	Yes	
read_only	Yes	Aurora MySQL manages the read-only and read/write state of DB instances based on the type of cluster. For example, a provisioned cluster has one read/write DB instance (the <i>primary instance</i>) and any other instances in the cluster are read-only (the Aurora Replicas). The writer instance can be switched to a read-only state by changing this parameter. Any reader instances are always in a read-only state, regardless of the value of this parameter.
read_rnd_buffer_size	Yes	
relay-log	No	
relay_log_info_repository	Yes	Removed from Aurora MySQL version 3.
relay_log_recovery	No	
safe-user-create	Yes	
secure_auth	Yes	Removed from Aurora MySQL version 3.
secure_file_priv	No	Aurora MySQL uses managed instances where you don't access the file system directly.
skip-slave-start	No	
skip_external_locking	No	
skip_show_database	Yes	
slave_checkpoint_group	Yes	Aurora MySQL version 1 and 2. Use <code>replica_checkpoint_group</code> in Aurora MySQL version 3.
replica_checkpoint_group	Yes	Aurora MySQL version 3 and higher

Parameter name	Modifiable	Notes
slave_checkpoint_period	Yes	Aurora MySQL version 1 and 2. Use <code>replica_checkpoint_period</code> in Aurora MySQL version 3.
replica_checkpoint_period	Yes	Aurora MySQL version 3 and higher
slave_parallel_workers	Yes	Aurora MySQL version 1 and 2. Use <code>replica_parallel_workers</code> in Aurora MySQL version 3.
replica_parallel_workers	Yes	Aurora MySQL version 3 and higher
slave_pending_jobs_size_max	Yes	Aurora MySQL version 1 and 2. Use <code>replica_pending_jobs_size_max</code> in Aurora MySQL version 3.
replica_pending_jobs_size_max	Yes	Aurora MySQL version 3 and higher
replica_skip_errors	Yes	Aurora MySQL version 3 and higher
slave_sql_verify_checksum	Yes	Aurora MySQL version 1 and 2. Use <code>replica_sql_verify_checksum</code> in Aurora MySQL version 3.
replica_sql_verify_checksum	Yes	Aurora MySQL version 3 and higher
slow_launch_time	Yes	
slow_query_log	Yes	For instructions on uploading the logs to CloudWatch Logs, see Publishing Amazon Aurora MySQL logs to Amazon CloudWatch Logs (p. 1017) .
slow_query_log_file	No	Aurora MySQL uses managed instances where you don't access the file system directly.
socket	No	
sort_buffer_size	Yes	
sql_mode	Yes	
sql_select_limit	Yes	
stored_program_cache	Yes	
sync_binlog	No	
sync_master_info	Yes	
sync_source_info	Yes	This parameter applies to Aurora MySQL 3 and higher.
sync_relay_log	Yes	Removed from Aurora MySQL version 3.
sync_relay_log_info	Yes	
sysdate-is-now	Yes	

Parameter name	Modifiable	Notes
table_cache_element_entry_ttl	No	
table_definition_cache	Yes	The default value is represented by a formula. For details about how the DBInstanceClassMemory value in the formula is calculated, see DB parameter formula variables (p. 363) .
table_open_cache	Yes	The default value is represented by a formula. For details about how the DBInstanceClassMemory value in the formula is calculated, see DB parameter formula variables (p. 363) .
table_open_cache_instances	Yes	
temp_pool	Yes	Removed from Aurora MySQL version 3.
temptable_max_mmap	Yes	This parameter applies to Aurora MySQL version 3 and higher. For details, see Storage engine for internal temporary tables (p. 754) .
temptable_max_ram	Yes	This parameter applies to Aurora MySQL version 3 and higher. For details, see Storage engine for internal temporary tables (p. 754) .
temptable_use_mmap	Yes	This parameter applies to Aurora MySQL version 3 and higher. For details, see Storage engine for internal temporary tables (p. 754) .
thread_handling	No	
thread_stack	Yes	
timed_mutexes	Yes	
tmp_table_size	Yes	
tmpdir	No	Aurora MySQL uses managed instances where you don't access the file system directly.
transaction_alloc_block_size	Yes	
transaction_isolation	Yes	This parameter applies to Aurora MySQL version 3 and higher. It replaces tx_isolation.
transaction_prealloc_size	Yes	
tx_isolation	Yes	Removed from Aurora MySQL version 3. It is replaced by transaction_isolation.
updatable_views_with_limit	Yes	

Parameter name	Modifiable	Notes
validate_password	No	
validate_password_dictionary_file	No	
validate_password_length	No	
validate_password_mixed_case_count	No	
validate_password_number_count	No	
validate_password_policy	No	
validate_password_special_char_count	No	
wait_timeout	Yes	Aurora evaluates the minimum value of <code>interactive_timeout</code> and <code>wait_timeout</code> . It then uses that minimum as the timeout to end all idle sessions, both interactive and noninteractive.

MySQL parameters that don't apply to Aurora MySQL

Because of architectural differences between Aurora MySQL and MySQL, some MySQL parameters don't apply to Aurora MySQL.

The following MySQL parameters don't apply to Aurora MySQL. This list is not exhaustive.

- `activate_all_roles_on_login`. This parameter isn't applicable to Aurora MySQL version 1 and 2. It is available in Aurora MySQL version 3.
- `big_tables`
- `bind_address`
- `character_sets_dir`
- `innodb_adaptive_flushing`
- `innodb_adaptive_flushing_lwm`
- `innodb_change_buffering`
- `innodb_checksum_algorithm`
- `innodb_data_file_path`
- `innodb_deadlock_detect`
- `innodb_dedicated_server`
- `innodb_doublewrite`
- `innodb_flush_method`
- `innodb_flush_neighbors`
- `innodb_io_capacity`
- `innodb_io_capacity_max`
- `innodb_buffer_pool_chunk_size`
- `innodb_buffer_pool_instances`
- `innodb_log_buffer_size`
- `innodb_default_row_format`

- `innodb_log_file_size`
- `innodb_log_files_in_group`
- `innodb_log_spin_cpu_abs_lwm`
- `innodb_log_spin_cpu_pct_hwm`
- `innodb_max_dirty_pages_pct`
- `innodb_numa_interleave`
- `innodb_page_size`
- `innodb_redo_log_encrypt`
- `innodb_undo_log_encrypt`
- `innodb_undo_log_truncate`
- `innodb_use_native_aio`
- `innodb_write_io_threads`
- `thread_cache_size`

MySQL status variables that don't apply to Aurora MySQL

Because of architectural differences between Aurora MySQL and MySQL, some MySQL status variables don't apply to Aurora MySQL.

The following MySQL status variables don't apply to Aurora MySQL. This list is not exhaustive.

- `innodb_buffer_pool_bytes_dirty`
- `innodb_buffer_pool_pages_dirty`
- `innodb_buffer_pool_pages_flushed`

Aurora MySQL version 3 removes the following status variables that were in Aurora MySQL version 2:

- `AuroraDb_lockmgr_bitmaps0_in_use`
- `AuroraDb_lockmgr_bitmaps1_in_use`
- `AuroraDb_lockmgr_bitmaps_mem_used`
- `AuroraDb_thread_deadlocks`
- `available_alter_table_log_entries`
- `Aurora_lockmgr_memory_used`
- `Aurora_missing_history_on_replica_incidents`
- `Aurora_new_lock_manager_lock_release_cnt`
- `Aurora_new_lock_manager_lock_release_total_duration_micro`
- `Aurora_new_lock_manager_lock_timeout_cnt`
- `Aurora_oom_response`
- `Aurora_total_op_memory`
- `Aurora_total_op_temp_space`
- `Aurora_used_alter_table_log_entries`
- `Aurora_using_new_lock_manager`
- `Aurora_volume_bytes_allocated`

- `Aurora_volume_bytes_left_extent`
- `Aurora_volume_bytes_left_total`
- `Com.Alter_db_upgrade`
- `Compression`
- `External_threads_connected`
- `Innodb_available_undo_logs`
- `Last_query_cost`
- `Last_query_partial_plans`
- `Slave_heartbeat_period`
- `Slave_last_heartbeat`
- `Slave_received_heartbeats`
- `Slave_retried_transactions`
- `Slave_running`
- `Time_since_zero_connections`

These MySQL status variables are available in Aurora MySQL version 1 or 2, but they aren't available in Aurora MySQL version 3:

- `Innodb_redo_log_enabled`
- `Innodb_undo tablespaces_total`
- `Innodb_undo tablespaces_implicit`
- `Innodb_undo tablespaces_explicit`
- `Innodb_undo tablespaces_active`

Aurora MySQL wait events

The following are some common wait events for Aurora MySQL.

Note

For information about the naming conventions used in MySQL wait events, see [Performance Schema instrument naming conventions](#) in the MySQL documentation.

`cpu`

The number of active connections that are ready to run is consistently higher than the number of vCPUs. For more information, see [cpu \(p. 841\)](#).

`io/aurora_redo_log_flush`

A session is persisting data to Aurora storage. Typically, this wait event is for a write I/O operation in Aurora MySQL. For more information, see [io/aurora_redo_log_flush \(p. 844\)](#).

`io/aurora_respond_to_client`

Query processing has completed and results are being returned to the application client for the following Aurora MySQL versions: 2.10.2 and higher 2.10 versions, 2.09.3 and higher 2.09 versions, 2.07.7 and higher 2.07 versions, and 1.22.6 and higher 1.22 versions. Compare the network bandwidth of the DB instance class with the size of the result set being returned. Also, check client-side response times. If the client is unresponsive and can't process the TCP packets, packet drops and TCP retransmissions can occur. This situation negatively affects network bandwidth. In versions lower than 2.10.2, 2.09.3, 2.07.7, and 1.22.6, the wait event erroneously includes idle time. To learn how to tune your database when this wait is prominent, see [io/aurora_respond_to_client \(p. 847\)](#).

io/file/csv/data

Threads are writing to tables in comma-separated value (CSV) format. Check your CSV table usage. A typical cause of this event is setting `log_output` on a table.

io/file/innodb/innodb_data_file

Threads are waiting on I/O from storage. This event is more prevalent in I/O-intensive workloads. When this wait event is prevalent, SQL statements might be running disk-intensive queries or requesting data that can't be satisfied from the InnoDB buffer pool. For more information, see [io/file/innodb/innodb_data_file \(p. 849\)](#).

io/file/sql/binlog

A thread is waiting on a binary log (binlog) file that is being written to disk.

io/socket/sql/client_connection

The `mysqld` program is busy creating threads to handle incoming new client connections. For more information, see [io/socket/sql/client_connection \(p. 851\)](#).

io/table/sql/handler

The engine is waiting for access to a table. This event occurs regardless of whether the data is cached in the buffer pool or accessed on disk. For more information, see [io/table/sql/handler \(p. 853\)](#).

lock/table/sql/handler

This wait event is a table lock wait event handler. For more information about atom and molecule events in the Performance Schema, see [Performance Schema atom and molecule events](#) in the MySQL documentation.

synch/cond/mysys/my_thread_var::suspend

The thread is suspended while waiting on a table-level lock because another thread issued `LOCK TABLES ... READ`.

synch/cond/sql/MDL_context::COND_wait_status

Threads are waiting on a table metadata lock. The engine uses this type of lock to manage concurrent access to a database schema and to ensure data consistency. For more information, see [Optimizing locking operations](#) in the MySQL documentation. To learn how to tune your database when this event is prominent, see [synch/cond/sql/MDL_context::COND_wait_status \(p. 857\)](#).

synch/cond/sql/MYSQL_BIN_LOG::COND_done

You have turned on binary logging. There might be a high commit throughput, large number transactions committing, or replicas reading binlogs. Consider using multirow statements or bundling statements into one transaction. In Aurora, use global databases instead of binary log replication, or use the `aurora_binlog_*` parameters.

synch/mutex/innodb/aurora_lock_thread_slot_futex

Multiple data manipulation language (DML) statements are accessing the same database rows at the same time. For more information, see [synch/mutex/innodb/aurora_lock_thread_slot_futex \(p. 864\)](#).

synch/mutex/innodb/buf_pool_mutex

The buffer pool isn't large enough to hold the working data set. Or the workload accesses pages from a specific table, which leads to contention in the buffer pool. For more information, see [synch/mutex/innodb/buf_pool_mutex \(p. 866\)](#).

synch/mutex/innodb/fil_system_mutex

The process is waiting for access to the tablespace memory cache. For more information, see [synch/mutex/innodb/fil_system_mutex \(p. 868\)](#).

synch/mutex/innodb/os_mutex

This event is part of an event semaphore. It provides exclusive access to variables used for signaling between threads. Uses include statistics threads, full-text search, buffer pool dump and load operations, and log flushes. This wait event is specific to Aurora MySQL version 1.

synch/mutex/innodb/trx_sys_mutex

Operations are checking, updating, deleting, or adding transaction IDs in InnoDB in a consistent or controlled manner. These operations require a `trx_sys` mutex call, which is tracked by Performance Schema instrumentation. Operations include management of the transaction system when the database starts or shuts down, rollbacks, undo cleanups, row read access, and buffer pool loads. High database load with a large number of transactions results in the frequent appearance of this wait event. For more information, see [synch/mutex/innodb/trx_sys_mutex \(p. 871\)](#).

synch/mutex/mysys/KEY_CACHE::cache_lock

The `keycache->cache_lock` mutex controls access to the key cache for MyISAM tables. In Aurora MySQL, this wait event is related to temporary table usage. Check the size of the `key_buffer_size`. Also check the values for `created_tmp_tables` or `created_tmp_disk_tables` at the time of the wait event spike. When it's justified, use multiple key caches.

synch/mutex/sql/FILE_AS_TABLE::LOCK_offsets

The engine acquires this mutex when opening or creating a table metadata file. When this wait event occurs with excessive frequency, the number of tables being created or opened has spiked.

synch/mutex/sql/FILE_AS_TABLE::LOCK_shim_lists

The engine acquires this mutex while performing operations such as `reset_size`, `detach_contents`, or `add_contents` on the internal structure that keeps track of opened tables. The mutex synchronizes access to the list contents. When this wait event occurs with high frequency, it indicates a sudden change in the set of tables that were previously accessed. The engine needs to access new tables or let go of the context related to previously accessed tables.

synch/mutex/sql/LOCK_open

The number of tables that your sessions are opening exceeds the size of the table definition cache or the table open cache. Increase the size of these caches.

synch/mutex/sql/LOCK_table_cache

The number of tables that your sessions are opening exceeds the size of the table definition cache or the table open cache. Increase the size of these caches.

synch/mutex/sql/LOG

In this wait event, there are threads waiting on a log lock. For example, a thread might wait for a lock to write to the slow query log file.

synch/mutex/sql/MYSQL_BIN_LOG::LOCK_commit

In this wait event, there is a thread that is waiting to acquire a lock with the intention of committing to the binary log. Binary logging contention can occur on databases with a very high change rate. Depending on your version of MySQL, there are certain locks being used to protect the consistency and durability of the binary log. In RDS for MySQL, binary logs are used for replication and the automated backup process. In Aurora MySQL, binary logs are not needed for native replication or backups. They are disabled by default but can be enabled and used for external replication or change data capture. For more information, see [The binary log](#) in the MySQL documentation.

sync/mutex/sql/MYSQL_BIN_LOG::LOCK_dump_thread_metrics_collection

If binary logging is turned on, the engine acquires this mutex when it prints active dump threads metrics to the engine error log and to the internal operations map.

sync/mutex/sql/MYSQL_BIN_LOG::LOCK_inactive_binlogs_map

If binary logging is turned on, the engine acquires this mutex when it adds to, deletes from, or searches through the list of binlog files behind the latest one.

sync/mutex/sql/MYSQL_BIN_LOG::LOCK_io_cache

If binary logging is turned on, the engine acquires this mutex during Aurora binlog IO cache operations: allocate, resize, free, write, read, purge, and access cache info. If this event occurs frequently, the engine is accessing the cache where binlog events are stored. To reduce wait times, reduce commits. Try grouping multiple statements into a single transaction.

synch/mutex/sql/MYSQL_BIN_LOG::LOCK_log

You have turned on binary logging. There might be high commit throughput, many transactions committing, or replicas reading binlogs. Consider using multirow statements or bundling statements into one transaction. In Aurora, use global databases instead of binary log replication or use the `aurora_binlog_*` parameters.

synch/mutex/sql/SERVER_THREAD::LOCK_sync

The mutex `SERVER_THREAD::LOCK_sync` is acquired during the scheduling, processing, or launching of threads for file writes. The excessive occurrence of this wait event indicates increased write activity in the database.

synch/rwlock/sql/TABLESPACES:lock

The engine acquires the `TABLESPACES:lock` mutex during the following tablespace operations: create, delete, truncate, and extend. The excessive occurrence of this wait event indicates a high frequency of tablespace operations. An example is loading a large amount of data into the database.

synch/rwlock/innodb/dict

In this wait event, there are threads waiting on an rwlock held on the InnoDB data dictionary.

synch/rwlock/innodb/dict_operation_lock

In this wait event, there are threads holding locks on InnoDB data dictionary operations.

synch/rwlock/innodb/dict sys RW lock

A high number of concurrent data control language statements (DCLs) in data definition language code (DDFs) are triggered at the same time. Reduce the application's dependency on DDFs during regular application activity.

synch/rwlock/innodb/hash_table_locks

The excessive occurrence of this wait event indicates contention when modifying the hash table that maps the buffer cache. Consider increasing the buffer cache size and improving access paths for the relevant queries. To learn how to tune your database when this wait is prominent, see [synch/rwlock/innodb/hash_table_locks \(p. 872\)](#).

synch/rwlock/innodb/index_tree_rw_lock

A large number of similar data manipulation language (DML) statements are accessing the same database object at the same time. Try using multirow statements. Also, spread the workload over different database objects. For example, implement partitioning.

synch/sxlock/innodb/dict_operation_lock

A high number of concurrent data control language statements (DCLs) in data definition language code (DDFs) are triggered at the same time. Reduce the application's dependency on DDFs during regular application activity.

synch/sxlock/innodb/dict_sys_lock

A high number of concurrent data control language statements (DCLs) in data definition language code (DDFs) are triggered at the same time. Reduce the application's dependency on DDFs during regular application activity.

synch/sxlock/innodb/hash_table_locks

The session couldn't find pages in the buffer pool. The engine either needs to read a file or modify the least-recently used (LRU) list for the buffer pool. Consider increasing the buffer cache size and improving access paths for the relevant queries.

synch/sxlock/innodb/index_tree_rw_lock

Many similar data manipulation language (DML) statements are accessing the same database object at the same time. Try using multirow statements. Also, spread the workload over different database objects. For example, implement partitioning.

Aurora MySQL thread states

The following are some common thread states for Aurora MySQL.

checking permissions

The thread is checking whether the server has the required privileges to run the statement.

checking query cache for query

The server is checking whether the current query is present in the query cache.

cleaned up

This is the final state of a connection whose work is complete but which hasn't been closed by the client. The best solution is to explicitly close the connection in code. Or you can set a lower value for `wait_timeout` in your parameter group.

closing tables

The thread is flushing the changed table data to disk and closing the used tables. If this isn't a fast operation, verify the network bandwidth consumption metrics against the instance class network bandwidth. Also, check that the parameter values for `table_open_cache` and `table_definition_cache` parameter allow for enough tables to be simultaneously open so that the engine doesn't need to open and close tables frequently. These parameters influence the memory consumption on the instance.

converting HEAP to MyISAM

The query is converting a temporary table from in-memory to on-disk. This conversion is necessary because the temporary tables created by MySQL in the intermediate steps of query processing grew too big for memory. Check the values of `tmp_table_size` and `max_heap_table_size`. In later versions, this thread state name is `converting HEAP to ondisk`.

converting HEAP to ondisk

The thread is converting an internal temporary table from an in-memory table to an on-disk table.

copy to tmp table

The thread is processing an `ALTER TABLE` statement. This state occurs after the table with the new structure has been created but before rows are copied into it. For a thread in this state, you can use the Performance Schema to obtain information about the progress of the copy operation.

creating sort index

Aurora MySQL is performing a sort because it can't use an existing index to satisfy the `ORDER BY` or `GROUP BY` clause of a query. For more information, see [creating sort index \(p. 876\)](#).

creating table

The thread is creating a permanent or temporary table.

delayed commit ok done

An asynchronous commit in Aurora MySQL has received an acknowledgement and is complete.

delayed commit ok initiated

The Aurora MySQL thread has started the async commit process but is waiting for acknowledgement. This is usually the genuine commit time of a transaction.

delayed send ok done

An Aurora MySQL worker thread that is tied to a connection can be freed while a response is sent to the client. The thread can begin other work. The state `delayed_send ok` means that the asynchronous acknowledgement to the client completed.

delayed send ok initiated

An Aurora MySQL worker thread has sent a response asynchronously to a client and is now free to do work for other connections. The transaction has started an async commit process that hasn't yet been acknowledged.

executing

The thread has begun running a statement.

freeing items

The thread has run a command. Some freeing of items done during this state involves the query cache. This state is usually followed by cleaning up.

init

This state occurs before the initialization of `ALTER TABLE`, `DELETE`, `INSERT`, `SELECT`, or `UPDATE` statements. Actions in this state include flushing the binary log or InnoDB log, and some cleanup of the query cache.

master has sent all binlog to slave

The primary node has finished its part of the replication. The thread is waiting for more queries to run so that it can write to the binary log (binlog).

opening tables

The thread is trying to open a table. This operation is fast unless an `ALTER TABLE` or a `LOCK TABLE` statement needs to finish, or it exceeds the value of `table_open_cache`.

optimizing

The server is performing initial optimizations for a query.

preparing

This state occurs during query optimization.

query end

This state occurs after processing a query but before the freeing items state.

removing duplicates

Aurora MySQL couldn't optimize a `DISTINCT` operation in the early stage of a query. Aurora MySQL must remove all duplicated rows before sending the result to the client.

searching rows for update

The thread is finding all matching rows before updating them. This stage is necessary if the `UPDATE` is changing the index that the engine uses to find the rows.

sending binlog event to slave

The thread read an event from the binary log and is sending it to the replica.

sending cached result to client

The server is taking the result of a query from the query cache and sending it to the client.

sending data

The thread is reading and processing rows for a `SELECT` statement but hasn't yet started sending data to the client. The process is identifying which pages contain the results necessary to satisfy the query. For more information, see [sending data \(p. 879\)](#).

sending to client

The server is writing a packet to the client. In earlier MySQL versions, this wait event was labeled `writing to net`.

starting

This is the first stage at the beginning of statement execution.

statistics

The server is calculating statistics to develop a query execution plan. If a thread is in this state for a long time, the server is probably disk-bound while performing other work.

storing result in query cache

The server is storing the result of a query in the query cache.

system lock

The thread has called `mysql_lock_tables`, but the thread state hasn't been updated since the call. This general state occurs for many reasons.

update

The thread is preparing to start updating the table.

updating

The thread is searching for rows and is updating them.

user lock

The thread issued a `GET_LOCK` call. The thread either requested an advisory lock and is waiting for it, or is planning to request it.

waiting for more updates

The primary node has finished its part of the replication. The thread is waiting for more queries to run so that it can write to the binary log (binlog).

waiting for schema metadata lock

This is a wait for a metadata lock.

waiting for stored function metadata lock

This is a wait for a metadata lock.

waiting for stored procedure metadata lock

This is a wait for a metadata lock.

waiting for table flush

The thread is executing `FLUSH TABLES` and is waiting for all threads to close their tables. Or the thread received notification that the underlying structure for a table changed, so it must reopen the table to get the new structure. To reopen the table, the thread must wait until all other threads have closed the table. This notification takes place if another thread has used one of the following

statements on the table: FLUSH TABLES, ALTER TABLE, RENAME TABLE, REPAIR TABLE, ANALYZE TABLE, or OPTIMIZE TABLE.

waiting for table level lock

One session is holding a lock on a table while another session tries to acquire the same lock on the same table.

waiting for table metadata lock

Aurora MySQL uses metadata locking to manage concurrent access to database objects and to ensure data consistency. In this wait event, one session is holding a metadata lock on a table while another session tries to acquire the same lock on the same table. When the Performance Schema is enabled, this thread state is reported as the wait event synch/cond/sql/MDL_context::COND_wait_status.

writing to net

The server is writing a packet to the network. In later MySQL versions, this wait event is labeled Sending to client.

Aurora MySQL isolation levels

Following, you can learn how DB instances in an Aurora MySQL cluster implement the database property of isolation. Doing so helps you understand how the Aurora MySQL default behavior balances between strict consistency and high performance. You can also decide when to change the default settings based on the characteristics of your workload.

Available isolation levels for writer instances

You can use the isolation levels REPEATABLE READ, READ COMMITTED, READ UNCOMMITTED, and SERIALIZABLE on the primary instance of an Aurora MySQL single-master cluster. You can use the isolation levels REPEATABLE READ, READ COMMITTED, and READ UNCOMMITTED on any DB instance in an Aurora MySQL multi-master cluster. These isolation levels work the same in Aurora MySQL as in RDS for MySQL.

REPEATABLE READ isolation level for reader instances

By default, Aurora MySQL DB instances configured as read-only Aurora Replicas always use the REPEATABLE READ isolation level. These DB instances ignore any SET TRANSACTION ISOLATION LEVEL statements and continue using the REPEATABLE READ isolation level.

READ COMMITTED isolation level for reader instances

If your application includes a write-intensive workload on the primary instance and long-running queries on the Aurora Replicas, you might experience substantial purge lag. *Purge lag* happens when internal garbage collection is blocked by long-running queries. The symptom that you see is a high value for history_list_length in output from the SHOW ENGINE INNODB STATUS command. You can monitor this value using the RollbackSegmentHistoryListLength metric in CloudWatch. This condition can reduce the effectiveness of secondary indexes and lead to reduced overall query performance and wasted storage space.

If you experience such issues, you can use an Aurora MySQL session-level configuration setting, aurora_read_replica_read_committed, to use the READ COMMITTED isolation level on Aurora Replicas. Using this setting can help reduce slowdowns and wasted space that can result from performing long-running queries at the same time as transactions that modify your tables.

We recommend making sure that you understand the specific Aurora MySQL behavior of the READ COMMITTED isolation before using this setting. The Aurora Replica READ COMMITTED behavior

complies with the ANSI SQL standard. However, the isolation is less strict than typical MySQL `READ COMMITTED` behavior that you might be familiar with. Thus, you might see different query results under `READ COMMITTED` on an Aurora MySQL read replica than for the same query under `READ COMMITTED` on the Aurora MySQL primary instance or on RDS for MySQL. You might use the `aurora_read_replica_read_committed` setting for such use cases as a comprehensive report that scans a very large database. You might avoid it for short queries with small result sets, where precision and repeatability are important.

The `READ COMMITTED` isolation level isn't available for sessions within a secondary cluster in an Aurora global database that use the write forwarding feature. For information about write forwarding, see [Using write forwarding in an Amazon Aurora global database \(p. 255\)](#).

Enabling `READ COMMITTED` for readers

To enable the `READ COMMITTED` isolation level for Aurora Replicas, enable the `aurora_read_replica_read_committed` configuration setting. Enable this setting at the session level while connected to a specific Aurora Replica. To do so, run the following SQL commands.

```
set session aurora_read_replica_read_committed = ON;
set session transaction isolation level read committed;
```

You might enable this configuration setting temporarily to perform interactive ad hoc (one-time) queries. You might also want to run a reporting or data analysis application that benefits from the `READ COMMITTED` isolation level, while leaving the default unchanged for other applications.

When the `aurora_read_replica_read_committed` setting is enabled, use the `SET TRANSACTION ISOLATION LEVEL` command to specify the isolation level for the appropriate transactions.

```
set transaction isolation level read committed;
```

Differences in `READ COMMITTED` behavior on Aurora replicas

The `aurora_read_replica_read_committed` setting makes the `READ COMMITTED` isolation level available for an Aurora Replica, with consistency behavior that is optimized for long-running transactions. The `READ COMMITTED` isolation level on Aurora Replicas has less strict isolation than on Aurora primary instances or multi-master instances. For that reason, enable this setting only on Aurora Replicas where you know that your queries can accept the possibility of certain types of inconsistent results.

Your queries can experience certain kinds of read anomalies when the `aurora_read_replica_read_committed` setting is turned on. Two kinds of anomalies are especially important to understand and handle in your application code. A *non-repeatable read* occurs when another transaction commits while your query is running. A long-running query can see different data at the start of the query than it sees at the end. A *phantom read* occurs when other transactions cause existing rows to be reorganized while your query is running, and one or more rows are read twice by your query.

Your queries might experience inconsistent row counts as a result of phantom reads. Your queries might also return incomplete or inconsistent results due to non-repeatable reads. For example, suppose that a join operation refers to tables that are concurrently modified by SQL statements such as `INSERT` or `DELETE`. In this case, the join query might read a row from one table but not the corresponding row from another table.

The ANSI SQL standard allows both of these behaviors for the `READ COMMITTED` isolation level. However, those behaviors are different than the typical MySQL implementation of `READ COMMITTED`. Thus, before enabling the `aurora_read_replica_read_committed` setting, check any existing SQL code to verify if it operates as expected under the looser consistency model.

Row counts and other results might not be strongly consistent under the `READ COMMITTED` isolation level while this setting is enabled. Thus, you typically enable the setting only while running analytic queries that aggregate large amounts of data and don't require absolute precision. If you don't have these kinds of long-running queries alongside a write-intensive workload, you probably don't need the `aurora_read_replica_read_committed` setting. Without the combination of long-running queries and a write-intensive workload, you're unlikely to encounter issues with the length of the history list.

Example Queries showing isolation behavior for READ COMMITTED on Aurora replicas

The following example shows how `READ COMMITTED` queries on an Aurora Replica might return non-repeatable results if transactions modify the associated tables at the same time. The table `BIG_TABLE` contains 1 million rows before any queries start. Other data manipulation language (DML) statements add, remove, or change rows while the are running.

The queries on the Aurora primary instance under the `READ COMMITTED` isolation level produce predictable results. However, the overhead of keeping the consistent read view for the lifetime of every long-running query can lead to expensive garbage collection later.

The queries on the Aurora Replica under the `READ COMMITTED` isolation level are optimized to minimize this garbage collection overhead. The tradeoff is that the results might vary depending on whether the queries retrieve rows that are added, removed, or reorganized by transactions that commit while the query is running. The queries are allowed to consider these rows but aren't required to. For demonstration purposes, the queries check only the number of rows in the table by using the `COUNT(*)` function.

Time	DML statement on Aurora primary instance	Query on Aurora primary instance with <code>READ COMMITTED</code>	Query on Aurora replica with <code>READ COMMITTED</code>
T1	<code>INSERT INTO big_table SELECT * FROM other_table LIMIT 1000000; COMMIT;</code>		
T2		Q1: <code>SELECT COUNT(*) FROM big_table;</code>	Q2: <code>SELECT COUNT(*) FROM big_table;</code>
T3	<code>INSERT INTO big_table (c1, c2) VALUES (1, 'one more row'); COMMIT;</code>		
T4		If Q1 finishes now, result is 1,000,000.	If Q2 finishes now, result is 1,000,000 or 1,000,001.
T5	<code>DELETE FROM big_table LIMIT 2; COMMIT;</code>		
T6		If Q1 finishes now, result is 1,000,000.	If Q2 finishes now, result is 1,000,000 or 1,000,001 or 999,999 or 999,998.
T7	<code>UPDATE big_table SET c2 =</code>		

Time	DML statement on Aurora primary instance	Query on Aurora primary instance with READ COMMITTED	Query on Aurora replica with READ COMMITTED
	CONCAT(c2,c2,c2); COMMIT;		
T8		If Q1 finishes now, result is 1,000,000.	If Q2 finishes now, result is 1,000,000 or 1,000,001 or 999,999, or possibly some higher number.
T9		Q3: SELECT COUNT(*) FROM big_table;	Q4: SELECT COUNT(*) FROM big_table;
T10		If Q3 finishes now, result is 999,999.	If Q4 finishes now, result is 999,999.
T11		Q5: SELECT COUNT(*) FROM parent_table p JOIN child_table c ON (p.id = c.id) WHERE p.id = 1000;	Q6: SELECT COUNT(*) FROM parent_table p JOIN child_table c ON (p.id = c.id) WHERE p.id = 1000;
T12	INSERT INTO parent_table (id, s) VALUES (1000, 'hello'); INSERT INTO child_table (id, s) VALUES (1000, 'world'); COMMIT;		
T13		If Q5 finishes now, result is 0.	If Q6 finishes now, result is 0 or 1.

If the queries finish quickly, before any other transactions perform DML statements and commit, the results are predictable and the same between the primary instance and the Aurora Replica.

The results for Q1 are highly predictable, because `READ COMMITTED` on the primary instance uses a strong consistency model similar to the `REPEATABLE READ` isolation level.

The results for Q2 might vary depending on what transactions commit while that query is running. For example, suppose that other transactions perform DML statements and commit while the queries are running. In this case, the query on the Aurora Replica with the `READ COMMITTED` isolation level might or might not take the changes into account. The row counts are not predictable in the same way as under the `REPEATABLE READ` isolation level. They also aren't as predictable as queries running under the `READ COMMITTED` isolation level on the primary instance, or on an RDS for MySQL instance.

The `UPDATE` statement at T7 doesn't actually change the number of rows in the table. However, by changing the length of a variable-length column, this statement can cause rows to be reorganized internally. A long-running `READ COMMITTED` transaction might see the old version of a row, and later within the same query see the new version of the same row. The query can also skip both the old and new versions of the row. Thus, the row count might be different than expected.

The results of Q5 and Q6 might be identical or slightly different. Query Q6 on the Aurora Replica under `READ COMMITTED` is able to see, but is not required to see, the new rows that are committed while the query is running. It might also see the row from one table but not from the other table. If the join query

doesn't find a matching row in both tables, it returns a count of zero. If the query does find both the new rows in `PARENT_TABLE` and `CHILD_TABLE`, the query returns a count of one. In a long-running query, the lookups from the joined tables might happen at widely separated times.

Note

These differences in behavior depend on the timing of when transactions are committed and when the queries process the underlying table rows. Thus, you're most likely to see such differences in report queries that take minutes or hours and that run on Aurora clusters processing OLTP transactions at the same time. These are the kinds of mixed workloads that benefit the most from the `READ COMMITTED` isolation level on Aurora Replicas.

Aurora MySQL hints

You can use SQL hints with Aurora MySQL queries to fine-tune performance. You can also use hints to prevent execution plans for important queries to change based on unpredictable conditions.

Tip

To verify the effect that a hint has on a query, examine the query plan produced by the `EXPLAIN` statement. Compare the query plans with and without the hint.

In Aurora MySQL version 3, you can use all the hints that are available in community MySQL 8.0. For details about these hints, see [Optimizer Hints](#) in the *MySQL Reference Manual*.

The following hints are available in Aurora MySQL 2.08 and higher. These hints apply to queries that use the hash join feature in Aurora MySQL version 2, especially queries that use the parallel query optimization.

HASH_JOIN, NO_HASH_JOIN

Turns on or off the ability of the optimizer to choose whether to use the hash join optimization method for a query. `HASH_JOIN` enables the optimizer to use hash join if that mechanism is more efficient. `NO_HASH_JOIN` prevents the optimizer from using hash join for the query. This hint is available in Aurora MySQL 2.08 and higher minor versions. It has no effect in Aurora MySQL version 3.

The following examples show how to use this hint.

```
EXPLAIN SELECT /*+ HASH_JOIN(t2) */ f1, f2
    FROM t1, t2 WHERE t1.f1 = t2.f1;

EXPLAIN SELECT /*+ NO_HASH_JOIN(t2) */ f1, f2
    FROM t1, t2 WHERE t1.f1 = t2.f1;
```

HASH_JOIN_PROBING, NO_HASH_JOIN_PROBING

In a hash join query, specifies whether or not to use the specified table for the probe side of the join. The query tests whether column values from the build table exist in the probe table, instead of reading the entire contents of the probe table. You can use `HASH_JOIN_PROBING` and `HASH_JOIN_BUILDING` to specify how hash join queries are processed without reordering the tables within the query text. This hint is available in Aurora MySQL 2.08 and higher minor versions. It has no effect in Aurora MySQL version 3.

The following examples show how to use this hint. Specifying the `HASH_JOIN_PROBING` hint for the table `T2` has the same effect as specifying `NO_HASH_JOIN_PROBING` for the table `T1`.

```
EXPLAIN SELECT /*+ HASH_JOIN(t2) HASH_JOIN_PROBING(t2) */ f1, f2
    FROM t1, t2 WHERE t1.f1 = t2.f1;

EXPLAIN SELECT /*+ HASH_JOIN(t2) NO_HASH_JOIN_PROBING(t1) */ f1, f2
```

```
FROM t1, t2 WHERE t1.f1 = t2.f1;
```

HASH_JOIN_BUILDING, NO_HASH_JOIN_BUILDING

In a hash join query, specifies whether or not to use the specified table for the build side of the join. The query processes all the rows from this table to build the list of column values to cross-reference with the other table. You can use `HASH_JOIN_PROBING` and `HASH_JOIN_BUILDING` to specify how hash join queries are processed without reordering the tables within the query text. This hint is available in Aurora MySQL 2.08 and higher minor versions. It has no effect in Aurora MySQL version 3.

The following examples show how to use this hint. Specifying the `HASH_JOIN_BUILDING` hint for the table `T2` has the same effect as specifying `NO_HASH_JOIN_BUILDING` for the table `T1`.

```
EXPLAIN SELECT /*+ HASH_JOIN(t2) HASH_JOIN_BUILDING(t2) */ f1, f2
    FROM t1, t2 WHERE t1.f1 = t2.f1;

EXPLAIN SELECT /*+ HASH_JOIN(t2) NO_HASH_JOIN_BUILDING(t1) */ f1, f2
    FROM t1, t2 WHERE t1.f1 = t2.f1;
```

JOIN_FIXED_ORDER

Specifies that tables in the query are joined based on the order they are listed in the query. It is especially useful with queries involving three or more tables. It is intended as a replacement for the MySQL `STRAIGHT_JOIN` hint. Equivalent to the MySQL `JOIN_FIXED_ORDER` hint. This hint is available in Aurora MySQL 2.08 and higher.

The following examples show how to use this hint.

```
EXPLAIN SELECT /*+ JOIN_FIXED_ORDER */ f1, f2
    FROM t1 JOIN t2 USING (id) JOIN t3 USING (id) JOIN t4 USING (id);
```

JOIN_ORDER

Specifies the join order for the tables in the query. It is especially useful with queries involving three or more tables. Equivalent to the MySQL `JOIN_ORDER` hint. This hint is available in Aurora MySQL 2.08 and higher.

The following examples show how to use this hint.

```
EXPLAIN SELECT /*+ JOIN_ORDER (t4, t2, t1, t3) */ f1, f2
    FROM t1 JOIN t2 USING (id) JOIN t3 USING (id) JOIN t4 USING (id);
```

JOIN_PREFIX

Specifies the tables to put first in the join order. It is especially useful with queries involving three or more tables. Equivalent to the MySQL `JOIN_PREFIX` hint. This hint is available in Aurora MySQL 2.08 and higher.

The following examples show how to use this hint.

```
EXPLAIN SELECT /*+ JOIN_ORDER (t4, t2) */ f1, f2
    FROM t1 JOIN t2 USING (id) JOIN t3 USING (id) JOIN t4 USING (id);
```

JOIN_SUFFIX

Specifies the tables to put last in the join order. It is especially useful with queries involving three or more tables. Equivalent to the MySQL `JOIN_SUFFIX` hint. This hint is available in Aurora MySQL 2.08 and higher.

The following examples show how to use this hint.

```
EXPLAIN SELECT /*+ JOIN_ORDER (t1, t3) */ f1, f2
    FROM t1 JOIN t2 USING (id) JOIN t3 USING (id) JOIN t4 USING (id);
```

For information about using hash join queries, see [Optimizing large Aurora MySQL join queries with hash joins \(p. 1038\)](#).

Aurora MySQL stored procedures

You can call the following stored procedures while connected to the primary instance in an Aurora MySQL cluster. These procedures control how transactions are replicated from an external database into Aurora MySQL, or from Aurora MySQL to an external database. To learn how to use replication based on global transaction identifiers (GTIDs) with Aurora MySQL, see [Using GTID-based replication for Aurora MySQL \(p. 954\)](#).

Topics

- [mysql.rds_assign_gtids_to_anonymous_transactions \(Aurora MySQL version 3 and higher\) \(p. 1076\)](#)
- [mysql.rds_set_master_auto_position \(Aurora MySQL version 1 and 2\) \(p. 1077\)](#)
- [mysql.rds_set_source_auto_position \(Aurora MySQL version 3 and higher\) \(p. 1077\)](#)
- [mysql.rds_set_source_auto_position \(Aurora MySQL version 3 and higher\) \(p. 1077\)](#)
- [mysql.rds_set_external_master_with_auto_position \(Aurora MySQL version 1 and 2\) \(p. 1078\)](#)
- [mysql.rds_set_external_source_with_auto_position \(Aurora MySQL version 3 and higher\) \(p. 1080\)](#)
- [mysql.rds_skip_transaction_with_gtid \(p. 1082\)](#)

[mysql.rds_assign_gtids_to_anonymous_transactions \(Aurora MySQL version 3 and higher\)](#)

Syntax

```
CALL mysql.rds_assign_gtids_to_anonymous_transactions(gtid_option);
```

Parameters

gtid_option

String value. The allowed values are OFF, LOCAL, or a specified UUID.

Usage notes

This procedure has the same effect as issuing the statement `CHANGE REPLICATION SOURCE TO ASSIGN_GTIDS_TO_ANONYMOUS_TRANSACTIONS = gtid_option` in community MySQL.

GTID must be turned to ON for *gtid_option* to be set to LOCAL or a specific UUID.

The default is OFF, meaning that the feature is not used.

LOCAL assigns a GTID including the replica's own UUID (the `server_uuid` setting).

Passing a parameter that is a UUID assigns a GTID that includes the specified UUID, such as the `server_uuid` setting for the replication source server.

Examples

To turn off this feature:

```
mysql> call mysql.rds_assign_gtids_to_anonymous_transactions('OFF');
+-----+
| Message |
+-----+
| ASSIGN_GTIDS_TO_ANONYMOUS_TRANSACTIONS has been set to: OFF |
+-----+
1 row in set (0.07 sec)
```

To use the replica's own UUID:

```
mysql> call mysql.rds_assign_gtids_to_anonymous_transactions('LOCAL');
+-----+
| Message |
+-----+
| ASSIGN_GTIDS_TO_ANONYMOUS_TRANSACTIONS has been set to: LOCAL |
+-----+
1 row in set (0.07 sec)
```

To use a specified UUID:

```
mysql> call mysql.rds_assign_gtids_to_anonymous_transactions('317a4760-
f3dd-3b74-8e45-0615ed29de0e');
+-----+
| Message |
+-----+
| ASSIGN_GTIDS_TO_ANONYMOUS_TRANSACTIONS has been set to: 317a4760-
f3dd-3b74-8e45-0615ed29de0e |
+-----+
1 row in set (0.07 sec)
```

[mysql.rds_set_master_auto_position \(Aurora MySQL version 1 and 2\)](#)

Sets the replication mode to be based on either binary log file positions or on global transaction identifiers (GTIDs).

Syntax

```
CALL mysql.rds_set_master_auto_position (auto_position_mode);
```

[mysql.rds_set_source_auto_position \(Aurora MySQL version 3 and higher\)](#)

Sets the replication mode to be based on either binary log file positions or on global transaction identifiers (GTIDs).

Syntax

```
CALL mysql.rds_set_source_auto_position (auto_position_mode);
```

Parameters

auto_position_mode

A value that indicates whether to use log file position replication or GTID-based replication:

- 0 – Use the replication method based on binary log file position. The default is 0.
- 1 – Use the GTID-based replication method.

Usage notes

For an Aurora MySQL DB cluster, you call this stored procedure while connected to the primary instance.

The master user must run the `mysql.rds_set_master_auto_position` procedure.

For Aurora, this procedure is supported for Aurora MySQL version 2.04 and later MySQL 5.7–compatible versions. GTID-based replication isn't supported for Aurora MySQL 1.1 or 1.0.

[mysql.rds_set_source_auto_position \(Aurora MySQL version 3 and higher\)](#)

Sets the replication mode to be based on either binary log file positions or on global transaction identifiers (GTIDs).

Syntax

```
CALL mysql.rds_set_source_auto_position (auto_position_mode);
```

Parameters

auto_position_mode

A value that indicates whether to use log file position replication or GTID-based replication:

- 0 – Use the replication method based on binary log file position. The default is 0.
- 1 – Use the GTID-based replication method.

Usage notes

For an Aurora MySQL DB cluster, you call this stored procedure while connected to the primary instance.

The administrative user must run the `mysql.rds_set_source_auto_position` procedure.

For Aurora, this procedure is supported for Aurora MySQL version 2.04 and later MySQL 5.7–compatible versions. GTID-based replication isn't supported for Aurora MySQL 1.1 or 1.0.

[mysql.rds_set_external_master_with_auto_position \(Aurora MySQL version 1 and 2\)](#)

Configures an Aurora MySQL primary instance to accept incoming replication from an external MySQL instance. This procedure also configures replication based on global transaction identifiers (GTIDs).

This procedure is available for both RDS for MySQL and Aurora MySQL. It works differently depending on the context. When used with Aurora MySQL, this procedure doesn't configure delayed replication. This limitation is because RDS for MySQL supports delayed replication but Aurora MySQL doesn't.

Syntax

```
CALL mysql.rds_set_external_master_with_auto_position (
    host_name
    , host_port
    , replication_user_name
    , replication_user_password
    , ssl_encryption
);
```

Parameters

host_name

The host name or IP address of the MySQL instance running external to Aurora to become the replication master.

host_port

The port used by the MySQL instance running external to Aurora to be configured as the replication master. If your network configuration includes Secure Shell (SSH) port replication that converts the port number, specify the port number that is exposed by SSH.

replication_user_name

The ID of a user with `REPLICATION CLIENT` and `REPLICATION SLAVE` permissions on the MySQL instance running external to Aurora. We recommend that you provide an account that is used solely for replication with the external instance.

replication_user_password

The password of the user ID specified in `replication_user_name`.

ssl_encryption

This option is not currently implemented. The default is 0.

Usage notes

For an Aurora MySQL DB cluster, you call this stored procedure while connected to the primary instance.

The master user must run the `mysql.rds_set_external_master_with_auto_position` procedure. The master user runs this procedure on the primary instance of an Aurora MySQL DB cluster that acts as a replication target. This can be the replication target of an external MySQL DB instance or an Aurora MySQL DB cluster.

For Aurora, this procedure is supported for Aurora MySQL version 2.04 and later MySQL 5.7-compatible versions. GTID-based replication isn't supported for Aurora MySQL 1.1 or 1.0. For Aurora MySQL version 3, use the procedure `mysql.rds_set_external_source_with_auto_position` instead.

Before you run `mysql.rds_set_external_master_with_auto_position`, configure the external MySQL DB instance to be a replication master. To connect to the external MySQL instance, specify values for `replication_user_name` and `replication_user_password`. These values must indicate a replication user that has `REPLICATION CLIENT` and `REPLICATION SLAVE` permissions on the external MySQL instance.

To configure an external MySQL instance as a replication master

1. Using the MySQL client of your choice, connect to the external MySQL instance and create a user account to be used for replication. The following is an example.

```
CREATE USER 'repl_user'@'mydomain.com' IDENTIFIED BY 'SomePassW0rd'
```

2. On the external MySQL instance, grant REPLICATION CLIENT and REPLICATION SLAVE privileges to your replication user. The following example grants REPLICATION CLIENT and REPLICATION SLAVE privileges on all databases for the 'repl_user' user for your domain.

```
GRANT REPLICATION CLIENT, REPLICATION SLAVE ON *.* TO 'repl_user'@'mydomain.com'  
IDENTIFIED BY 'SomePassW0rd'
```

When you call `mysql.rds_set_external_master_with_auto_position`, Amazon RDS records certain information. This information is the time, the user, and an action of "set master" in the `mysql.rds_history` and `mysql.rds_replication_status` tables.

To skip a specific GTID-based transaction that is known to cause a problem, you can use the [mysql.rds_skip_transaction_with_gtid \(p. 1082\)](#) stored procedure. For more information about working with GTID-based replication, see [Using GTID-based replication for Aurora MySQL \(p. 954\)](#).

Examples

When run on an Aurora primary instance, the following example configures the Aurora cluster to act as a read replica of an instance of MySQL running external to Aurora.

```
call mysql.rds_set_external_master_with_auto_position(  
    'Externaldb.some.com',  
    3306,  
    'repl_user'@'mydomain.com',  
    'SomePassW0rd');
```

mysql.rds_set_external_source_with_auto_position (Aurora MySQL version 3 and higher)

Configures an Aurora MySQL primary instance to accept incoming replication from an external MySQL instance. This procedure also configures replication based on global transaction identifiers (GTIDs).

This procedure is available for both RDS for MySQL and Aurora MySQL. It works differently depending on the context. When used with Aurora MySQL, this procedure doesn't configure delayed replication. This limitation is because RDS for MySQL supports delayed replication but Aurora MySQL doesn't.

Syntax

```
CALL mysql.rds_set_external_source_with_auto_position (  
    host_name  
    , host_port  
    , replication_user_name  
    , replication_user_password  
    , ssl_encryption  
)
```

Parameters

host_name

The host name or IP address of the MySQL instance running external to Aurora to become the replication source.

host_port

The port used by the MySQL instance running external to Aurora to be configured as the replication source. If your network configuration includes Secure Shell (SSH) port replication that converts the port number, specify the port number that is exposed by SSH.

replication_user_name

The ID of a user with REPLICATION CLIENT and REPLICATION SLAVE permissions on the MySQL instance running external to Aurora. We recommend that you provide an account that is used solely for replication with the external instance.

replication_user_password

The password of the user ID specified in *replication_user_name*.

ssl_encryption

This option is not currently implemented. The default is 0.

Usage notes

For an Aurora MySQL DB cluster, you call this stored procedure while connected to the primary instance.

The administrative user must run the `mysql.rds_set_external_source_with_auto_position` procedure. The administrative user runs this procedure on the primary instance of an Aurora MySQL DB cluster that acts as a replication target. This can be the replication target of an external MySQL DB instance or an Aurora MySQL DB cluster.

For Aurora, this procedure is supported for Aurora MySQL version 2.04 and later MySQL 5.7-compatible versions. It's also supported for Aurora MySQL version 3. GTID-based replication isn't supported for Aurora MySQL 1.1 or 1.0.

Before you run `mysql.rds_set_external_source_with_auto_position`, configure the external MySQL DB instance to be a replication source. To connect to the external MySQL instance, specify values for *replication_user_name* and *replication_user_password*. These values must indicate a replication user that has REPLICATION CLIENT and REPLICATION SLAVE permissions on the external MySQL instance.

To configure an external MySQL instance as a replication source

1. Using the MySQL client of your choice, connect to the external MySQL instance and create a user account to be used for replication. The following is an example.

```
CREATE USER 'repl_user'@'mydomain.com' IDENTIFIED BY 'SomePassW0rd'
```

2. On the external MySQL instance, grant REPLICATION CLIENT and REPLICATION SLAVE privileges to your replication user. The following example grants REPLICATION CLIENT and REPLICATION SLAVE privileges on all databases for the 'repl_user' user for your domain.

```
GRANT REPLICATION CLIENT, REPLICATION SLAVE ON *.* TO 'repl_user'@'mydomain.com'  
IDENTIFIED BY 'SomePassW0rd'
```

When you call `mysql.rds_set_external_source_with_auto_position`, Amazon RDS records certain information. This information is the time, the user, and an action of "set master" in the `mysql.rds_history` and `mysql.rds_replication_status` tables.

To skip a specific GTID-based transaction that is known to cause a problem, you can use the [mysql.rds_skip_transaction_with_gtid \(p. 1082\)](#) stored procedure. For more information about working with GTID-based replication, see [Using GTID-based replication for Aurora MySQL \(p. 954\)](#).

Examples

When run on an Aurora primary instance, the following example configures the Aurora cluster to act as a read replica of an instance of MySQL running external to Aurora.

```
call mysql.rds_set_external_source_with_auto_position(
  'Externaldb.some.com',
  3306,
  'repl_user'@'mydomain.com',
  'SomePassWord');
```

mysql.rds_skip_transaction_with_gtid

Skips replication of a transaction with the specified global transaction identifier (GTID) on an Aurora primary instance.

You can use this procedure for disaster recovery when a specific GTID transaction is known to cause a problem. Use this stored procedure to skip the problematic transaction. Examples of problematic transactions include transactions that disable replication, delete important data, or cause the DB instance to become unavailable.

Syntax

```
CALL mysql.rds_skip_transaction_with_gtid (gtid_to_skip);
```

Parameters

gtid_to_skip

The GTID of the replication transaction to skip.

Usage notes

For an Aurora MySQL DB cluster, you call this stored procedure while connected to the primary instance.

The master user must run the `mysql.rds_skip_transaction_with_gtid` procedure.

For Aurora, this procedure is supported for Aurora MySQL version 2.04 and later MySQL 5.7-compatible versions. It's also supported for Aurora MySQL version 3. GTID-based replication isn't supported for Aurora MySQL 1.1 or 1.0.

Database engine updates for Amazon Aurora MySQL

Amazon Aurora releases updates regularly. Updates are applied to Aurora DB clusters during system maintenance windows. The timing when updates are applied depends on the region and maintenance window setting for the DB cluster, as well as the type of update.

Updates are applied to all instances in a DB cluster at the same time. An update requires a database restart on all instances in a DB cluster, so you experience 20 to 30 seconds of downtime, after which you can resume using your DB cluster or clusters. You can view or change your maintenance window settings from the [AWS Management Console](#).

Following, you can learn how to choose the right Aurora MySQL version for your cluster, how to specify the version when you create or upgrade a cluster, and the procedures to upgrade a cluster from one version to another with minimal interruption.

Topics

- [Aurora MySQL version numbers and special versions \(p. 1083\)](#)

- [Preparing for Amazon Aurora MySQL-Compatible Edition version 1 end of life \(p. 1086\)](#)
- [Upgrading Amazon Aurora MySQL DB clusters \(p. 1088\)](#)
- [Database engine updates for Amazon Aurora MySQL version 3 \(p. 1108\)](#)
- [Database engine updates for Amazon Aurora MySQL version 2 \(p. 1108\)](#)
- [Database engine updates for Amazon Aurora MySQL version 1 \(p. 1196\)](#)
- [Database engine updates for Aurora MySQL Serverless clusters \(p. 1247\)](#)
- [MySQL bugs fixed by Aurora MySQL database engine updates \(p. 1250\)](#)
- [Security vulnerabilities fixed in Amazon Aurora MySQL \(p. 1271\)](#)

Aurora MySQL version numbers and special versions

Although Aurora MySQL-Compatible Edition is compatible with the MySQL database engines, Aurora MySQL includes features and bug fixes that are specific to particular Aurora MySQL versions. Application developers can check the Aurora MySQL version in their applications by using SQL. Database administrators can check and specify Aurora MySQL versions when creating or upgrading Aurora MySQL DB clusters and DB instances.

Topics

- [Checking or specifying Aurora MySQL engine versions through AWS \(p. 1083\)](#)
- [Checking Aurora MySQL versions using SQL \(p. 1084\)](#)
- [Aurora MySQL long-term support \(LTS\) releases \(p. 1085\)](#)
- [Upgrade paths between 5.6-compatible and 5.7-compatible clusters \(p. 1085\)](#)

Checking or specifying Aurora MySQL engine versions through AWS

When you perform administrative tasks using the AWS Management Console, AWS CLI, or RDS API, you specify the Aurora MySQL version in a descriptive alphanumeric format.

Starting with Aurora MySQL 2.03.2 and 1.19.0, Aurora engine versions have the following syntax.

```
mysql-major-version.mysql_aurora.aurora-mysql-version
```

The *mysql-major-version-* portion is 5 . 6, 5 . 7, or 8 . 0. This value represents the version of the client protocol and general level of MySQL feature support for the corresponding Aurora MySQL version.

The *aurora-mysql-version* is a dotted value with three parts: the Aurora MySQL major version, the Aurora MySQL minor version, and the patch level. The major version is 1, 2, or 3. Those values represent Aurora MySQL compatible with MySQL 5.6, 5.7, or 8.0 respectively. The minor version represents the feature release within the 1.x, 2.x, or 3.x series. The patch level begins at 0 for each minor version, and represents the set of subsequent bug fixes that apply to the minor version. Occasionally, a new feature is incorporated into a minor version but not made visible immediately. In these cases, the feature undergoes fine-tuning and is made public in a later patch level.

All 1.x Aurora MySQL engine versions are wire-compatible with Community MySQL 5.6.10a. All 2.x Aurora MySQL engine versions are wire-compatible with Community MySQL 5.7.12. All 3.x Aurora MySQL engine versions are wire-compatible with MySQL 8.0.23.

For example, the engine versions for Aurora MySQL 3.01.0, 2.03.2, and 1.19.0 are the following.

```
8.0.mysql_aurora.3.01.0  
5.7.mysql_aurora.2.03.2
```

5.6.mysql_aurora.1.19.0

Note

There isn't a one-to-one correspondence between community MySQL versions and the Aurora MySQL 1.x and 2.x versions. For Aurora MySQL version 3, there is a more direct mapping. To check which bug fixes and new features are in a particular Aurora MySQL release, see [Database engine updates for Amazon Aurora MySQL version 3 \(p. 1108\)](#), [Database engine updates for Amazon Aurora MySQL version 2 \(p. 1108\)](#) and [Database engine updates for Amazon Aurora MySQL version 1 \(p. 1196\)](#). For a chronological list of new features and releases, see [Document history \(p. 1824\)](#). To check the minimum version required for a security-related fix, see [Security vulnerabilities fixed in Amazon Aurora MySQL \(p. 1271\)](#).

For Aurora MySQL 2.x, all versions 2.03.1 and lower are represented by the engine version 5.7.12. In the same way, all versions before 1.19.0 are represented by the engine version 5.6.10a. These older version designations don't include the 5.6.mysql_aurora prefix. When you specified 5.7.12 or 5.6.10a while creating or modifying a cluster, you got the highest version before the 2.03.2 and 1.19.0 versions where the version numbering changed. To determine the exact version number for those older versions, you used the SQL technique explained in [Checking Aurora MySQL versions using SQL \(p. 1084\)](#).

You specify the Aurora MySQL engine version in some AWS CLI commands and RDS API operations. For example, you specify the --engine-version option when you run the AWS CLI commands [create-db-cluster](#) and [modify-db-cluster](#). You specify the EngineVersion parameter when you run the RDS API operations [CreateDBCluster](#) and [ModifyDBCluster](#).

In Aurora MySQL 1.19.0 and higher or 2.03.2 and higher, the engine version in the AWS Management Console also includes the Aurora version. Upgrading the cluster changes the displayed value. This change helps you to specify and check the precise Aurora MySQL versions, without the need to connect to the cluster or run any SQL commands.

Tip

For Aurora clusters managed through AWS CloudFormation, this change in the EngineVersion setting can trigger actions by AWS CloudFormation. For information about how AWS CloudFormation treats changes to the EngineVersion setting, see [the AWS CloudFormation documentation](#).

Before Aurora MySQL 1.19.0 and 2.03.2, the process to update the engine version is to use the **Apply a Pending Maintenance Action** option for the cluster. This process doesn't change the Aurora MySQL engine version that the console displays. For example, suppose that you see an Aurora MySQL cluster with a reported engine version of 5.6.10a or 5.7.12. To find out the specific version, connect to the cluster and query the AURORA_VERSION system variable as described previously.

Checking Aurora MySQL versions using SQL

The Aurora version numbers that you can retrieve in your application using SQL queries use the format <major version>. <minor version>. <patch version>. You can get this version number for any DB instance in your Aurora MySQL cluster by querying the AURORA_VERSION system variable. To get this version number, use one of the following queries.

```
select aurora_version();
select @@aurora_version;
```

Those queries produce output similar to the following.

```
mysql> select aurora_version(), @@aurora_version;
+-----+-----+
| aurora_version() | @@aurora_version |
+-----+-----+
| 2.08.1           | 2.08.1          |
+-----+-----+
```

The version numbers that the console, CLI, and RDS API return by using the techniques described in [Checking or specifying Aurora MySQL engine versions through AWS \(p. 1083\)](#) are typically more descriptive. However, for versions before 2.03.2 and 1.19, AWS always returns the version numbers 5.7.12 or 5.6.10a. For those older versions, use the SQL technique to check the precise version number.

Aurora MySQL long-term support (LTS) releases

Each new Aurora MySQL version remains available for a certain amount of time for you to use when you create or upgrade a DB cluster. After this period, you must upgrade any clusters that use that version. You can manually upgrade your cluster before the support period ends, or Aurora can automatically upgrade it for you when its Aurora MySQL version is no longer supported.

Aurora designates certain Aurora MySQL versions as long-term support (LTS) releases. DB clusters that use LTS releases can stay on the same version longer and undergo fewer upgrade cycles than clusters that use non-LTS releases. Aurora supports each LTS release for at least one year after that release becomes available. When a DB cluster that's on an LTS release is required to upgrade, Aurora upgrades it to the next LTS release. That way, the cluster doesn't need to be upgraded again for a long time.

During the lifetime of an Aurora MySQL LTS release, new patch levels introduce fixes to important issues. The patch levels don't include any new features. You can choose whether to apply such patches to DB clusters running the LTS release. For certain critical fixes, Amazon might perform a managed upgrade to a patch level within the same LTS release. Such managed upgrades are performed automatically within the cluster maintenance window.

We recommend that you upgrade to the latest release, instead of using the LTS release, for most of your Aurora MySQL clusters. Doing so takes advantage of Aurora as a managed service and gives you access to the latest features and bug fixes. The LTS releases are intended for clusters with the following characteristics:

- You can't afford downtime on your Aurora MySQL application for upgrades outside of rare occurrences for critical patches.
- The testing cycle for the cluster and associated applications takes a long time for each update to the Aurora MySQL database engine.
- The database version for your Aurora MySQL cluster has all the DB engine features and bug fixes that your application needs.

The current LTS releases for Aurora MySQL are the following:

- Aurora MySQL version 2.07.*. For more details about this version, see [Aurora MySQL database engine updates 2021-11-24 \(version 2.07.7\) \(p. 1140\)](#).
- Aurora MySQL version 1.22.*. For more details about this version, see [Aurora MySQL database engine updates 2021-06-03 \(version 1.22.5\) \(p. 1204\)](#).

These older versions are also designated as LTS releases:

- Aurora MySQL version 2.04.
- Aurora MySQL version 1.19.

Upgrade paths between 5.6-compatible and 5.7-compatible clusters

For most Aurora MySQL 1.x and 2.x versions, you can upgrade a MySQL 5.6-compatible cluster to any version of a MySQL 5.7-compatible cluster.

However, if your cluster is running Aurora MySQL 1.23 or higher, any upgrade to Aurora MySQL version 2.x must be to Aurora MySQL 2.09 or higher. This restriction applies even when you upgrade by restoring a snapshot to create a new Aurora cluster. Aurora MySQL 1.23 includes improvements in Aurora storage. For example, the maximum size of the cluster volume is larger in Aurora MySQL 1.23 and later. Aurora MySQL 2.09 is the first 2.x version that has the same storage enhancements.

Preparing for Amazon Aurora MySQL-Compatible Edition version 1 end of life

Amazon Aurora MySQL-Compatible Edition version 1 (with MySQL 5.6 compatibility) is planned to reach end of life on February 28, 2023. Amazon advises that you upgrade all clusters running Aurora MySQL version 1 to Aurora MySQL version 2 (with MySQL 5.7 compatibility) or Aurora MySQL version 3 (with MySQL 8.0 compatibility). Do this before Aurora MySQL version 1 reaches the end of its support period.

For Aurora provisioned DB clusters, you can complete upgrades from Aurora MySQL version 1 to Aurora MySQL version 2 by several methods. You can find instructions for the in-place upgrade mechanism in [How to perform an in-place upgrade \(p. 1098\)](#). Another way to complete the upgrade is to take a snapshot of an Aurora MySQL version 1 cluster and restore the snapshot to an Aurora MySQL version 2 cluster. Or you can follow a multistep process that runs the old and new clusters side by side. For more details about each method, see [Upgrading from Aurora MySQL 1.x to 2.x \(p. 1094\)](#).

For Aurora provisioned DB clusters, you can complete upgrades from Aurora MySQL version 1 to Aurora MySQL version 3 by using a two-stage upgrade process. The first stage requires an upgrade from Aurora MySQL version 1 to Aurora MySQL version 2 using the methods described preceding. The second stage requires an upgrade from Aurora MySQL version 2 to Aurora MySQL version 3. To perform this upgrade, take a snapshot of an Aurora MySQL version 2 cluster and restore the snapshot to an Aurora MySQL version 3 cluster. For more details, see [Upgrading from Aurora MySQL 2.x to 3.x \(p. 1093\)](#).

You can find upcoming end-of-life dates for Aurora major versions in [Amazon Aurora versions \(p. 5\)](#). Amazon automatically upgrades any clusters that you don't upgrade yourself before the end-of-life date. After the end-of-life date, these automatic upgrades to the subsequent major version occur during a scheduled maintenance window for clusters.

The following are additional milestones for upgrading Aurora MySQL version 1 clusters that are reaching end of life. For each, the start time is 00:00 Universal Coordinated Time (UTC).

1. Now through February 28, 2023 – You can at any time start upgrades of Aurora MySQL version 1 (with MySQL 5.6 compatibility) clusters to Aurora MySQL version 2 (with MySQL 5.7 compatibility). From Aurora MySQL version 2, you can do a further upgrade to Aurora MySQL version 3 (with MySQL 8.0 compatibility) for Aurora provisioned DB clusters.
2. September 27, 2022 – After this time, you can't create new Aurora MySQL version 1 clusters or instances from either the AWS Management Console or the AWS Command Line Interface (AWS CLI). You also can't add new secondary Regions to an Aurora global database. This might affect your ability to recover from an unplanned outage as outlined in [Recovering an Amazon Aurora global database from an unplanned outage \(p. 267\)](#), because you can't complete steps 5 and 6 after this time. You can still do the following for existing Aurora MySQL version 1 clusters until February 28, 2023:
 - Restore a snapshot taken of an Aurora MySQL version 1 cluster.
 - Add read replicas.
 - Change instance configuration.
 - Perform point-in-time restore.
 - Create clones of existing version 1 clusters.
3. February 28, 2023 – After this time, we plan to automatically upgrade Aurora MySQL version 1 clusters to the default version of Aurora MySQL version 2 within a scheduled maintenance window that follows. Restoring Aurora MySQL version 1 DB snapshots results in an automatic upgrade of the restored cluster to the default version of Aurora MySQL version 2 at that time.

Upgrading between major versions requires more extensive planning and testing than for a minor version. The process can take substantial time. After the upgrade is finished, you also might have follow-up work to do. For example, you might need to follow up due to differences in SQL compatibility, the way certain MySQL-related features work, or parameter settings between the old and new versions.

To learn more about the methods, planning, testing, and troubleshooting of Aurora MySQL major version upgrades, be sure to thoroughly read [Upgrading the major version of an Aurora MySQL DB cluster \(p. 1093\)](#).

Finding clusters affected by this end-of-life process

To find clusters affected by this end-of-life process, use the following procedures.

Console

To find an Aurora MySQL version 1 cluster

1. Sign in to the AWS Management Console and open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. In the navigation pane, choose **Databases**.
3. In the **Filter by databases** box, enter **5.6**.
4. Check for Aurora MySQL in the engine column.

AWS CLI

To find clusters affected by this end-of-life process using the AWS CLI, call the [describe-db-clusters](#) command. You can use the sample script following.

Example

```
aws rds describe-db-clusters --include-share --query 'DBClusters[?Engine==`aurora`].{EV:EngineVersion, DBCI:DBClusterIdentifier, EM:EngineMode}' --output table --region us-east-1

+-----+-----+
|      DescribeDBClusters      |
+-----+-----+
|   DBCI    |   EM    |   EV    |
+-----+-----+
| my-database-1 | serverless | 5.6.10a |
+-----+-----+
```

RDS API

To find Aurora MySQL DB clusters running Aurora MySQL version 1, use the RDS [DescribeDBClusters](#) API operation with the following required parameters:

- **DescribeDBClusters**
 - **Filters.Filter.N**
 - **Name**
 - **engine**
 - **Values.Value.N**
 - **['aurora']**

Upgrading Amazon Aurora MySQL DB clusters

You can upgrade an Aurora MySQL DB cluster to get bug fixes, new Aurora MySQL features, or to change to an entirely new version of the underlying database engine. The following sections show how.

Tip

The type of upgrade that you do depends on how much downtime you can afford for your cluster, how much verification testing you plan to do, how important the specific bug fixes or new features are for your use case, and whether you plan to do frequent small upgrades or occasional upgrades that skip several intermediate versions. For each upgrade, you can change the major version, the minor version, and the patch level for your cluster. If you aren't familiar with the distinction between Aurora MySQL major versions, minor versions, and patch levels, you can read the background information at [Aurora MySQL version numbers and special versions \(p. 1083\)](#).

Topics

- [Upgrading the minor version or patch level of an Aurora MySQL DB cluster \(p. 1088\)](#)
- [Upgrading the major version of an Aurora MySQL DB cluster \(p. 1093\)](#)

Upgrading the minor version or patch level of an Aurora MySQL DB cluster

You can use the following methods to upgrade the minor version of a DB cluster or to patch a DB cluster:

- [Upgrading Aurora MySQL by modifying the engine version \(p. 1088\)](#) (for Aurora MySQL 1.19.0 and higher, or 2.03.2 and higher)
- [Enabling automatic upgrades between minor Aurora MySQL versions \(p. 1089\)](#)
- [Upgrading Aurora MySQL by applying pending maintenance to an Aurora MySQL DB cluster \(p. 1090\)](#) (before Aurora MySQL 1.19.0 or 2.03.2)

For information about how zero-downtime patching can reduce interruptions during the upgrade process, see [Using zero-downtime patching \(p. 1091\)](#).

Upgrading Aurora MySQL by modifying the engine version

Upgrading the minor version of an Aurora MySQL cluster applies additional fixes and new features to an existing cluster. You can do this type of upgrade for clusters that are running Amazon Aurora MySQL version 1.19.0 and higher, or 2.03.2 and higher.

This kind of upgrade applies to Aurora MySQL clusters where the original version and the upgraded version are both in the Aurora MySQL 1.x series, or both in the Aurora MySQL 2.x series. The process is fast and straightforward because it doesn't involve any conversion for the Aurora MySQL metadata or reorganization of your table data.

You perform this kind of upgrade by modifying the engine version of the DB cluster using the AWS Management Console, AWS CLI, or the RDS API. If your cluster is running Aurora MySQL 1.x, choose a higher 1.x version. If your cluster is running Aurora MySQL 2.x, choose a higher 2.x version.

Note

If you're performing a minor upgrade on an Aurora global database, upgrade all of the secondary clusters before you upgrade the primary cluster.

To modify the engine version of a DB cluster

- **By using the console** – Modify the properties of your cluster. In the **Modify DB cluster** window, change the Aurora MySQL engine version in the **DB engine version** box. If you aren't familiar with the general

procedure for modifying a cluster, follow the instructions at [Modifying the DB cluster by using the console, CLI, and API \(p. 372\)](#).

- **By using the AWS CLI** – Call the [modify-db-cluster](#) AWS CLI command, and specify the name of your DB cluster for the `--db-cluster-identifier` option and the engine version for the `--engine-version` option.

For example, to upgrade to Aurora MySQL version 2.03.2, set the `--engine-version` option to `5.7.mysql_aurora.2.03.2`. Specify the `--apply-immediately` option to immediately update the engine version for your DB cluster.

- **By using the RDS API** – Call the [ModifyDBCluster](#) API operation, and specify the name of your DB cluster for the `DBClusterIdentifier` parameter and the engine version for the `EngineVersion` parameter. Set the `ApplyImmediately` parameter to `true` to immediately update the engine version for your DB cluster.

Enabling automatic upgrades between minor Aurora MySQL versions

For an Amazon Aurora MySQL DB cluster, you can specify that Aurora upgrades the DB cluster automatically to new minor versions as those versions are released. You do so by enabling the automatic minor version upgrade property of the DB cluster using the AWS Management Console, AWS CLI, or the RDS API.

The automatic upgrades occur during the maintenance window for the database.

Important

Until August 2020, you could specify this setting for a DB instance that was part of an Aurora MySQL DB cluster, but the setting had no effect. Now, the setting does apply to Aurora MySQL. If you have clusters created before August 2020, check whether the DB instances in the cluster already had the **Enable auto minor version upgrade** setting enabled. If so, confirm that this setting is still appropriate and change it if not. Aurora only performs the automatic upgrade if all DB instances in your cluster have this setting enabled.

Automatic minor version upgrade applies also to clusters running the LTS version for Aurora MySQL 1.x or 2.x. To prevent those clusters from being automatically upgraded, make sure to turn off the **Enable auto minor version upgrade** setting.

Automatic minor version upgrade doesn't apply to the following kinds of Aurora MySQL clusters:

- Multi-master clusters.
- Clusters that are part of an Aurora global database.
- Clusters that have cross-Region replicas.

If any of the DB instances in a cluster don't have the auto minor version upgrade setting turned on, Aurora doesn't automatically upgrade any of the instances in that cluster. Make sure to keep that setting consistent for all the DB instances in the cluster.

The outage duration varies depending on workload, cluster size, the amount of binary log data, and if Aurora can use the zero-downtime patching (ZDP) feature. Aurora restarts the database cluster, so you might experience a short period of unavailability before resuming use of your cluster. In particular, the amount of binary log data affects recovery time. The DB instance processes the binary log data during recovery. Thus, a high volume of binary log data increases recovery time.

To enable automatic minor version upgrades for an Aurora MySQL DB cluster

1. Follow the general procedure to modify the DB instances in your cluster, as described in [Modify a DB instance in a DB cluster \(p. 373\)](#). Repeat this procedure for each DB instance in your cluster.
2. Do the following to enable automatic minor version upgrades for your cluster:

- **By using the console** – Complete the following steps:
 1. Sign in to the Amazon RDS console. choose **Databases**, and find the DB cluster where you want to turn automatic minor version upgrade on or off.
 2. Choose each DB instance in the DB cluster that you want to modify. Apply the following change for each DB instance in sequence:
 - a. Choose **Modify**.
 - b. Choose the **Enable auto minor version upgrade** setting. This setting is part of the **Maintenance** section.
 - c. Choose **Continue** and check the summary of modifications.
 - d. (Optional) Choose **Apply immediately** to apply the changes immediately.
 - e. On the confirmation page, choose **Modify DB instance**.
- **By using the AWS CLI** – Call the [modify-db-instance](#) AWS CLI command. Specify the name of your DB instance for the `--db-instance-identifier` option and `true` for the `--auto-minor-version-upgrade` option. Optionally, specify the `--apply-immediately` option to immediately enable this setting for your DB instance. Run a separate `modify-db-instance` command for each DB instance in the cluster.
- **By using the RDS API** – Call the [ModifyDBInstance](#) API operation and specify the name of your DB cluster for the `DBInstanceIdentifier` parameter and `true` for the `AutoMinorVersionUpgrade` parameter. Optionally, set the `ApplyImmediately` parameter to `true` to immediately enable this setting for your DB instance. Call a separate `ModifyDBInstance` operation for each DB instance in the cluster.

You can use a CLI command such as the following to check the status of the **Enable auto minor version upgrade** for all of the DB instances in your Aurora MySQL clusters.

```
aws rds describe-db-instances \
--query '*[]'.
{DBClusterIdentifier:DBClusterIdentifier,DBInstanceIdentifier:DBInstanceIdentifier,AutoMinorVersionUpgr...
```

That command produces output similar to the following.

```
[  
  {  
    "DBInstanceIdentifier": "db-t2-medium-instance",  
    "DBClusterIdentifier": "cluster-57-2020-06-03-6411",  
    "AutoMinorVersionUpgrade": true  
  },  
  {  
    "DBInstanceIdentifier": "db-t2-small-original-size",  
    "DBClusterIdentifier": "cluster-57-2020-06-03-6411",  
    "AutoMinorVersionUpgrade": false  
  },  
  {  
    "DBInstanceIdentifier": "instance-2020-05-01-2332",  
    "DBClusterIdentifier": "cluster-57-2020-05-01-4615",  
    "AutoMinorVersionUpgrade": true  
  },  
  ... output omitted ...
```

[Upgrading Aurora MySQL by applying pending maintenance to an Aurora MySQL DB cluster](#)

When upgrading to Aurora MySQL version 1.x versions, new database engine minor versions and patches show as an **available** maintenance upgrade for your DB cluster. You can upgrade or patch the database version of your DB cluster by applying the available maintenance action. We recommend applying the

update on a nonproduction DB cluster first, so that you can see how changes in the new version affect your instances and applications.

To apply pending maintenance actions

- **By using the console** – Complete the following steps:
 1. Sign in to the Amazon RDS console, choose **Databases**, and choose the DB cluster that shows the **available** maintenance upgrade.
 2. For **Actions**, choose **Upgrade now** to immediately update the database version for your DB cluster, or **Upgrade at next window** to update the database version for your DB cluster during the next DB cluster maintenance window.
- **By using the AWS CLI** – Call the [apply-pending-maintenance-action](#) AWS CLI command, and specify the Amazon Resource Name (ARN) for your DB cluster for the `--resource-id` option and `system-update` for the `--apply-action` option. Set the `--opt-in-type` option to `immediate` to immediately update the database version for your DB cluster, or `next-maintenance` to update the database version for your DB cluster during the next cluster maintenance window.
- **By using the RDS API** – Call the [ApplyPendingMaintenanceAction](#) API operation, and specify the ARN for your DB cluster for the `ResourceId` parameter and `system-update` for the `ApplyAction` parameter. Set the `OptInType` parameter to `immediate` to immediately update the database version for your DB cluster, or `next-maintenance` to update the database version for your instance during the next cluster maintenance window.

For more information on how Amazon RDS manages database and operating system updates, see [Maintaining an Amazon Aurora DB cluster \(p. 443\)](#).

Note

If your current Aurora MySQL version is 1.14.x but lower than 1.14.4, you can upgrade only to 1.14.4 (which supports db.r4 instance classes). Also, to upgrade from 1.14.x to a higher minor Aurora MySQL version, such as 1.17, the 1.14.x version must be 1.14.4.

Using zero-downtime patching

Performing upgrades for Aurora MySQL DB clusters involves the possibility of an outage when the database is shut down and while it's being upgraded. By default, if you start the upgrade while the database is busy, you lose all the connections and transactions that the DB cluster is processing. If you wait until the database is idle to perform the upgrade, you might have to wait a long time.

The zero-downtime patching (ZDP) feature attempts, on a best-effort basis, to preserve client connections through an Aurora MySQL upgrade. If ZDP completes successfully, application sessions are preserved and the database engine restarts while the upgrade is in progress. The database engine restart can cause a drop in throughput lasting for a few seconds to approximately one minute.

ZDP is available in Aurora MySQL 2.07.2 and higher 2.07 versions, and 2.10.0 and higher, compatible with MySQL 5.7, and 3.01.0 and higher, compatible with MySQL 8.0.

In Aurora MySQL version 2, ZDP only applies to Aurora MySQL DB instances that use the `db.t2` or `db.t3` instance classes. In Aurora MySQL version 3, ZDP applies to all instance classes.

You can see metrics of important attributes during ZDP in the MySQL error log. You can also see information about when Aurora MySQL uses ZDP or chooses not use ZDP on the **Events** page in the AWS Management Console.

In Aurora MySQL 2.10 and higher, Aurora can perform a zero-downtime patch when binary log replication is enabled. Aurora MySQL automatically drops the connection to the binlog target during a ZDP operation. Aurora MySQL automatically reconnects to the binlog target and resumes replication after the restart finishes.

ZDP also works in combination with the reboot enhancements in Aurora MySQL 2.10 and higher. Patching the writer DB instance automatically patches readers at the same time. After performing the patch, Aurora restores the connections on both the writer and reader DB instances. Before Aurora MySQL 2.10, ZDP applies only to the writer DB instance of a cluster.

ZDP might not complete successfully under the following conditions:

- Long-running queries or transactions are in progress. If Aurora can perform ZDP in this case, any open transactions are canceled.
- Open Secure Socket Layer (SSL) connections exist.
- Temporary tables or table locks are in use, for example while data definition language (DDL) statements run. If Aurora can perform ZDP in this case, any open transactions are canceled.
- Pending parameter changes exist.

If no suitable time window for performing ZDP becomes available because of one or more of these conditions, patching reverts to the standard behavior.

Although connections remain intact following a successful ZDP operation, some variables and features are reinitialized. The following kinds of information aren't preserved through a restart caused by zero-downtime patching:

- Global variables. Aurora restores session variables, but it doesn't restore global variables after the restart.
- Status variables. In particular, the uptime value reported by the engine status is reset after a restart that uses the ZDR or ZDP mechanisms.
- `LAST_INSERT_ID`.
- In-memory auto_increment state for tables. The in-memory auto-increment state is reinitialized. For more information about auto-increment values, see [MySQL Reference Manual](#).
- Diagnostic information from `INFORMATION_SCHEMA` and `PERFORMANCE_SCHEMA` tables. This diagnostic information also appears in the output of commands such as `SHOW PROFILE` and `SHOW PROFILES`.

The following activities related to zero-downtime restart are reported on the **Events** page:

- Attempting to upgrade the database with zero downtime.
- Attempt to upgrade the database with zero downtime finished. The event reports how long the process took. The event also reports how many connections were preserved during the restart and how many connections were dropped. You can consult the database error log to see more details about what happened during the restart.

The following table summarizes how ZDP works for upgrading from and to specific Aurora MySQL versions. The instance class of the DB instance also affects whether Aurora uses the ZDP mechanism.

Original version	Upgraded version	Does ZDP apply?
Aurora MySQL 1.*	Any	No
Aurora MySQL 2.*, before 2.07.2	Any	No

Original version	Upgraded version	Does ZDP apply?
Aurora MySQL 2.07.2, 2.07.3	2.07.4 and higher 2.07 versions, 2.10.*	Yes, on the writer instance for T2 and T3 instance classes only. Aurora only performs ZDP if a quiet point is found before a timeout occurs. After the timeout, Aurora performs a regular restart.
2.07.4 and higher 2.07 versions	2.10.*	Yes, on the writer instance for T2 and T3 instances only. Aurora rolls back transactions for active and idle transactions. Connections using SSL, temporary tables, table locks, or user locks are disconnected. Aurora might restart the engine and drop all connections if the engine takes too long to start after ZDP finishes.

Alternative blue-green upgrade technique

Blog post: [Performing major version upgrades for Aurora MySQL with minimum downtime](#).

Upgrading the major version of an Aurora MySQL DB cluster

In an Aurora MySQL version number such as 2.08.1, the 2 represents the major version. Aurora MySQL version 1 is compatible with MySQL 5.6. Aurora MySQL version 2 is compatible with MySQL 5.7. Aurora MySQL version 3 is compatible with MySQL 8.0.23.

Upgrading between major versions requires more extensive planning and testing than for a minor version. The process can take substantial time. After the upgrade is finished, you also might have followup work to do. For example, this might occur due to differences in SQL compatibility, the way certain MySQL-related features work, or parameter settings between the old and new versions.

Topics

- [Upgrading from Aurora MySQL 2.x to 3.x \(p. 1093\)](#)
- [Upgrading from Aurora MySQL 1.x to 2.x \(p. 1094\)](#)
- [Planning a major version upgrade for an Aurora MySQL cluster \(p. 1094\)](#)
- [Aurora MySQL major version upgrade paths \(p. 1095\)](#)
- [How the Aurora MySQL in-place major version upgrade works \(p. 1096\)](#)
- [How to perform an in-place upgrade \(p. 1098\)](#)
- [How in-place upgrades affect the parameter groups for a cluster \(p. 1099\)](#)
- [Changes to cluster properties between Aurora MySQL version 1 and 2 \(p. 1100\)](#)
- [In-place major upgrades for global databases \(p. 1101\)](#)
- [After the upgrade \(p. 1101\)](#)
- [Troubleshooting for Aurora MySQL in-place upgrade \(p. 1101\)](#)
- [Aurora MySQL in-place upgrade tutorial \(p. 1103\)](#)
- [Alternative blue-green upgrade technique \(p. 1107\)](#)

Upgrading from Aurora MySQL 2.x to 3.x

Currently, upgrading to Aurora MySQL version 3 requires restoring a snapshot of an Aurora MySQL version 2 cluster to create a new version 3 cluster. If your original cluster is running Aurora MySQL version 1, you first upgrade to version 2 and then use the snapshot restore technique to create the version 3 cluster. For general information about Aurora MySQL version 3 and the new features that you can use after you upgrade, see [Aurora MySQL version 3 compatible with MySQL 8.0 \(p. 748\)](#). For details and examples of performing this type of upgrade, see [Upgrade planning for Aurora MySQL version 3 \(p. 760\)](#) and [Upgrading to Aurora MySQL version 3 \(p. 760\)](#).

Tip

When you upgrade the major version of your cluster from 2.x to 3.x, the original cluster and the upgraded one both use the same `aurora-mysql` value for the `engine` attribute.

Upgrading from Aurora MySQL 1.x to 2.x

Upgrading the major version from 1.x to 2.x changes the `engine` attribute of the cluster from `aurora` to `aurora-mysql`. Make sure to update any AWS CLI or API automation that you use with this cluster to account for the changed `engine` value.

If you have a MySQL 5.6-compatible cluster and want to upgrade it to a MySQL-5.7 compatible cluster, you can do so by running an upgrade process on the cluster itself. This kind of upgrade is an *in-place upgrade*, in contrast to upgrades that you do by creating a new cluster. This technique keeps the same endpoint and other characteristics of the cluster. The upgrade is relatively fast because it doesn't require copying all your data to a new cluster volume. This stability helps to minimize any configuration changes in your applications. It also helps to reduce the amount of testing for the upgraded cluster, because the number of DB instances and their instance classes all stay the same.

The in-place upgrade mechanism involves shutting down your DB cluster while the operation takes place. Aurora performs a clean shutdown and completes outstanding operations such as transaction rollback and undo purge.

The in-place upgrade is convenient, because it is simple to perform and minimizes configuration changes to associated applications. For example, an in-place upgrade preserves the endpoints and set of DB instances for your cluster. However, the time needed for an in-place upgrade can vary depending on the properties of your schema and how busy the cluster is. Thus, depending on the needs for your cluster, you can choose between in-place upgrade, snapshot restore as described in [Restoring from a DB cluster snapshot \(p. 497\)](#), or other upgrade techniques such as the one described in [Alternative blue-green upgrade technique \(p. 1107\)](#).

If your cluster is running a version that's lower than 1.22.3, the upgrade might take longer because Aurora MySQL automatically performs an upgrade to 1.22.3 as a first step. To minimize downtime during the major version upgrade, you can do an initial minor version upgrade to Aurora MySQL 1.22.3 in advance.

Planning a major version upgrade for an Aurora MySQL cluster

To make sure that your applications and administration procedures work smoothly after upgrading a cluster between major versions, you can do some advance planning and preparation. To see what sorts of management code to update for your AWS CLI scripts or RDS API-based applications, see [How in-place upgrades affect the parameter groups for a cluster \(p. 1099\)](#) and [Changes to cluster properties between Aurora MySQL version 1 and 2 \(p. 1100\)](#).

You can learn the sorts of issues that you might encounter during the upgrade by reading [Troubleshooting for Aurora MySQL in-place upgrade \(p. 1101\)](#). For issues that might cause the upgrade to take a long time, you can test those conditions in advance and correct them.

To verify application compatibility, performance, maintenance procedures, and similar considerations for the upgraded cluster, you can perform a simulation of the upgrade before doing the real upgrade. This technique can be especially useful for production clusters. Here, it's important to minimize downtime and have the upgraded cluster ready to go as soon as the upgrade is finished.

Note

This technique applies to upgrades from Aurora MySQL version 1 to version 2. Currently, you can't upgrade from Aurora MySQL version 2 to 3 by using cloning.

Use the following steps:

1. Create a clone of the original cluster. Follow the procedure in [Cloning a volume for an Aurora DB cluster \(p. 402\)](#).

2. Set up a similar set of writer and reader DB instances as in the original cluster.
3. Perform an in-place upgrade of the cloned cluster. Follow the procedure in [How to perform an in-place upgrade \(p. 1098\)](#). Start the upgrade immediately after creating the clone. That way, the cluster volume is still identical to the state of the original cluster. If the clone sits idle before you do the upgrade, Aurora performs database cleanup processes in the background. In that case, the upgrade of the clone isn't an accurate simulation of upgrading the original cluster.
4. Test application compatibility, performance, administration procedures, and so on, using the cloned cluster.
5. If you encounter any issues, adjust your upgrade plans to account for them. For example, adapt any application code to be compatible with the feature set of the higher version. Estimate how long the upgrade is likely to take based on the amount of data in your cluster. You might also choose to schedule the upgrade for a time when the cluster isn't busy.
6. After you are satisfied that your applications and workload work properly with the test cluster, you can perform the in-place upgrade for your production cluster.
7. To minimize the total downtime of your cluster during a major version upgrade, make sure that the workload on the cluster is low or zero at the time of the upgrade. In particular, make sure that there are no long running transactions in progress when you start the upgrade.

Aurora MySQL major version upgrade paths

Not all kinds or versions of Aurora MySQL clusters can use the in-place upgrade mechanism. You can learn the appropriate upgrade path for each Aurora MySQL cluster by consulting the following table.

Type of Aurora MySQL DB cluster	Can it use in-place upgrade?	Action
Aurora MySQL provisioned cluster, 1.22.3 or higher	Yes	This is the fastest upgrade path. Aurora doesn't need to perform an intermediate upgrade first.
Aurora MySQL provisioned cluster, earlier than 1.22.3	Yes	The upgrade might take longer than if the cluster is already running Aurora MySQL 1.22.3 or higher. During a major version upgrade, Aurora MySQL performs some database cleanup using a minimum Aurora MySQL version of 1.22.3. Aurora MySQL automatically performs an upgrade to 1.22.3 as a first step before doing that cleanup.
Aurora MySQL provisioned cluster, 2.0 or higher	No	In-place upgrade is only for 5.6-compatible Aurora MySQL clusters, to make possible compatibility with MySQL 5.7. Aurora MySQL version 2 is already compatible with 5.7. Use the procedure for upgrading the minor version or patch level to change from one 5.7-compatible version to another.
Aurora MySQL provisioned cluster, 3.1.0 or higher	No	For information about upgrading to Aurora MySQL version 3, see Upgrade planning for Aurora MySQL version 3 (p. 760) and Upgrading to Aurora MySQL version 3 (p. 760) .
Aurora MySQL provisioned cluster, 3.1.0 or higher	No	For information about upgrading to Aurora MySQL version 3, see Upgrade planning for Aurora MySQL version 3 (p. 760) and Upgrading to Aurora MySQL version 3 (p. 760) .
Aurora Serverless cluster	No	Make a snapshot of the 5.6-compatible Aurora Serverless cluster. Restore the snapshot to a 5.7-compatible cluster.

Type of Aurora MySQL DB cluster	Can it use in-place upgrade?	Action
		You can choose to make the new cluster Aurora Serverless or some other kind of 5.7-compatible cluster.
Cluster in an Aurora global database	Yes	Follow the procedure for doing an in-place upgrade for clusters in an Aurora global database. Perform the upgrade on the primary cluster in the global database. Aurora upgrades the primary cluster and all the secondary clusters in the global database at the same time. If you use the AWS CLI or RDS API, call the <code>modify-global-cluster</code> command or <code>ModifyGlobalCluster</code> operation instead of <code>modify-db-cluster</code> or <code>ModifyDBCluster</code> .
Multi-master cluster	No	Currently, multi-master replication isn't available for Aurora MySQL 5.7-compatible clusters.
Parallel query cluster	Maybe	If you have an existing parallel query cluster using an older Aurora MySQL version, upgrade the cluster to Aurora MySQL 1.23 first. Follow the procedure in Upgrade considerations for parallel query (p. 893) . You make some changes to configuration parameters to turn parallel query back on after this initial upgrade. Then you can perform an in-place upgrade. In this case, choose 2.09.1 or higher for the Aurora MySQL version.
Cluster that is the target of binary log replication	Maybe	If the binary log replication is from a 5.6-compatible Aurora MySQL cluster, you can perform an in-place upgrade. You can't perform the upgrade if the binary log replication is from an RDS MySQL or an on-premises MySQL DB instance. In that case, you can upgrade using the snapshot restore mechanism.
Cluster with zero DB instances	No	Using the AWS CLI or the RDS API, you can create an Aurora MySQL cluster without any attached DB instances. In the same way, you can also remove all DB instances from an Aurora MySQL cluster while leaving the data in the cluster volume intact. While a cluster has zero DB instances, you can't perform an in-place upgrade. The upgrade mechanism requires a writer instance in the cluster to perform conversions on the system tables, data files, and so on. In this case, use the AWS CLI or the RDS API to create a writer instance for the cluster. Then you can perform an in-place upgrade.
Cluster with backtrack enabled	Yes	You can perform an in-place upgrade for an Aurora MySQL cluster that uses the backtrack feature. However, after the upgrade, you can't backtrack the cluster to a time before the upgrade.

How the Aurora MySQL in-place major version upgrade works

Aurora MySQL performs a major version upgrade as a multistage process. You can check the current status of an upgrade. Some of the upgrade steps also provide progress information. As each stage

begins, Aurora MySQL records an event. You can examine events as they occur on the [Events](#) page in the RDS console. For more information about working with events, see [Using Amazon RDS event notification \(p. 675\)](#).

Important

Once the process begins, it runs until the upgrade either succeeds or fails. You can't cancel the upgrade while it's underway. If the upgrade fails, Aurora rolls back all the changes and your cluster has the same engine version, metadata, and so on as before.

The upgrade process consists of these stages:

1. Aurora performs a series of checks before beginning the upgrade process. Your cluster keeps running while Aurora does these checks. For example, the cluster can't have any XA transactions in the prepared state or be processing any data definition language (DDL) statements. For example, you might need to shut down applications that are submitting certain kinds of SQL statements. Or you might simply wait until certain long-running statements are finished. Then try the upgrade again. Some checks test for conditions that don't prevent the upgrade but might make the upgrade take a long time.

If Aurora detects that any required conditions aren't met, modify the conditions identified in the event details. Follow the guidance in [Troubleshooting for Aurora MySQL in-place upgrade \(p. 1101\)](#). If Aurora detects conditions that might cause a slow upgrade, plan to monitor the upgrade over an extended period.

2. Aurora takes your cluster offline. Then Aurora performs a similar set of tests as in the previous stage, to confirm that no new issues arose during the shutdown process. If Aurora detects any conditions at this point that would prevent the upgrade, Aurora cancels the upgrade and brings the cluster back online. In this case, confirm when the conditions no longer apply and start the upgrade again.
3. Aurora creates a snapshot of your cluster volume. Suppose that you discover compatibility or other kinds of issues after the upgrade is finished. Or suppose that you want to perform testing using both the original and upgraded clusters. In such cases, you can restore from this snapshot to create a new cluster with the original engine version and the original data.

Tip

This snapshot is a manual snapshot. However, Aurora can create it and continue with the upgrade process even if you have reached your quota for manual snapshots. This snapshot remains permanently until you delete it. After you finish all post-upgrade testing, you can delete this snapshot to minimize storage charges.

4. Aurora clones your cluster volume. Cloning is a fast operation that doesn't involve copying the actual table data. If Aurora encounters an issue during the upgrade, it reverts to the original data from the cloned cluster volume and brings the cluster back online. The temporary cloned volume during the upgrade isn't subject to the usual limit on the number of clones for a single cluster volume.
5. Aurora performs a clean shutdown for the writer DB instance. During the clean shutdown, progress events are recorded every 15 minutes for the following operations. You can examine events as they occur on the [Events](#) page in the RDS console.
 - Aurora purges the undo records for old versions of rows.
 - Aurora rolls back any uncommitted transactions.
6. Aurora upgrades the engine version on the writer DB instance:
 - Aurora installs the binary for the new engine version on the writer DB instance.
 - Aurora uses the writer DB instance to upgrade your data to MySQL 5.7-compatible format. During this stage, Aurora modifies the system tables and performs other conversions that affect the data in your cluster volume. In particular, Aurora upgrades the partition metadata in the system tables to be compatible with the MySQL 5.7 partition format. This stage can take a long time if the tables in your cluster have a large number of partitions.

If any errors occur during this stage, you can find the details in the MySQL error logs. After this stage starts, if the upgrade process fails for any reason, Aurora restores the original data from the cloned cluster volume.

7. Aurora upgrades the engine version on the reader DB instances.
8. The upgrade process is completed. Aurora records a final event to indicate that the upgrade process completed successfully. Now your DB cluster is running the new major version.

How to perform an in-place upgrade

Console

To upgrade the major version of an Aurora MySQL DB cluster

1. (Optional) Review the background material in [How the Aurora MySQL in-place major version upgrade works \(p. 1096\)](#). Perform any pre-upgrade planning and testing, as described in [Planning a major version upgrade for an Aurora MySQL cluster \(p. 1094\)](#).
2. Sign in to the AWS Management Console and open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
3. If you used a custom parameter group with the original 1.x cluster, create a corresponding MySQL 5.7-compatible parameter group. Make any necessary adjustments to the configuration parameters in that new parameter group. For more information, see [How in-place upgrades affect the parameter groups for a cluster \(p. 1099\)](#).
4. In the navigation pane, choose **Databases**.
5. In the list, choose the DB cluster that you want to modify.
6. Choose **Modify**.
7. For **Version**, choose an Aurora MySQL 2.x version.
8. Choose **Continue**.
9. On the next page, specify when to perform the upgrade. Choose **During the next scheduled maintenance window or Immediately**.
10. (Optional) Periodically examine the **Events** page in the RDS console during the upgrade. Doing so helps you to monitor the progress of the upgrade and identify any issues. If the upgrade encounters any issues, consult [Troubleshooting for Aurora MySQL in-place upgrade \(p. 1101\)](#) for the steps to take.
11. If you created a new MySQL 5.7-compatible parameter group at the start of this procedure, associate the custom parameter group with your upgraded cluster. For more information, see [How in-place upgrades affect the parameter groups for a cluster \(p. 1099\)](#).

Note

Performing this step requires you to restart the cluster again to apply the new parameter group.

12. (Optional) After you complete any post-upgrade testing, delete the manual snapshot that Aurora created at the beginning of the upgrade.

AWS CLI

To upgrade the major version of an Aurora MySQL DB cluster, use the AWS CLI `modify-db-cluster` command with the following required parameters:

- `--db-cluster-identifier`
- `--engine aurora-mysql`
- `--engine-version`
- `--allow-major-version-upgrade`
- `--apply-immediately` or `--no-apply-immediately`

If your cluster uses any custom parameter groups, also include one or both of the following options:

- `--db-cluster-parameter-group-name`, if the cluster uses a custom cluster parameter group
- `--db-instance-parameter-group-name`, if any instances in the cluster use a custom DB parameter group

The following example upgrades the `sample-cluster` DB cluster to Aurora MySQL version 2.09.0. The upgrade happens immediately, instead of waiting for the next maintenance window.

Example

For Linux, macOS, or Unix:

```
aws rds modify-db-cluster \
    --db-cluster-identifier sample-cluster \
    --engine aurora-mysql \
    --engine-version 5.7.mysql_aurora.2.09.0 \
    --allow-major-version-upgrade \
    --apply-immediately
```

For Windows:

```
aws rds modify-db-cluster ^
    --db-cluster-identifier sample-cluster ^
    --engine aurora-mysql ^
    --engine-version 5.7.mysql_aurora.2.09.0 ^
    --allow-major-version-upgrade ^
    --apply-immediately
```

You can combine other CLI commands with `modify-db-cluster` to create an automated end-to-end process for performing and verifying upgrades. For more information and examples, see [Aurora MySQL in-place upgrade tutorial \(p. 1103\)](#).

Note

If your cluster is part of an Aurora global database, the in-place upgrade procedure is slightly different. You call the [modify-global-cluster](#) command operation instead of `modify-db-cluster`. For more information, see [In-place major upgrades for global databases \(p. 1101\)](#).

RDS API

To upgrade the major version of an Aurora MySQL DB cluster, use the RDS API [ModifyDBCluster](#) operation with the following required parameters:

- `DBClusterIdentifier`
- `Engine`
- `EngineVersion`
- `AllowMajorVersionUpgrade`
- `ApplyImmediately` (set to `true` or `false`)

Note

If your cluster is part of an Aurora global database, the in-place upgrade procedure is slightly different. You call the [ModifyGlobalCluster](#) operation instead of `ModifyDBCluster`. For more information, see [In-place major upgrades for global databases \(p. 1101\)](#).

How in-place upgrades affect the parameter groups for a cluster

Aurora parameter groups have different sets of configuration settings for clusters that are compatible with MySQL 5.6 or 5.7. When you perform an in-place upgrade, the upgraded cluster and all its instances

must use corresponding 5.7-compatible cluster and instance parameter groups. If your cluster and instances use the default 5.6-compatible parameter groups, the upgraded cluster and instance start with the default 5.7-compatible parameter groups. If your cluster and instances use any custom parameter groups, you must create corresponding 5.7-compatible parameter groups and specify those during the upgrade process.

If your original cluster uses a custom 5.6-compatible cluster parameter group, create a corresponding 5.7-compatible cluster parameter group. You associate that parameter group with the cluster as part of the upgrade process.

Similarly, create any corresponding 5.7-compatible DB parameter group. You associate that parameter group with all the DB instances in the cluster as part of the upgrade process.

Important

If you specify any custom parameter group during the upgrade process, you must manually reboot the cluster after the upgrade finishes. Doing so makes the cluster begin using your custom parameter settings.

Changes to cluster properties between Aurora MySQL version 1 and 2

For MySQL 5.6-compatible clusters, the value that you use for the `engine` parameter in AWS CLI commands or RDS API operations is `aurora`. For MySQL 5.7-compatible or MySQL 8.0-compatible clusters, the corresponding value is `aurora-mysql`. When you upgrade from Aurora MySQL version 1 to version 2 or version 3, make sure to change any applications or scripts you use to set up or manage Aurora MySQL clusters and DB instances.

Also, change your code that manipulates parameter groups to account for the fact that the default parameter group names are different for MySQL 5.6-, 5.7-, and 8.0-compatible clusters. The default parameter group name for Aurora MySQL version 1 clusters is `default.aurora5.6`. The corresponding parameter group names for Aurora MySQL version 2 and 3 clusters are `default.aurora-mysql5.7` and `default.aurora-mysql8.0`.

For example, you might have code like the following that applies to your cluster before an upgrade.

```
# Add a new DB instance to a MySQL 5.6-compatible cluster.  
create-db-instance --db-instance-identifier instance-2020-04-28-6889 --db-cluster-  
identifier cluster-2020-04-28-2690 \  
--db-instance-class db.t2.small --engine aurora --region us-east-1  
  
# Find the Aurora MySQL v1.x versions available for minor version upgrades and patching.  
aws rds describe-orderable-db-instance-options --engine aurora --region us-east-1 \  
--query 'OrderableDBInstanceOptions[].[EngineVersion:EngineVersion]' --output text  
  
# Check the default parameter values for MySQL 5.6-compatible clusters.  
aws rds describe-db-parameters --db-parameter-group-name default.aurora5.6 --region us-  
east-1
```

After upgrading the major version of the cluster, modify that code as follows.

```
# Add a new DB instance to a MySQL 5.7-compatible cluster.  
create-db-instance --db-instance-identifier instance-2020-04-28-3333 --db-cluster-  
identifier cluster-2020-04-28-2690 \  
--db-instance-class db.t2.small --engine aurora-mysql --region us-east-1  
  
# Find the Aurora MySQL v2.x versions available for minor version upgrades and patching.  
aws rds describe-orderable-db-instance-options --engine aurora-mysql --region us-east-1 \  
--query 'OrderableDBInstanceOptions[].[EngineVersion:EngineVersion]' --output text  
  
# Check the default parameter values for MySQL 5.7-compatible clusters.  
aws rds describe-db-parameters --db-parameter-group-name default.aurora-mysql5.7 --region  
us-east-1
```

In-place major upgrades for global databases

For an Aurora global database, you upgrade the global database cluster. Aurora automatically upgrades all of the clusters at the same time and makes sure that they all run the same engine version. This requirement is because any changes to system tables, data file formats, and so on, are automatically replicated to all the secondary clusters.

Follow the instructions in [How the Aurora MySQL in-place major version upgrade works \(p. 1096\)](#). When you specify what to upgrade, make sure to choose the global database cluster instead of one of the clusters it contains.

If you use the AWS Management Console, choose the item with the role **Global database**.

DB identifier	Role	Engine
global-cluster	Global database	Aurora MySQL
cluster1	Primary cluster	Aurora MySQL
dbinstance-1	Writer instance	Aurora MySQL
cluster-2	Secondary cluster	Aurora MySQL
dbinstance-2	Reader instance	Aurora MySQL

If you use the AWS CLI or RDS API, start the upgrade process by calling the `modify-global-cluster` command or `ModifyGlobalCluster` operation instead of `modify-db-cluster` or `ModifyDBCluster`.

After the upgrade

If the cluster you upgraded had the backtrack feature enabled, you can't backtrack the upgraded cluster to a time that's before the upgrade.

Troubleshooting for Aurora MySQL in-place upgrade

Troubleshooting for Aurora MySQL in-place upgrade

Reason for in-place upgrade being canceled or slow	Solution to allow in-place upgrade to complete within maintenance window	
Cluster has XA transactions in the prepared state	Aurora cancels the upgrade.	Commit or roll back all prepared XA transactions.
Cluster is processing any data definition language (DDL) statements	Aurora cancels the upgrade.	Consider waiting and performing the upgrade after all DDL statements are finished.

Reason for in-place upgrade being canceled or slow	Solution to allow in-place upgrade to complete within maintenance window	
Cluster has uncommitted changes for many rows	Upgrade might take a long time.	The upgrade process rolls back the uncommitted changes. The indicator for this condition is the value of <code>TRX_ROWS_MODIFIED</code> in the <code>INFORMATION_SCHEMA.INNODB_TRX</code> table. Consider performing the upgrade only after all large transactions are committed or rolled back.
Cluster has high number of undo records	Upgrade might take a long time.	Even if the uncommitted transactions don't affect a large number of rows, they might involve a large volume of data. For example, you might be inserting large BLOBs. Aurora doesn't automatically detect or generate an event for this kind of transaction activity. The indicator for this condition is the history list length. The upgrade process rolls back the uncommitted changes. Consider performing the upgrade only after the history list length is smaller.
Cluster is in the process of committing a large binary log transaction	Upgrade might take a long time.	The upgrade process waits until the binary log changes are applied. More transactions or DDL statements could start during this period, further slowing down the upgrade process. Schedule the upgrade process when the cluster isn't busy with generating binary log replication changes. Aurora doesn't automatically detect or generate an event for this condition.

You can use the following steps to perform your own checks for some of the conditions in the preceding table. That way, you can schedule the upgrade at a time when you know the database is in a state where the upgrade can complete successfully and quickly.

- You can check for open XA transactions by executing the `XA RECOVER` statement. You can then commit or roll back the XA transactions before starting the upgrade.
- You can check for DDL statements by executing a `SHOW PROCESSLIST` statement and looking for `CREATE`, `DROP`, `ALTER`, `RENAME`, and `TRUNCATE` statements in the output. Allow all DDL statements to finish before starting the upgrade.
- You can check the total number of uncommitted rows by querying the `INFORMATION_SCHEMA.INNODB_TRX` table. The table contains one row for each transaction. The `TRX_ROWS_MODIFIED` column contains the number of rows modified or inserted by the transaction.
- You can check the length of the InnoDB history list by executing the `SHOW ENGINE INNODB STATUS` SQL statement and looking for the `History list length` in the output. You can also check the value directly by running the following query:

```
SELECT count FROM information_schema.innodb_metrics WHERE name = 'trx_rseg_history_len';
```

The length of the history list corresponds to the amount of undo information stored by the database to implement multi-version concurrency control (MVCC).

Aurora MySQL in-place upgrade tutorial

The following Linux examples show how you might perform the general steps of the in-place upgrade procedure using the AWS CLI.

This first example creates an Aurora DB cluster that's running a 1.x version of Aurora MySQL. The cluster includes a writer DB instance and a reader DB instance. The `wait db-instance-available` command pauses until the writer DB instance is available. That's the point when the cluster is ready to be upgraded.

```
$ aws rds create-db-cluster --db-cluster-identifier cluster-56-2020-11-17-3824 --engine aurora \
    --db-cluster-version 5.6.mysql_aurora.1.22.3
...
$ aws rds create-db-instance --db-instance-identifier instance-2020-11-17-7832 \
    --db-cluster-identifier cluster-56-2020-11-17-3824 --db-instance-class db.t2.medium --engine aurora
...
$ aws rds wait db-instance-available --db-instance-identifier instance-2020-11-17-7832 --region us-east-1
```

The Aurora MySQL 2.x versions that you can upgrade the cluster to depend on the 1.x version that the cluster is currently running and on the AWS Region where the cluster is located. The first command, with `--output text`, just shows the available target version. The second command shows the full JSON output of the response. In that detailed response, you can see details such as the `aurora-mysql` value you use for the `engine` parameter, and the fact that upgrading to 2.07.3 represents a major version upgrade.

```
$ aws rds describe-db-clusters --db-cluster-identifier cluster-56-2020-11-17-9355 \
    --query '[*].{EngineVersion:EngineVersion}' --output text
5.6.mysql_aurora.1.22.3

$ aws rds describe-db-engine-versions --engine aurora --engine-version
5.6.mysql_aurora.1.22.3 \
    --query '[*].[ValidUpgradeTarget]'
[
    [
        [
            {
                "Engine": "aurora-mysql",
                "EngineVersion": "5.7.mysql_aurora.2.07.3",
                "Description": "Aurora (MySQL 5.7) 2.07.3",
                "AutoUpgrade": false,
                "IsMajorVersionUpgrade": true
            }
        ]
    ]
]
```

This example shows how if you enter a target version number that isn't a valid upgrade target for the cluster, Aurora won't perform the upgrade. Aurora also won't perform a major version upgrade unless you include the `--allow-major-version-upgrade` parameter. That way, you can't accidentally perform an upgrade that has the potential to require extensive testing and changes to your application code.

```
$ aws rds modify-db-cluster --db-cluster-identifier cluster-56-2020-11-17-9355 \
    --engine-version 5.7.mysql_aurora.2.04.9 --region us-east-1 --apply-immediately
An error occurred (InvalidParameterCombination) when calling the ModifyDBCluster
operation: Cannot find upgrade target from 5.6.mysql_aurora.1.22.3 with requested version
5.7.mysql_aurora.2.04.9.

$ aws rds modify-db-cluster --db-cluster-identifier cluster-56-2020-11-17-9355 \
```

```
--engine-version 5.7.mysql_aurora.2.09.0 --region us-east-1 --apply-immediately
An error occurred (InvalidParameterCombination) when calling the ModifyDBCluster operation:
The AllowMajorVersionUpgrade flag must be present when upgrading to a new major version.

$ aws rds modify-db-cluster --db-cluster-identifier cluster-56-2020-11-17-9355 \
--engine-version 5.7.mysql_aurora.2.09.0 --region us-east-1 --apply-immediately --allow-
major-version-upgrade
{
    "DBClusterIdentifier": "cluster-56-2020-11-17-9355",
    "Status": "available",
    "Engine": "aurora",
    "EngineVersion": "5.6.mysql_aurora.1.22.3"
}
```

It takes a few moments for the status of the cluster and associated DB instances to change to upgrading. The version numbers for the cluster and DB instances only change when the upgrade is finished. Again, you can use the `wait db-instance-available` command for the writer DB instance to wait until the upgrade is finished before proceeding.

```
$ aws rds describe-db-clusters --db-cluster-identifier cluster-56-2020-11-17-9355 \
--query '*[].[Status,EngineVersion]' --output text
upgrading 5.6.mysql_aurora.1.22.3

$ aws rds describe-db-instances --db-instance-identifier instance-2020-11-17-5158 \
--query '*[.{
    DBInstanceIdentifier:DBInstanceIdentifier,DBInstanceStatus:DBInstanceState} | [0]]'
{
    "DBInstanceIdentifier": "instance-2020-11-17-5158",
    "DBInstanceStatus": "upgrading"
}

$ aws rds wait db-instance-available --db-instance-identifier instance-2020-11-17-5158
```

At this point, the version number for the cluster matches the one that was specified for the upgrade.

```
$ aws rds describe-db-clusters --region us-east-1 --db-cluster-identifier
cluster-56-2020-11-17-9355 \
--query '*[].[EngineVersion]' --output text
5.7.mysql_aurora.2.09.0
```

The preceding example did an immediate upgrade by specifying the `--apply-immediately` parameter. To let the upgrade happen at a convenient time when the cluster isn't expected to be busy, you can specify the `--no-apply-immediately` parameter. Doing so makes the upgrade start during the next maintenance window for the cluster. The maintenance window defines the period during which maintenance operations can start. A long-running operation might not finish during the maintenance window. Thus, you don't need to define a larger maintenance window even if you expect that the upgrade might take a long time.

The following example upgrades a cluster that's initially running Aurora MySQL version 1.22.2. In the `describe-db-engine-versions` output, the `False` and `True` values represent the `IsMajorVersionUpgrade` property. From version 1.22.2, you can upgrade to some other 1.* versions. Those upgrades aren't considered major version upgrades and so don't require an in-place upgrade. In-place upgrade is only available for upgrades to the 2.07 and 2.09 versions that are shown in the list.

```
$ aws rds describe-db-clusters --region us-east-1 --db-cluster-identifier
cluster-56-2020-11-17-3824 \
--query '*[].{EngineVersion:EngineVersion}' --output text
5.6.mysql_aurora.1.22.2
$ aws rds describe-db-engine-versions --engine aurora --engine-version
5.6.mysql_aurora.1.22.2 \
```

```
--query '*[].[ValidUpgradeTarget]|[0][0]|[*].[EngineVersion,IsMajorVersionUpgrade]' --
output text
5.6.mysql_aurora.1.22.3 False
5.6.mysql_aurora.1.23.0 False
5.6.mysql_aurora.1.23.1 False
5.7.mysql_aurora.2.07.1 True
5.7.mysql_aurora.2.07.1 True
5.7.mysql_aurora.2.07.2 True
5.7.mysql_aurora.2.07.3 True
5.7.mysql_aurora.2.09.1 True

$ aws rds modify-db-cluster --db-cluster-identifier cluster-56-2020-11-17-9355 \
--engine-version 5.7.mysql_aurora.2.09.0 --region us-east-1 --no-apply-immediately --
allow-major-version-upgrade
...
```

When a cluster is created without a specified maintenance window, Aurora picks a random day of the week. In this case, the `modify-db-cluster` command is submitted on a Monday. Thus, we change the maintenance window to be Tuesday morning. All times are represented in the UTC time zone. The `tue:10:00-tue:10:30` window corresponds to 2-2:30 AM Pacific time. The change in the maintenance window takes effect immediately.

```
$ aws rds describe-db-clusters --db-cluster-identifier cluster-56-2020-11-17-9355 --region
us-east-1 --query '*[].[PreferredMaintenanceWindow]'
[
    [
        "sat:08:20-sat:08:50"
    ]
]

$ aws rds modify-db-cluster --db-cluster-identifier cluster-56-2020-11-17-3824 --preferred-
maintenance-window tue:10:00-tue:10:30"
$ aws rds describe-db-clusters --db-cluster-identifier cluster-56-2020-11-17-3824 --region
us-east-1 --query '*[].[PreferredMaintenanceWindow]'
[
    [
        "tue:10:00-tue:10:30"
    ]
]
```

```
$ aws rds create-db-cluster --engine aurora --db-cluster-identifier
cluster-56-2020-11-17-9355 \
--region us-east-1 --master-username my_username --master-user-password my_password
{
    "DBClusterIdentifier": "cluster-56-2020-11-17-9355",
    "DBClusterArn": "arn:aws:rds:us-east-1:123456789012:cluster:cluster-56-2020-11-17-9355",
    "Engine": "aurora",
    "EngineVersion": "5.6.mysql_aurora.1.22.2",
    "Status": "creating",
    "Endpoint": "cluster-56-2020-11-17-9355.cluster-ccfbt21ixr91.us-east-1-
integ.rds.amazonaws.com",
    "ReaderEndpoint": "cluster-56-2020-11-17-9355.cluster-ro-ccfbt21ixr91.us-east-1-
integ.rds.amazonaws.com"
}

$ aws rds create-db-instance --db-instance-identifier instance-2020-11-17-5158 \
--db-cluster-identifier cluster-56-2020-11-17-9355 --db-instance-class db.r5.large --
region us-east-1 --engine aurora
{
    "DBInstanceIdentifier": "instance-2020-11-17-5158",
    "DBClusterIdentifier": "cluster-56-2020-11-17-9355",
    "DBInstanceClass": "db.r5.large",
    "DBInstanceStatus": "creating"
```

```

}

$ aws rds wait db-instance-available --db-instance-identifier instance-2020-11-17-5158 --
region us-east-1

```

The following example shows how to get a report of the events generated by the upgrade. The `--duration` argument represents the number of minutes to retrieve the event information. This argument is needed because by default, `describe-events` only returns events from the last hour.

```

$ aws rds describe-events --source-type db-cluster --source-identifier
cluster-56-2020-11-17-3824 --duration 20160
{
    "Events": [
        {
            "SourceIdentifier": "cluster-56-2020-11-17-3824",
            "SourceType": "db-cluster",
            "Message": "DB cluster created",
            "EventCategories": [
                "creation"
            ],
            "Date": "2020-11-17T01:24:11.093000+00:00",
            "SourceArn": "arn:aws:rds:us-
east-1:123456789012:cluster:cluster-56-2020-11-17-3824"
        },
        {
            "SourceIdentifier": "cluster-56-2020-11-17-3824",
            "SourceType": "db-cluster",
            "Message": "Upgrade in progress: Performing online pre-upgrade checks.",
            "EventCategories": [
                "maintenance"
            ],
            "Date": "2020-11-18T22:57:08.450000+00:00",
            "SourceArn": "arn:aws:rds:us-
east-1:123456789012:cluster:cluster-56-2020-11-17-3824"
        },
        {
            "SourceIdentifier": "cluster-56-2020-11-17-3824",
            "SourceType": "db-cluster",
            "Message": "Upgrade in progress: Performing offline pre-upgrade checks.",
            "EventCategories": [
                "maintenance"
            ],
            "Date": "2020-11-18T22:57:59.519000+00:00",
            "SourceArn": "arn:aws:rds:us-
east-1:123456789012:cluster:cluster-56-2020-11-17-3824"
        },
        {
            "SourceIdentifier": "cluster-56-2020-11-17-3824",
            "SourceType": "db-cluster",
            "Message": "Upgrade in progress: Creating pre-upgrade snapshot
[preupgrade-cluster-56-2020-11-17-3824-5-6-mysql-aurora-1-22-2-to-5-7-mysql-
aurora-2-09-0-2020-11-18-22-55].",
            "EventCategories": [
                "maintenance"
            ],
            "Date": "2020-11-18T23:00:22.318000+00:00",
            "SourceArn": "arn:aws:rds:us-
east-1:123456789012:cluster:cluster-56-2020-11-17-3824"
        },
        {
            "SourceIdentifier": "cluster-56-2020-11-17-3824",
            "SourceType": "db-cluster",
            "Message": "Upgrade in progress: Cloning volume.",
            "EventCategories": [

```

```

        "maintenance"
    ],
    "Date": "2020-11-18T23:01:45.428000+00:00",
    "SourceArn": "arn:aws:rds:us-
east-1:123456789012:cluster:cluster-56-2020-11-17-3824"
},
{
    "SourceIdentifier": "cluster-56-2020-11-17-3824",
    "SourceType": "db-cluster",
    "Message": "Upgrade in progress: Purging undo records for old row versions.
Records remaining: 164",
    "EventCategories": [
        "maintenance"
    ],
    "Date": "2020-11-18T23:02:25.141000+00:00",
    "SourceArn": "arn:aws:rds:us-
east-1:123456789012:cluster:cluster-56-2020-11-17-3824"
},
{
    "SourceIdentifier": "cluster-56-2020-11-17-3824",
    "SourceType": "db-cluster",
    "Message": "Upgrade in progress: Purging undo records for old row versions.
Records remaining: 164",
    "EventCategories": [
        "maintenance"
    ],
    "Date": "2020-11-18T23:06:23.036000+00:00",
    "SourceArn": "arn:aws:rds:us-
east-1:123456789012:cluster:cluster-56-2020-11-17-3824"
},
{
    "SourceIdentifier": "cluster-56-2020-11-17-3824",
    "SourceType": "db-cluster",
    "Message": "Upgrade in progress: Upgrading database objects.",
    "EventCategories": [
        "maintenance"
    ],
    "Date": "2020-11-18T23:06:48.208000+00:00",
    "SourceArn": "arn:aws:rds:us-
east-1:123456789012:cluster:cluster-56-2020-11-17-3824"
},
{
    "SourceIdentifier": "cluster-56-2020-11-17-3824",
    "SourceType": "db-cluster",
    "Message": "Database cluster major version has been upgraded",
    "EventCategories": [
        "maintenance"
    ],
    "Date": "2020-11-18T23:10:28.999000+00:00",
    "SourceArn": "arn:aws:rds:us-
east-1:123456789012:cluster:cluster-56-2020-11-17-3824"
}
]
}

```

Alternative blue-green upgrade technique

For situations where the top priority is to perform an immediate switchover from the old cluster to an upgraded one, you can use a multistep process that runs the old and new clusters side-by-side. In this case, you replicate data from the old cluster to the new one until you are ready for the new cluster to take over. For details, see this blog post: [Performing major version upgrades for Amazon Aurora MySQL with minimum downtime](#).

Database engine updates for Amazon Aurora MySQL version 3

The following are database engine updates for Amazon Aurora MySQL version 3.

Topics

- [Aurora MySQL database engine updates 2021-11-18 \(version 3.01.0, compatible with MySQL 8.0.23\) \(p. 1108\)](#)

Aurora MySQL database engine updates 2021-11-18 (version 3.01.0, compatible with MySQL 8.0.23)

Version: 3.01.0

Aurora MySQL 3.01.0 is generally available. Aurora MySQL 3.01 versions are compatible with MySQL 8.0.23, Aurora MySQL 2.x versions are compatible with MySQL 5.7, and Aurora MySQL 1.x versions are compatible with MySQL 5.6.

For details of new features in Aurora MySQL version 3 and differences between Aurora MySQL version 3 and Aurora MySQL version 2 or community MySQL 8.0, see [Comparison of Aurora MySQL version 2 and Aurora MySQL version 3 \(p. 749\)](#) and [Comparison of Aurora MySQL version 3 and community MySQL 8.0 \(p. 756\)](#).

Currently supported Aurora MySQL releases are 1.14.*, 1.15.*, 1.16.*, 1.17.*, 1.18.*, 1.19.*, 1.20.*, 1.21.*, 1.22.*, 1.23.*, 2.01.*, 2.02.*, 2.03.*, 2.04.*, 2.05.*, 2.06.*, 2.07.*, 2.08.*, 2.09.*, 2.10.*, and 3.01.*.

You can restore a snapshot from any currently supported Aurora MySQL version 2 cluster into Aurora MySQL 3.01.0.

For information on planning an upgrade to Aurora MySQL version 3, see [Upgrade planning for Aurora MySQL version 3 \(p. 760\)](#). For the upgrade procedure itself, see [Upgrading to Aurora MySQL version 3 \(p. 760\)](#). For general information about Aurora MySQL upgrades, see [Upgrading Amazon Aurora MySQL DB clusters \(p. 1088\)](#).

To create a cluster with an older version of Aurora MySQL, specify the engine version through the AWS Management Console, the AWS CLI, or the Amazon RDS API.

If you have any questions or concerns, AWS Support is available on the community forums and through [AWS Premium Support](#). For more information, see [Maintaining an Amazon Aurora DB cluster \(p. 443\)](#).

Improvements

Aurora MySQL version 3.01.0 is generally compatible with community MySQL 8.0.23. This version includes the security fixes for Common Vulnerabilities and Exposures (CVE) issues as of community MySQL 8.0.23.

Aurora MySQL version 3.01.0 contains all the Aurora-specific bug fixes through Aurora MySQL version 2.10.0.

For details of new features in Aurora MySQL version 3, see [Features from community MySQL 8.0 \(p. 748\)](#) and [New parallel query optimizations \(p. 749\)](#).

Database engine updates for Amazon Aurora MySQL version 2

The following are Amazon Aurora version 2 database engine updates:

- [Aurora MySQL database engine updates 2022-01-26 \(version 2.10.2\) \(p. 1110\)](#)
- [Aurora MySQL database engine updates 2021-10-21 \(version 2.10.1\) \(p. 1113\)](#)
- [Aurora MySQL database engine updates 2021-05-25 \(version 2.10.0\) \(p. 1115\)](#)
- [Aurora MySQL database engine updates 2021-11-12 \(version 2.09.3\) \(p. 1119\)](#)
- [Aurora MySQL database engine updates 2021-02-26 \(version 2.09.2\) \(p. 1122\)](#)
- [Aurora MySQL database engine updates 2020-12-11 \(version 2.09.1\) \(p. 1124\)](#)
- [Aurora MySQL database engine updates 2020-09-17 \(version 2.09.0\) \(p. 1127\)](#)
- [Aurora MySQL database engine updates 2022-01-06 \(version 2.08.4\) \(p. 1131\)](#)
- [Aurora MySQL database engine updates 2020-11-12 \(version 2.08.3\) \(p. 1133\)](#)
- [Aurora MySQL database engine updates 2020-08-28 \(version 2.08.2\) \(p. 1135\)](#)
- [Aurora MySQL database engine updates 2020-06-18 \(version 2.08.1\) \(p. 1136\)](#)
- [Aurora MySQL database engine updates 2020-06-02 \(version 2.08.0\) \(p. 1137\)](#)
- [Aurora MySQL database engine updates 2021-11-24 \(version 2.07.7\) \(p. 1140\)](#)
- [Aurora MySQL database engine updates 2021-09-02 \(version 2.07.6\) \(p. 1142\)](#)
- [Aurora MySQL database engine updates 2021-07-06 \(version 2.07.5\) \(p. 1143\)](#)
- [Aurora MySQL database engine updates 2021-03-04 \(version 2.07.4\) \(p. 1145\)](#)
- [Aurora MySQL database engine updates 2020-11-10 \(version 2.07.3\) \(p. 1147\)](#)
- [Aurora MySQL database engine updates 2020-04-17 \(version 2.07.2\) \(p. 1149\)](#)
- [Aurora MySQL database engine updates 2019-12-23 \(version 2.07.1\) \(p. 1151\)](#)
- [Aurora MySQL database engine updates 2019-11-25 \(version 2.07.0\) \(p. 1153\)](#)
- [Aurora MySQL database engine updates 2019-11-22 \(version 2.06.0\) \(deprecated\) \(p. 1155\)](#)
- [Aurora MySQL database engine updates 2019-11-11 \(version 2.05.0\) \(deprecated\) \(p. 1158\)](#)
- [Aurora MySQL database engine updates 2020-08-14 \(version 2.04.9\) \(p. 1159\)](#)
- [Aurora MySQL database engine updates 2019-11-20 \(version 2.04.8\) \(p. 1163\)](#)
- [Aurora MySQL database engine updates 2019-11-14 \(version 2.04.7\) \(p. 1164\)](#)
- [Aurora MySQL database engine updates 2019-09-19 \(version 2.04.6\) \(p. 1166\)](#)
- [Aurora MySQL database engine updates 2019-07-08 \(version 2.04.5\) \(p. 1168\)](#)
- [Aurora MySQL database engine updates 2019-05-29 \(version 2.04.4\) \(p. 1169\)](#)
- [Aurora MySQL database engine updates 2019-05-09 \(version 2.04.3\) \(p. 1170\)](#)
- [Aurora MySQL database engine updates 2019-05-02 \(version 2.04.2\) \(p. 1172\)](#)
- [Aurora MySQL database engine updates 2019-03-25 \(version 2.04.1\) \(p. 1173\)](#)
- [Aurora MySQL database engine updates 2019-03-25 \(version 2.04.0\) \(p. 1175\)](#)
- [Aurora MySQL database engine updates 2019-02-07 \(version 2.03.4\) \(deprecated\) \(p. 1176\)](#)
- [Aurora MySQL database engine updates 2019-01-18 \(version 2.03.3\) \(deprecated\) \(p. 1177\)](#)
- [Aurora MySQL database engine updates 2019-01-09 \(version 2.03.2\) \(deprecated\) \(p. 1179\)](#)
- [Aurora MySQL database engine updates 2018-10-24 \(version 2.03.1\) \(deprecated\) \(p. 1181\)](#)
- [Aurora MySQL database engine updates 2018-10-11 \(version 2.03\) \(deprecated\) \(p. 1182\)](#)
- [Aurora MySQL database engine updates 2018-10-08 \(version 2.02.5\) \(deprecated\) \(p. 1184\)](#)
- [Aurora MySQL database engine updates 2018-09-21 \(version 2.02.4\) \(deprecated\) \(p. 1185\)](#)
- [Aurora MySQL database engine updates 2018-08-23 \(version 2.02.3\) \(deprecated\) \(p. 1187\)](#)
- [Aurora MySQL database engine updates 2018-06-04 \(version 2.02.2\) \(deprecated\) \(p. 1189\)](#)
- [Aurora MySQL database engine updates 2018-05-03 \(version 2.02\) \(deprecated\) \(p. 1191\)](#)
- [Aurora MySQL database engine updates 2018-03-13 \(version 2.01.1\) \(deprecated\) \(p. 1193\)](#)
- [Aurora MySQL database engine updates 2018-02-06 \(version 2.01\) \(deprecated\) \(p. 1194\)](#)

Aurora MySQL database engine updates 2022-01-26 (version 2.10.2)

Version: 2.10.2

Aurora MySQL 2.10.2 is generally available. Aurora MySQL 2.x versions are compatible with MySQL 5.7, and Aurora MySQL 1.x versions are compatible with MySQL 5.6.

If you have any questions or concerns, AWS Support is available on the community forums and through [AWS Premium Support](#). For more information, see [Maintaining an Amazon Aurora DB cluster \(p. 443\)](#).

Improvements

Security fixes:

- [CVE-2021-35624](#)
- [CVE-2021-35604](#)
- [CVE-2021-2390](#)
- [CVE-2021-2389](#)
- [CVE-2021-2385](#)
- [CVE-2021-2356](#)
- [CVE-2019-17543](#)
- [CVE-2019-2960](#)

General improvements:

- Added a performance optimization to help reduce database IO latency in 24XL instance classes.
- Added support for ECDHE SSL ciphers. For more information on configuring your clients to use these SSL Ciphers please see the following MySQL documentation, [encrypted connection protocols ciphers](#)
- Fixed security issues related to Aurora MySQL integration with other AWS Services such as Amazon S3, Amazon ML, and AWS Lambda.
- Fixed an issue which can cause a database instance restart to fail when the database has approximately over 1GB of user and privilege combinations.
- Fixed an issue with Parallel Query which could cause the database to return incorrect groupings or sort order when executing queries with a GROUP BY clause and a WHERE clause that contain a range predicate.
- Fixed an issue which causes general_log and slow_log tables to become inaccessible after an in-place major version upgrade from Aurora MySQL 1.x (compatible with MySQL 5.6) to Aurora MySQL 2.x (compatible with MySQL 5.7).
- Fixed an issue which, in rare cases, causes the database instance to restart when innodb_trx, innodb_locks or innodb_lockwaits tables are queried while the database is under heavy workload. Monitoring tools such as Performance Insights may query such tables.
- Fixed an issue where the value of a TIMESTAMP column of an existing row is updated to the latest timestamp when all of the following conditions are satisfied:
 1. A trigger exists for the table.
 2. An INSERT is performed on the table that has an ON DUPLICATE KEY UPDATE clause.
 3. The inserted row causes a duplicate value violation in a UNIQUE index or PRIMARY KEY.
 4. One or more columns are of TIMESTAMP data type and have a default value of CURRENT_TIMESTAMP.
- Fixed an issue which, in rare cases, could prevent a binlog replica from connecting to an instance with binlog enabled.

- Fixed an issue where, in rare conditions, transactions were unable to commit when running on an instance with binlog enabled.
- Fixed an issue where new connections could not be established to an instance with binlog enabled.
- Fixed an issue which can cause excessive internal logging when attempting zero downtime patching and restart causing local storage to fill up.
- Fixed an issue that causes a binlog replica to stop with an HA_ERR_FOUND_DUPP_KEY error when replicating certain DDL and DCL statements. The issue occurs when the source instance is configured with MIXED binary logging format and READ COMMITTED or READ UNCOMMITTED isolation level.
- Fixed an issue where the binlog replication I/O thread is unable to keep up with the primary instance, when multi-threaded replication is enabled
- Fixed an issue where, in rare conditions, a high number of active connections to the database instance may cause the CloudWatch CommitLatency metric to be incorrectly reported.
- Fixed an issue which causes local storage on Graviton instances to fill up when performing LOAD FROM S3 or SELECT INTO S3.
- Fixed an issue which can cause wrong query results when querying a table with a foreign key and both of the following conditions are met:
 1. Query cache is enabled
 2. A transaction with a cascading delete or update on that table is rolled back
- Fixed an issue which, in rare conditions, can cause Aurora reader instances to restart. The chance of this issue occurring increases as the number of transaction rollbacks increases.
- Fixed an issue where the number of mutex 'LOCK_epoch_id_master' occurrences in Performance Schema increases when a session is opened and closed.
- Fixed an issue which can cause an increasing number of deadlocks for workloads which have many transactions updating the same set of rows concurrently.
- Fixed an issue which, in rare conditions, can cause the instances to restart when the database volume grows to a multiple of 160GB.
- Fixed an issue with Parallel Query which could cause the database to restart when executing SQL statements with a LIMIT clause.
- Fixed an issue which, in rare conditions, can cause the database instance to restart when using XA transactions with the READ COMMITTED isolation level.
- Fixed an issue where, after an Aurora Read instance restarts, it may restart again if there is a heavy DDL workload during the restart.
- Fixed an issue with incorrect reporting of Aurora reader replication lag.
- Fixed an issue which, in rare conditions, can cause a writer instance to restart when an in-memory data-integrity check fails.
- Fixed an issue which, in rare conditions, incorrectly shows the "Database Load" chart in Performance Insights (PI) sessions as actively using CPU even though the sessions have finished processing and are idle.
- Fixed an issue which, in rare conditions, can cause the database server to restart when a query is processed using Parallel Query.
- Fixed an issue which, in rare conditions, can cause the writer instance in a primary Global Database cluster to restart because of a race condition during Global Database replication.
- Fixed an issue that can occur during a database instance restart, which can cause more than one restart.

Integration of MySQL Community Edition bug fixes

- Fixed an issue in InnoDB where an error in code related to table statistics raised an assertion in the dictOstats.cc source file. (Bug #24585978)

- Fixed an issue where a secondary index over a virtual column became corrupted when the index was built online. For [UPDATE](#) statements, we fix this as follows: If the virtual column value of the index record is set to NULL, then we generate this value from the cluster index record. (Bug #30556595))
- Fixed an issue in InnoDB where deleting marked rows were able to acquire an external read lock before a partial rollback was completed. The external read lock prevented conversion of an implicit lock to an explicit lock during the partial rollback, causing an assertion failure. (Bug #29195848)
- Fixed an issue where the empty host names in accounts could cause the server to misbehave. (Bug #28653104)
- Fixed an issue in InnoDB where a query interruption during a lock wait caused an error. (Bug #28068293)
- Fixed an issue in replication where Interleaved transactions could sometimes deadlock the slave applier when the transaction isolation level was set to [REPEATABLE READ](#). (Bug #25040331)
- Fixed an issue which can cause binlog replicas to stall due to lock wait timeout. (Bug #27189701)

Comparison with Aurora MySQL version 1

The following Amazon Aurora MySQL features are supported in Aurora MySQL version 1 (compatible with MySQL 5.6), but these features are currently not supported in Aurora MySQL version 2 (compatible with MySQL 5.7).

- Asynchronous key prefetch (AKP). For more information, see [Optimizing Amazon Aurora indexed join queries with asynchronous key prefetch \(p. 1036\)](#).
- Hash joins. For more information, see [Optimizing large Aurora MySQL join queries with hash joins \(p. 1038\)](#).
- Native functions for synchronously invoking AWS Lambda functions. For more information, see [Invoking a Lambda function with an Aurora MySQL native function \(p. 1012\)](#).
- Scan batching. For more information, see [Aurora MySQL database engine updates 2017-12-11 \(p. 1225\)](#).
- Migrating data from MySQL using an Amazon S3 bucket. For more information, see [Migrating data from MySQL by using an Amazon S3 bucket \(p. 784\)](#).

MySQL 5.7 compatibility

This Aurora MySQL version is wire-compatible with MySQL 5.7 and includes features such as JSON support, spatial indexes, and generated columns. Aurora MySQL uses a native implementation of spatial indexing using z-order curves to deliver >20x better write performance and >10x better read performance than MySQL 5.7 for spatial datasets.

This Aurora MySQL version does not currently support the following MySQL 5.7 features:

- Group replication plugin
- Increased page size
- InnoDB buffer pool loading at startup
- InnoDB full-text parser plugin
- Multisource replication
- Online buffer pool resizing
- Password validation plugin
- Query rewrite plugins
- Replication filtering
- The `CREATE TABLESPACE` SQL statement

Aurora MySQL database engine updates 2021-10-21 (version 2.10.1)

Version: 2.10.1

Aurora MySQL 2.10.1 is generally available. Aurora MySQL 2.x versions are compatible with MySQL 5.7, and Aurora MySQL 1.x versions are compatible with MySQL 5.6.

Currently supported Aurora MySQL releases are 1.14.*, 1.15.*, 1.16.*, 1.17.*, 1.18.*, 1.19.*, 1.20.*, 1.21.*, 1.22.*, 1.23.*, 2.01.*, 2.02.*, 2.03.*, 2.04.*, 2.05.*, 2.06.*, 2.07.*, 2.08.*, 2.09.*, and 2.10.*.

You can upgrade an existing Aurora MySQL 2.* database cluster to Aurora MySQL 2.10.1. For clusters running Aurora MySQL version 1, you can upgrade an existing Aurora MySQL 1.23 or higher cluster directly to 2.10.1. You can also restore a snapshot from any currently supported Aurora MySQL release into Aurora MySQL 2.10.1.

To create a cluster with an older version of Aurora MySQL, specify the engine version through the AWS Management Console, the AWS CLI, or the Amazon RDS API.

If you have any questions or concerns, AWS Support is available on the community forums and through [AWS Premium Support](#). For more information, see [Maintaining an Amazon Aurora DB cluster \(p. 443\)](#).

Note

For information on how to upgrade your Aurora MySQL database cluster, see [Upgrading the minor version or patch level of an Aurora MySQL DB cluster \(p. 1088\)](#).

Improvements

Security fixes:

- [CVE-2021-2194](#)
- [CVE-2021-2174](#)
- [CVE-2021-2154](#)
- [CVE-2021-2307](#)
- [CVE-2021-2226](#)
- [CVE-2021-2171](#)
- [CVE-2021-2169](#)
- [CVE-2021-2166](#)
- [CVE-2021-2160](#)
- [CVE-2021-2060](#)
- [CVE-2021-2032](#)
- [CVE-2021-2001](#)

Availability improvements:

- Added the ability to cleanly shut down the cluster for future major version upgrades.

General improvements:

- Fixed an issue that can cause high CPU consumption on the reader instances due to excessive logging of informational messages in internal diagnostic log files.
- Fixed an issue where the value of a TIMESTAMP column of an existing row is updated to the latest timestamp when all of the following conditions are satisfied:

1. A trigger exists for the table.
 2. An INSERT is performed on the table that has an ON DUPLICATE KEY UPDATE clause.
 3. The inserted row causes a duplicate value violation in a UNIQUE index or PRIMARY KEY.
 4. One or more columns are of TIMESTAMP data type and have a default value of CURRENT_TIMESTAMP.
- Fixed an issue introduced in version 2.10.0 which causes use of json_merge function to raise an error code in certain cases. In particular, when json_merge function is used in a DDL containing generated columns, it can return error code 1305.
 - Fixed an issue where, in rare conditions, read replicas restarts when a large object's update history is being validated for a transaction's read view on the read replica.
 - Fixed an issue which, in rare conditions, causes a writer instance to restart when an in-memory data-integrity check fails.

Integration of MySQL community edition bug fixes

- CURRENT_TIMESTAMP PRODUCES ZEROS IN TRIGGER. (Bug #25209512)

Comparison with Aurora MySQL version 1

The following Amazon Aurora MySQL features are supported in Aurora MySQL version 1 (compatible with MySQL 5.6), but these features are currently not supported in Aurora MySQL version 2 (compatible with MySQL 5.7).

- Asynchronous key prefetch (AKP). For more information, see [Optimizing Amazon Aurora indexed join queries with asynchronous key prefetch \(p. 1036\)](#).
- Hash joins. For more information, see [Optimizing large Aurora MySQL join queries with hash joins \(p. 1038\)](#).
- Native functions for synchronously invoking AWS Lambda functions. For more information, see [Invoking a Lambda function with an Aurora MySQL native function \(p. 1012\)](#).
- Scan batching. For more information, see [Aurora MySQL database engine updates 2017-12-11 \(p. 1225\)](#).
- Migrating data from MySQL using an Amazon S3 bucket. For more information, see [Migrating data from MySQL by using an Amazon S3 bucket \(p. 784\)](#).

MySQL 5.7 compatibility

This Aurora MySQL version is wire-compatible with MySQL 5.7 and includes features such as JSON support, spatial indexes, and generated columns. Aurora MySQL uses a native implementation of spatial indexing using z-order curves to deliver >20x better write performance and >10x better read performance than MySQL 5.7 for spatial datasets.

This Aurora MySQL version does not currently support the following MySQL 5.7 features:

- Group replication plugin
- Increased page size
- InnoDB buffer pool loading at startup
- InnoDB full-text parser plugin
- Multisource replication
- Online buffer pool resizing
- Password validation plugin

- Query rewrite plugins
- Replication filtering
- The `CREATE TABLESPACE` SQL statement

Aurora MySQL database engine updates 2021-05-25 (version 2.10.0)

Version: 2.10.0

Aurora MySQL 2.10.0 is generally available. Aurora MySQL 2.x versions are compatible with MySQL 5.7, and Aurora MySQL 1.x versions are compatible with MySQL 5.6.

Currently supported Aurora MySQL releases are 1.14.*, 1.15.*, 1.16.*, 1.17.*, 1.18.*, 1.19.*, 1.20.*, 1.21.*, 1.22.*, 1.23.*, 2.01.*, 2.02.*, 2.03.*, 2.04.*, 2.05.*, 2.06.*, 2.07.*, 2.08.*, 2.09.*, and 2.10.*.

You can upgrade an existing Aurora MySQL 2.* database cluster to Aurora MySQL 2.10.0. For clusters running Aurora MySQL version 1, you can upgrade an existing Aurora MySQL 1.23 or higher cluster directly to 2.10.0. You can also restore a snapshot from any currently supported Aurora MySQL release into Aurora MySQL 2.10.0.

To create a cluster with an older version of Aurora MySQL, specify the engine version through the AWS Management Console, the AWS CLI, or the Amazon RDS API.

If you have any questions or concerns, AWS Support is available on the community forums and through [AWS Premium Support](#). For more information, see [Maintaining an Amazon Aurora DB cluster \(p. 443\)](#).

Note

For information on how to upgrade your Aurora MySQL database cluster, see [Upgrading the minor version or patch level of an Aurora MySQL DB cluster \(p. 1088\)](#).

Improvements

Security fixes:

- [CVE-2021-23841](#)
- [CVE-2021-3449](#)
- [CVE-2020-28196](#)
- [CVE-2020-14790](#)
- [CVE-2020-14776](#)
- [CVE-2020-14567](#)
- [CVE-2020-14559](#)
- [CVE-2020-14553](#)
- [CVE-2020-14547](#)
- [CVE-2020-14540](#)
- [CVE-2020-14539](#)
- [CVE-2018-3251](#)
- [CVE-2018-3156](#)
- [CVE-2018-3143](#)

New features:

- The db.t3.large instance class is now supported for Aurora MySQL.
- *Binary log replication:*
 - Introduced the binlog I/O cache to improve binlog performance by reducing contention between writer threads and dump threads. For more information, see [Optimizing binary log replication \(p. 948\)](#).
 - In [Aurora MySQL version 2.08](#), we introduced improved binary log (binlog) processing to reduce crash recovery time and commit time latency when very large transactions are involved. These improvements are now supported for clusters that have GTID enabled.
- *Improved reader instance availability:*
 - Previously, when a writer instance restarted, all reader instances in an Aurora MySQL cluster restarted as well. With today's launch, in-Region reader instances continue to serve read requests during a writer instance restart, improving read availability in the cluster. For more information, see [Rebooting an Aurora MySQL cluster \(version 2.10 and higher\) \(p. 452\)](#).

Important

After you upgrade to Aurora MySQL 2.10, rebooting the writer instance doesn't perform a reboot of the entire cluster. If you want to reboot the entire cluster, now you reboot any reader instances in the cluster after rebooting the writer instance.

- Improved the performance of the read ahead page reads requested by logical read ahead (LRA) technique. This was done by batching the multiple page reads in a single request sent to Aurora storage. As a result, the queries that use the LRA optimization execute up to 3x faster.
- *Zero-downtime restarts and patching:*
 - Improved zero-downtime restart (ZDR) and zero-downtime patching (ZDP) to enable ZDR and ZDP in a wider range of scenarios, including the added support for cases when binary logging is enabled. Also, improved visibility into ZDR and ZDP events. See documentation for details: [Zero-downtime restart \(ZDR\) for Amazon Aurora MySQL \(p. 920\)](#) and [Using zero-downtime patching \(p. 1091\)](#).

Availability improvements:

- Improvements for faster startup when the database has a large number of temporary indexes and tables created during a prior interrupted DDL activity.
- Fixed multiple issues related to repeated restarts during the crash recovery of interrupted DDL operations, such as `DROP TRIGGER`, `ALTER TABLE`, and specifically `ALTER TABLE` that modifies the type of partitioning or number of partitions in a table.
- Fixed an issue that could cause a server restart during Database Activity Streams (DAS) log processing.
- Fixed an issue printing an error message while processing an `ALTER` query on system tables.

General improvements:

- Fixed an issue where the query cache could return stale results on a reader instance.
- Fixed an issue where some Aurora commit metrics were not being updated when the system variable `innodb_flush_log_at_trx_commit` was set to 0 or 2.
- Fixed an issue where a query result stored in the query cache was not refreshed by multistatement transactions.
- Fixed an issue that could cause the last-modified timestamp of binary log files to not be updated correctly. This could lead to binary log files being purged prematurely, before reaching the customer-configured retention period.
- Fixed incorrect reported binlog filename and position from InnoDB after crash recovery.
- Fixed an issue that could cause large transactions to generate incorrect binlog events if the `binlog_checksum` parameter was set to `NONE`.
- Fixed an issue that caused a binlog replica to stop with an error if the replicated transaction contained a DDL statement and a large number of row changes.

- Fixed an issue leading to a restart in a reader instance when dropping a table.
- Fixed an issue that caused open source connectors to fail when attempting to consume a binlog file with a large transaction.
- Fixed an issue that could lead to incorrect query results on the large geometry column after creating a spatial index on the table with the large geometry values.
- The database now recreates the temporary tablespace during restart, which allows the associated storage space to be freed and reclaimed.
- Fixed an issue that prevented `performance_schema` tables from being truncated on Aurora reader instances.
- Fixed an issue that caused a binlog replica to stop with an `HA_ERR_KEY_NOT_FOUND` error.
- Fixed an issue that caused the database to restart when running `FLUSH TABLES WITH READ LOCK` statement.
- Fixed an issue that prevented the use of user-level lock functions on Aurora reader instances.

Integration of MySQL community edition bug fixes

- Interleaved transactions could sometimes deadlock the replica applier when the transaction isolation level was set to `REPEATABLE READ`. (Bug #25040331)
- When a stored procedure contained a statement referring to a view which in turn referred to another view, the procedure could not be invoked successfully more than once. (Bug #87858, Bug #26864199)
- For queries with many `OR` conditions, the optimizer now is more memory-efficient and less likely to exceed the memory limit imposed by the `range_optimizer_max_mem_size` system variable. In addition, the default value for that variable has been raised from 1,536,000 to 8,388,608. (Bug #79450, Bug #22283790)
- *Replication:* In the `next_event()` function, which is called by a replica's SQL thread to read the next event from the relay log, the SQL thread did not release the `relaylog.log_lock` it acquired when it ran into an error (for example, due to a closed relay log), causing all other threads waiting to acquire a lock on the relay log to hang. With this fix, the lock is released before the SQL thread leaves the function under the situation. (Bug #21697821)
- Fixing a memory corruption for `ALTER TABLE` with virtual column. (Bug #24961167; Bug #24960450)
- *Replication:* Multithreaded replicas could not be configured with small queue sizes using `slave_pending_jobs_size_max` if they ever needed to process transactions larger than that size. Any packet larger than `slave_pending_jobs_size_max` was rejected with the error `ER_MTS_EVENT_BIGGER_PENDING_JOBS_SIZE_MAX`, even if the packet was smaller than the limit set by `slave_max_allowed_packet`. With this fix, `slave_pending_jobs_size_max` becomes a soft limit rather than a hard limit. If the size of a packet exceeds `slave_pending_jobs_size_max` but is less than `slave_max_allowed_packet`, the transaction is held until all the replica workers have empty queues, and then processed. All subsequent transactions are held until the large transaction has been completed. The queue size for replica workers can therefore be limited while still allowing occasional larger transactions. (Bug #21280753, Bug #77406)
- *Replication:* When using a multithreaded replica, applier errors displayed worker ID data that was inconsistent with data externalized in Performance Schema replication tables. (Bug #25231367)
- *Replication:* On a GTID-based replication replica running with `-gtid-mode=ON`, `-log-bin=OFF`, and using `-slave-skip-errors`, when an error was encountered that should be ignored `Exec_Master_Log_Pos` was not being correctly updated, causing `Exec_Master_Log_Pos` to loose synchrony with `Read_master_log_pos`. If a `GTID_NEXT` was not specified, the replica would never update its GTID state when rolling back from a single statement transaction. The `Exec_Master_Log_Pos` would not be updated because even though the transaction was finished, its GTID state would show otherwise. The fix removes the restraint of updating the GTID state when a transaction is rolled back only if `GTID_NEXT` is specified. (Bug #22268777)
- *Replication:* A partially failed statement was not correctly consuming an auto-generated or specified GTID when binary logging was disabled. The fix ensures that a partially failed `DROP TABLE`, a partially

failed `DROP USER`, or a partially failed `DROP VIEW` consume respectively the relevant GTID and save it into `@@GLOBAL.GTID_EXECUTED` and `mysql.gtid_executed` table when binary logging is disabled. (Bug #21686749)

- *Replication:* Replicas running MySQL 5.7 could not connect to a MySQL 5.5 source due to an error retrieving the `server_uuid`, which is not part of MySQL 5.5. This was caused by changes in the method of retrieving the `server_uuid`. (Bug #22748612)
- *Replication:* The GTID transaction skipping mechanism that silently skips a GTID transaction that was previously executed did not work properly for XA transactions. (Bug #25041920)
- `>XA ROLLBACK` statements that failed because an incorrect transaction ID was given, could be recorded in the binary log with the correct transaction ID, and could therefore be actioned by replication replicas. A check is now made for the error situation before binary logging takes place, and failed XA ROLLBACK statements are not logged. (Bug #26618925)
- *Replication:* If a replica was set up using a `CHANGE MASTER TO` statement that did not specify the source log file name and source log position, then shut down before `START SLAVE` was issued, then restarted with the option `-relay-log-recovery` set, replication did not start. This happened because the receiver thread had not been started before relay log recovery was attempted, so no log rotation event was available in the relay log to provide the source log file name and source log position. In this situation, the replica now skips relay log recovery and logs a warning, then proceeds to start replication. (Bug #28996606, Bug #93397)
- *Replication:* In row-based replication, a message that incorrectly displayed field lengths was returned when replicating from a table with a `utf8mb3` column to a table of the same definition where the column was defined with a `utf8mb4` character set. (Bug #25135304, Bug #83918)
- *Replication:* When a `RESET SLAVE` statement was issued on a replication replica with GTIDs in use, the existing relay log files were purged, but the replacement new relay log file was generated before the set of received GTIDs for the channel had been cleared. The former GTID set was therefore written to the new relay log file as the `PREVIOUS_GTIDS` event, causing a fatal error in replication stating that the replica had more GTIDs than the source, even though the `gtid_executed` set for both servers was empty. Now, when `RESET SLAVE` is issued, the set of received GTIDs is cleared before the new relay log file is generated, so that this situation does not occur. (Bug #27411175)
- *Replication:* With GTIDs in use for replication, transactions including statements that caused a parsing error (`ER_PARSE_ERROR`) could not be skipped manually by the recommended method of injecting an empty or replacement transaction with the same GTID. This action should result in the replica identifying the GTID as already used, and therefore skipping the unwanted transaction that shared its GTID. However, in the case of a parsing error, because the statement was parsed before the GTID was checked to see if it needed to be skipped, the replication applier thread stopped due to the parsing error, even though the intention was for the transaction to be skipped anyway. With this fix, the replication applier thread now ignores parsing errors if the transaction concerned needs to be skipped because the GTID was already used. Note that this behavior change does not apply in the case of workloads consisting of binary log output produced by `mysqlbinlog`. In that situation, there would be a risk that a transaction with a parsing error that immediately follows a skipped transaction would also be silently skipped, when it ought to raise an error. (Bug #27638268)
- *Replication:* Enable the SQL thread to GTID skip a partial transaction. (Bug #25800025)
- *Replication:* When a negative or fractional timeout parameter was supplied to `WAIT_UNTIL_SQL_THREAD_AFTER_GTIDS()`, the server behaved in unexpected ways. With this fix:
 - A fractional timeout value is read as-is, with no round-off.
 - A negative timeout value is rejected with an error if the server is on a strict SQL mode; if the server is not on a strict SQL mode, the value makes the function return `NULL` immediately without any waiting and then issue a warning. (Bug #24976304, Bug #83537)
- *Replication:* If the `WAIT_FOR_EXECUTED_GTID_SET()` function was used with a timeout value including a fractional part (for example, 1.5), an error in the casting logic meant that the timeout was rounded down to the nearest whole second, and to zero for values less than 1 second (for example, 0.1). The casting logic has now been corrected so that the timeout value is applied as originally specified with no rounding. Thanks to Dirkjan Bussink for the contribution. (Bug #29324564, Bug #94247)

- With GTIDs enabled, [XA COMMIT](#) on a disconnected XA transaction within a multiple-statement transaction raised an assertion. (Bug #22173903)
- Replication:* An assertion was raised in debug builds if an [XA ROLLBACK](#) statement was issued for an unknown transaction identifier when the [gtid_next](#) value had been set manually. The server now does not attempt to update the GTID state if an XA ROLLBACK statement fails with an error. (Bug #27928837, Bug #90640)
- Fix wrong sorting order issue when multiple `CASE` functions are used in `ORDER BY` clause (Bug#22810883).

Comparison with Aurora MySQL version 1

The following Amazon Aurora MySQL features are supported in Aurora MySQL version 1 (compatible with MySQL 5.6), but these features are currently not supported in Aurora MySQL version 2 (compatible with MySQL 5.7).

- Asynchronous key prefetch (AKP). For more information, see [Optimizing Amazon Aurora indexed join queries with asynchronous key prefetch \(p. 1036\)](#).
- Hash joins. For more information, see [Optimizing large Aurora MySQL join queries with hash joins \(p. 1038\)](#).
- Native functions for synchronously invoking AWS Lambda functions. For more information, see [Invoking a Lambda function with an Aurora MySQL native function \(p. 1012\)](#).
- Scan batching. For more information, see [Aurora MySQL database engine updates 2017-12-11 \(p. 1225\)](#).
- Migrating data from MySQL using an Amazon S3 bucket. For more information, see [Migrating data from MySQL by using an Amazon S3 bucket \(p. 784\)](#).

MySQL 5.7 compatibility

This Aurora MySQL version is wire-compatible with MySQL 5.7 and includes features such as JSON support, spatial indexes, and generated columns. Aurora MySQL uses a native implementation of spatial indexing using z-order curves to deliver >20x better write performance and >10x better read performance than MySQL 5.7 for spatial datasets.

This Aurora MySQL version does not currently support the following MySQL 5.7 features:

- Group replication plugin
- Increased page size
- InnoDB buffer pool loading at startup
- InnoDB full-text parser plugin
- Multisource replication
- Online buffer pool resizing
- Password validation plugin
- Query rewrite plugins
- Replication filtering
- The `CREATE TABLESPACE` SQL statement

Aurora MySQL database engine updates 2021-11-12 (version 2.09.3)

Version: 2.09.3

Aurora MySQL 2.09.3 is generally available. Aurora MySQL 2.x versions are compatible with MySQL 5.7, and Aurora MySQL 1.x versions are compatible with MySQL 5.6.

To create a cluster with an older version of Aurora MySQL, specify the engine version through the AWS Management Console, the AWS CLI, or the Amazon RDS API.

If you have any questions or concerns, AWS Support is available on the community forums and through [AWS Premium Support](#). For more information, see [Maintaining an Amazon Aurora DB cluster \(p. 443\)](#).

Note

For information on how to upgrade your Aurora MySQL database cluster, see [Upgrading the minor version or patch level of an Aurora MySQL DB cluster \(p. 1088\)](#).

Improvements

Security fixes:

Fixes and other enhancements to fine-tune handling in a managed environment. Additional CVE fixes below:

- [CVE-2021-23841](#)
- [CVE-2021-3712](#)
- [CVE-2021-3449](#)
- [CVE-2021-2307](#)
- [CVE-2021-2226](#)
- [CVE-2021-2174](#)
- [CVE-2021-2171](#)
- [CVE-2021-2169](#)
- [CVE-2021-2166](#)
- [CVE-2021-2154](#)
- [CVE-2021-2060](#)
- [CVE-2021-2032](#)
- [CVE-2021-2001](#)
- [CVE-2020-28196](#)
- [CVE-2020-14769](#)
- [CVE-2019-17543](#)
- [CVE-2019-2960](#)

Availability improvements:

- Introduced an optimization which can reduce contention for queries that are executed on tables in `information_schema`.
- Add support for ECDHE SSL ciphers.

General improvements:

- Fixed an issue which, in rare conditions, can cause a writer instance to restart when an in-memory data-integrity check fails.
- Fixed an issue which, in rare conditions, can cause the database instance to restart when the cluster volume is expanding while binary logging is enabled.
- Fixed a rare race condition during a database instance restart, which can cause more than one restart.

- Fixed an issue which can cause a database instance restart to fail when the database has a large number of user and privilege combinations.
- Fixed an issue with parallel query which can cause the database to restart when executing SQL statements with LIMIT clause.
- Fixed an issue with incorrect reporting of aurora replication lag.
- Fixed an issue which can cause general_log and slow_log tables to become inaccessible after in-place major version upgrade from Aurora-MySQL 1.x (based on MySQL 5.6) to Aurora-MySQL 2.x (based on MySQL 5.7).
- Fixed an issue which, in rare cases, can cause the database instance to restart when innodb_trx, innodb_locks or innodb_lockwaits tables are queried while the database is under heavy workload. Monitoring tools and features such as performance insights may query such tables.
- Fixed an issue which can cause a database instance to restart when "FLUSH TABLES WITH READ LOCK" SQL statement is executed.
- Fixed an issue where the InnoDB purge process pauses during the deletion of a reader instance leading to a temporary increase in history list length.
- Fixed an issue with parallel query which can cause the database to restart when executing a SQL statement against a table containing a virtual column.
- Fixed an issue with parallel query which can cause the database to return incorrect groupings or sort order when executing queries with GROUP BY clause and a WHERE clause containing a range predicate.
- Fixed an issue in parallel query which, in rare conditions, can cause the database to restart when executing SQL statements with JSON functions.
- Fixed an issue which, in rare conditions, can cause the writer instance in primary Global Database cluster to restart because of a race condition during Global Database Replication.
- Fixed an issue that can cause a Binlog replica to stop with an HA_ERR_FOUND_DUPP_KEY error when replicating certain DDL and DCL statements. The issue occurs when the source instance is configured with MIXED binary logging format and READ COMMITTED or READ UNCOMMITTED isolation level.
- Fixed an issue which, in rare conditions, can cause the database instance to restart when using XA transactions in READ COMMITTED isolation level.
- Fixed an issue where the value of a TIMESTAMP column of an existing row is updated to the latest timestamp when all of the following conditions are satisfied: 1. a trigger exists for the table; 2. an INSERT is performed on the table that has an ON DUPLICATE KEY UPDATE clause; 3. the inserted row can cause a duplicate value violation in a UNIQUE index or PRIMARY KEY; and 4. one or more columns are of TIMESTAMP data type and have a default value of CURRENT_TIMESTAMP.
- Fixed an issue which, in rare conditions, can cause a reader instance to restart due to an incorrect check processing.
- Fixed an issue which can cause the reader instance to restart when the writer instance grows the database volume to cross specific volume size boundaries.
- Fixed an issue which can cause longer restart times for database instances using cloned cluster volumes.
- Fixed an issue where a database instance restart may fail one or more times after a TRUNCATE TABLE operation was performed on the writer instance.
- Fixed an issue which, in rare conditions, can cause the database instance to restart.
- Fixed an issue which, in rare conditions, can cause the writer instance to restart when the database volume grows to a multiple of 160GB.

Integration of MySQL community edition bug fixes

- Bug #23533396 - When adding a new index, the server dropped an internally defined foreign key index and attempted to use a secondary index defined on a virtual generated column as the foreign

key index, causing a server exit. InnoDB now permits a foreign key constraint to reference a secondary index defined on a virtual generated column.

- Bug #29550513 - Replication: A locking issue in the WAIT_FOR_EXECUTED_GTID_SET() function can cause the server to hang in certain circumstances. The issue has now been corrected.

Comparison with Aurora MySQL version 1

The following Amazon Aurora MySQL features are supported in Aurora MySQL version 1 (compatible with MySQL 5.6), but these features are currently not supported in Aurora MySQL version 2 (compatible with MySQL 5.7).

- Asynchronous key prefetch (AKP). For more information, see [Optimizing Amazon Aurora indexed join queries with asynchronous key prefetch \(p. 1036\)](#).
- Hash joins. For more information, see [Optimizing large Aurora MySQL join queries with hash joins \(p. 1038\)](#).
- Native functions for synchronously invoking AWS Lambda functions. For more information, see [Invoking a Lambda function with an Aurora MySQL native function \(p. 1012\)](#).
- Scan batching. For more information, see [Aurora MySQL database engine updates 2017-12-11 \(p. 1225\)](#).
- Migrating data from MySQL using an Amazon S3 bucket. For more information, see [Migrating data from MySQL by using an Amazon S3 bucket \(p. 784\)](#).

MySQL 5.7 compatibility

This Aurora MySQL version is wire-compatible with MySQL 5.7 and includes features such as JSON support, spatial indexes, and generated columns. Aurora MySQL uses a native implementation of spatial indexing using z-order curves to deliver >20x better write performance and >10x better read performance than MySQL 5.7 for spatial datasets.

This Aurora MySQL version does not currently support the following MySQL 5.7 features:

- Group replication plugin
- Increased page size
- InnoDB buffer pool loading at startup
- InnoDB full-text parser plugin
- Multisource replication
- Online buffer pool resizing
- Password validation plugin
- Query rewrite plugins
- Replication filtering
- The CREATE TABLESPACE SQL statement

Aurora MySQL database engine updates 2021-02-26 (version 2.09.2)

Version: 2.09.2

Aurora MySQL 2.09.2 is generally available. Aurora MySQL 2.x versions are compatible with MySQL 5.7 and Aurora MySQL 1.x versions are compatible with MySQL 5.6.

Currently supported Aurora MySQL releases are 1.14.* , 1.15.* , 1.16.* , 1.17.* , 1.18.* , 1.19.* , 1.20.* , 1.21.* , 1.22.* , 1.23.* , 2.01.* , 2.02.* , 2.03.* , 2.04.* , 2.05.* , 2.06.* , 2.07.* , 2.08.* , and 2.09.*.

You can upgrade an existing Aurora MySQL 2.* database cluster to Aurora MySQL 2.09.2. For clusters running Aurora MySQL version 1, you can upgrade an existing Aurora MySQL 1.23 or higher cluster directly to 2.09.2. You can also restore a snapshot from any currently supported Aurora MySQL release into Aurora MySQL 2.09.2.

To create a cluster with an older version of Aurora MySQL, specify the engine version through the AWS Management Console, the AWS CLI, or the Amazon RDS API.

If you have any questions or concerns, AWS Support is available on the community forums and through [AWS Premium Support](#). For more information, see [Maintaining an Amazon Aurora DB cluster \(p. 443\)](#).

Note

For information on how to upgrade your Aurora MySQL database cluster, see [Upgrading the minor version or patch level of an Aurora MySQL DB cluster \(p. 1088\)](#).

Improvements

New features:

- Aurora MySQL clusters now support the following EC2 R6g instances powered by Arm-based AWS Graviton2 processors: `r6g.large`, `r6g.xlarge`, `r6g.2xlarge`, `r6g.4xlarge`, `r6g.8xlarge`, `r6g.12xlarge`, `r6g.16xlarge`. For more information, see [Aurora DB instance classes \(p. 54\)](#).

Security fixes:

Fixes and other enhancements to fine-tune handling in a managed environment. Additional CVE fixes below:

- [CVE-2020-14775](#)
- [CVE-2020-14793](#)
- [CVE-2020-14765](#)
- [CVE-2020-14769](#)
- [CVE-2020-14812](#)
- [CVE-2020-14760](#)
- [CVE-2020-14672](#)
- [CVE-2020-14790](#)
- [CVE-2020-1971](#)

Availability improvements:

- Fixed an issue introduced in 2.09.0 that can cause elevated write latency during the scaling of the cluster storage volume.
- Fixed an issue in the dynamic resizing feature that could cause Aurora Read Replicas to restart.
- Fixed an issue that could cause longer downtime during upgrade from 1.23.* to 2.09.*.
- Fixed an issue where a DDL or DML could cause engine restart during a page prefetch request.
- Fixed an issue that caused a binlog replica to stop with an error if the replicated transaction contains a DDL statement and a large number of row changes.
- Fixed an issue where a database acting as a binlog replica could restart while replicating a DDL event on the `mysql.time_zone` table.

- Fixed an issue that could cause large transactions to generate incorrect binlog events if the `binlog_checksum` parameter was set to `NONE`.
- Fixed an issue that caused a binlog replica to stop with an `HA_ERR_KEY_NOT_FOUND` error.
- Improved overall stability.

Comparison with Aurora MySQL version 1

The following Amazon Aurora MySQL features are supported in Aurora MySQL Version 1 (compatible with MySQL 5.6), but these features are currently not supported in Aurora MySQL Version 2 (compatible with MySQL 5.7).

- Asynchronous key prefetch (AKP). For more information, see [Optimizing Amazon Aurora indexed join queries with asynchronous key prefetch \(p. 1036\)](#).
- Hash joins. For more information, see [Optimizing large Aurora MySQL join queries with hash joins \(p. 1038\)](#).
- Native functions for synchronously invoking AWS Lambda functions. For more information, see [Invoking a Lambda function with an Aurora MySQL native function \(p. 1012\)](#).
- Scan batching. For more information, see [Aurora MySQL database engine updates 2017-12-11 \(p. 1225\)](#).
- Migrating data from MySQL using an Amazon S3 bucket. For more information, see [Migrating data from MySQL by using an Amazon S3 bucket \(p. 784\)](#).

MySQL 5.7 compatibility

This Aurora MySQL version is wire-compatible with MySQL 5.7 and includes features such as JSON support, spatial indexes, and generated columns. Aurora MySQL uses a native implementation of spatial indexing using z-order curves to deliver >20x better write performance and >10x better read performance than MySQL 5.7 for spatial datasets.

This Aurora MySQL version does not currently support the following MySQL 5.7 features:

- Group replication plugin
- Increased page size
- InnoDB buffer pool loading at startup
- InnoDB full-text parser plugin
- Multisource replication
- Online buffer pool resizing
- Password validation plugin
- Query rewrite plugins
- Replication filtering
- The `CREATE TABLESPACE` SQL statement

Aurora MySQL database engine updates 2020-12-11 (version 2.09.1)

Version: 2.09.1

Aurora MySQL 2.09.1 is generally available. Aurora MySQL 2.x versions are compatible with MySQL 5.7 and Aurora MySQL 1.x versions are compatible with MySQL 5.6.

Currently supported Aurora MySQL releases are 1.14.* , 1.15.* , 1.16.* , 1.17.* , 1.18.* , 1.19.* , 1.20.* , 1.21.* , 1.22.* , 1.23.* , 2.01.* , 2.02.* , 2.03.* , 2.04.* , 2.05.* , 2.06.* , 2.07.* , 2.08.* , and 2.09.*.

You can upgrade an existing Aurora MySQL 2.* database cluster to Aurora MySQL 2.09.1. For clusters running Aurora MySQL version 1, you can upgrade an existing Aurora MySQL 1.23 or higher cluster directly to 2.09.1. You can also restore a snapshot from any currently supported Aurora MySQL release into Aurora MySQL 2.09.1.

To create a cluster with an older version of Aurora MySQL, specify the engine version through the AWS Management Console, the AWS CLI, or the RDS API.

If you have any questions or concerns, AWS Support is available on the community forums and through [AWS Premium Support](#). For more information, see [Maintaining an Amazon Aurora DB cluster \(p. 443\)](#).

Note

For information on how to upgrade your Aurora MySQL database cluster, see [Upgrading the minor version or patch level of an Aurora MySQL DB cluster \(p. 1088\)](#).

Improvements

Security fixes:

Fixes and other enhancements to fine-tune handling in a managed environment. Additional CVE fixes below:

- [CVE-2020-14567](#)
- [CVE-2020-14559](#)
- [CVE-2020-14553](#)
- [CVE-2020-14547](#)
- [CVE-2020-14540](#)
- [CVE-2020-2812](#)
- [CVE-2020-2806](#)
- [CVE-2020-2780](#)
- [CVE-2020-2765](#)
- [CVE-2020-2763](#)
- [CVE-2020-2760](#)
- [CVE-2020-2579](#)

Incompatible changes:

This version introduces a permission change that affects the behavior of the `mysqldump` command. Users must have the `PROCESS` privilege to access the `INFORMATION_SCHEMA.FILES` table. To run the `mysqldump` command without any changes, grant the `PROCESS` privilege to the database user that the `mysqldump` command connects to. You can also run the `mysqldump` command with the `--no-tablespaces` option. With that option, the `mysqldump` output doesn't include any `CREATE LOGFILE GROUP` or `CREATE TABLESPACE` statements. In that case, the `mysqldump` command doesn't access the `INFORMATION_SCHEMA.FILES` table, and you don't need to grant the `PROCESS` permission.

Availability improvements:

- Fixed an issue that might cause a client session to hang when the database engine encounters an error while reading from or writing to the network.
- Fixed a memory leak in dynamic resizing feature, introduced in 2.09.0.

Global databases:

- Fixed multiple issues where a global database secondary Region's replicas might restart when upgraded to release 2.09.0 while the primary Region writer was on an older release version.

Integration of MySQL community edition bug fixes

- **Replication:** Interleaved transactions could sometimes deadlock the slave applier when the transaction isolation level was set to `REPEATABLE READ`. (Bug #25040331)
- For a table having a `TIMESTAMP` or `DATETIME` column having a default of `CURRENT_TIMESTAMP`, the column could be initialized to `0000-00-00 00:00:00` if the table had a `BEFORE INSERT` trigger. (Bug #25209512, Bug #84077)
- For an `INSERT` statement for which the `VALUES` list produced values for the second or later row using a subquery containing a join, the server could exit after failing to resolve the required privileges. (Bug #23762382)

Comparison with Aurora MySQL version 1

The following Amazon Aurora MySQL features are supported in Aurora MySQL Version 1 (compatible with MySQL 5.6), but these features are currently not supported in Aurora MySQL Version 2 (compatible with MySQL 5.7).

- Asynchronous key prefetch (AKP). For more information, see [Optimizing Amazon Aurora indexed join queries with asynchronous key prefetch \(p. 1036\)](#).
- Hash joins. For more information, see [Optimizing large Aurora MySQL join queries with hash joins \(p. 1038\)](#).
- Native functions for synchronously invoking AWS Lambda functions. For more information, see [Invoking a Lambda function with an Aurora MySQL native function \(p. 1012\)](#).
- Scan batching. For more information, see [Aurora MySQL database engine updates 2017-12-11 \(p. 1225\)](#).
- Migrating data from MySQL using an Amazon S3 bucket. For more information, see [Migrating data from MySQL by using an Amazon S3 bucket \(p. 784\)](#).

MySQL 5.7 compatibility

This Aurora MySQL version is wire-compatible with MySQL 5.7 and includes features such as JSON support, spatial indexes, and generated columns. Aurora MySQL uses a native implementation of spatial indexing using z-order curves to deliver >20x better write performance and >10x better read performance than MySQL 5.7 for spatial datasets.

This Aurora MySQL version does not currently support the following MySQL 5.7 features:

- Group replication plugin
- Increased page size
- InnoDB buffer pool loading at startup
- InnoDB full-text parser plugin
- Multisource replication
- Online buffer pool resizing
- Password validation plugin
- Query rewrite plugins
- Replication filtering

- The `CREATE TABLESPACE` SQL statement

Aurora MySQL database engine updates 2020-09-17 (version 2.09.0)

Version: 2.09.0

Aurora MySQL 2.09.0 is generally available. Aurora MySQL 2.x versions are compatible with MySQL 5.7 and Aurora MySQL 1.x versions are compatible with MySQL 5.6.

Currently supported Aurora MySQL releases are 1.14.* , 1.15.* , 1.16.* , 1.17.* , 1.18.* , 1.19.* , 1.20.* , 1.21.* , 1.22.* , 1.23.* , 2.01.* , 2.02.* , 2.03.* , 2.04.* , 2.05.* , 2.06.* , 2.07.* , 2.08.* , and 2.09.*.

You can restore a snapshot from Aurora MySQL 1.23.* into Aurora MySQL 2.09.0. You also have the option to upgrade existing Aurora MySQL 2.* database clusters to Aurora MySQL 2.09.0. You can't upgrade an existing Aurora MySQL 1.23.* cluster directly to 2.09.0; however, you can restore its snapshot to Aurora MySQL 2.09.0.

Important

The improvements to Aurora storage in this version limit the available upgrade paths from Aurora MySQL 1.* to Aurora MySQL 2.09. When you upgrade an Aurora MySQL 1.* cluster to 2.09, you must upgrade from Aurora MySQL 1.23.

To create a cluster with an older version of Aurora MySQL, specify the engine version through the AWS Management Console, the AWS CLI, or the RDS API.

If you have any questions or concerns, AWS Support is available on the community forums and through [AWS Premium Support](#). For more information, see [Maintaining an Amazon Aurora DB cluster \(p. 443\)](#).

Note

For information on how to upgrade your Aurora MySQL database cluster, see [Upgrading the minor version or patch level of an Aurora MySQL DB cluster \(p. 1088\)](#).

Improvements

New features:

- With this release, you can create Amazon Aurora MySQL database instances with up to 128 tebibytes (TiB) of storage. The new storage limit is an increase from the prior 64 TiB. The 128 TiB storage size supports larger databases. This capability is not supported on small instances sizes (db.t2 or db.t3). A single tablespace cannot grow beyond 64 TiB due to [InnoDB limitations with 16 KB page size](#).

Aurora alerts you when the cluster volume size is near 128 TiB, so that you can take action prior to hitting the size limit. The alerts appear in the mysql log and RDS Events in the AWS Management Console.

- You can now turn parallel query on or off for an existing cluster by changing the value of the DB cluster parameter `aurora_parallel_query`. You don't need to use the `parallelquery` setting for the `--engine-mode` parameter when creating the cluster.

Parallel query is now expanded to be available in all regions where Aurora MySQL is available.

There are a number of other functionality enhancements and changes to the procedures for upgrading and enabling parallel query in an Aurora cluster. For more information, see [Working with parallel query for Amazon Aurora MySQL \(p. 881\)](#).

- Aurora dynamically resizes your cluster storage space. With dynamic resizing, the storage space for your Aurora DB cluster automatically decreases when you remove data from the DB cluster. For more information, see [Storage scaling \(p. 396\)](#).

Note

The dynamic resizing feature is being deployed in phases to the AWS Regions where Aurora is available. Depending on the Region where your cluster is, this feature might not be available yet. For more information, see [the What's New announcement](#).

High priority fixes:

- Backport of Community Bug #27659490: SELECT USING DYNAMIC RANGE AND INDEX MERGE USE TOO MUCH MEMORY (OOM)
- Bug #26881508: MYSQL #1: DISABLE_ABORT_ON_ERROR IN AUTH_COMMON.H
- Backport of Community Bug #24437124: POSSIBLE BUFFER OVERFLOW ON CREATE TABLE
- Backport of Bug #27158030: INNODB ONLINE ALTER CRASHES WITH CONCURRENT DML
- Bug #29770705: SERVER CRASHED WHILE EXECUTING SELECT WITH SPECIFIC WHERE CLAUSE
- Backport of BUG #26502135: MYSQLD SEGFAULTS IN MDL_CONTEXT::TRY_ACQUIRE_LOCK_IMPL
- Backport of Bug #26935001: ALTER TABLE AUTO_INCREMENT TRIES TO READ INDEX FROM DISCARDED TABLESPACE
- Bug #28491099: [FATAL] MEMORY BLOCK IS INVALID | INNODB: ASSERTION FAILURE: UTOUT.CC:670
- Bug #30499288: GCC 9.2.1 REPORTS A NEW WARNING FOR OS_FILE_GET_PARENT_DIR
- Bug #29952565 where MYSQLD GOT SIGNAL 11 WHILE EXECUTING A QUERY(UNION + ORDER BY + SUB-QUERY)
- Bug #30628268: OUT OF MEMORY CRASH
- Bug #30441969: BUG #29723340: MYSQL SERVER CRASH AFTER SQL QUERY WITH DATA ?AST
- Bug #30569003: 5.7 REPLICATION BREAKAGE WITH SYNTAX ERROR WITH GRANT MANAGEMENT
- Bug #29915479: RUNNING COM_REGISTER_SLAVE WITHOUT COM_BINLOG_DUMP CAN RESULTS IN SERVER EXIT
- Bug #30569003: 5.7 REPLICATION BREAKAGE WITH SYNTAX ERROR WITH GRANT MANAGEMENT
- Bug #29915479: RUNNING COM_REGISTER_SLAVE WITHOUT COM_BINLOG_DUMP CAN RESULTS IN SERVER EXIT
- Bug #20712046: SHOW PROCESSLIST AND PERFORMANCE_SCHEMA TABLES DO NOT MASK PASSWORD FROM QUERY
- Backport bug #18898433: EXTREMELY SLOW PERFORMANCE WITH OUTER JOINS AND JOIN BUFFER (fixed in 5.7.21). Queries with many left joins were slow if join buffering was used (for example, using the block nested loop algorithm). (Bug #18898433, Bug #72854)"
- Backport bug #26402045: MYSQLD CRASHES ON QUERY (fixed in MySQL 5.7.23). Certain cases of subquery materialization could cause a server exit. These queries now produce an error suggesting that materialization be disabled. (Bug #26402045)
- [Backport from MySQL] users other than rdsadmin is disallowed to update pfs table in the reader replica.
- Fix the issue where the customer can not update the perfschema in the reader replica
- Bug #26666274: INFINITE LOOP IN PERFORMANCE SCHEMA BUFFER CONTAINER
- [Bug #26997096](#): relay_log_space value is not updated in a synchronized manner so that its value sometimes much higher than the actual disk space used by relay logs.
- [BUG #25082593](#): FOREIGN KEY VALIDATION DOESN'T NEED TO ACQUIRE GAP LOCK IN READ COMMITTED
- [CVE-2019-2731](#)
- [CVE-2018-2645](#)
- [CVE-2019-2581](#)

- [CVE-2018-2787](#)
- [CVE-2019-2482](#)
- [CVE-2018-2640](#)
- [CVE-2018-2784](#)
- [CVE-2019-2628](#)
- [CVE-2019-2911](#)
- [CVE-2019-2628](#)
- [CVE-2018-3284](#)
- [CVE-2018-3065](#)
- [CVE-2019-2537](#)
- [CVE-2019-2948](#)
- [CVE-2019-2434](#)
- [CVE-2019-2420](#)

Availability improvements:

- Enable lock manager ABA fix by default.
- Fixed an issue in the lock manager where a race condition can cause a lock to be shared by two transactions, causing the database to restart.
- Fixed an issue when creating a temporary table with compressed row format might result in a restart.
- Fix default value of `table_open_cache` on 16XL and 24XL instances which could cause repeated failovers and high CPU utilization on large instances classes (R4/R5-16XL, R5-12XL, R5-24XL). This impacted 2.07.x.
- Fixed an issue where restoring a cluster from Amazon S3 to Aurora MySQL version 2.08.0 takes longer than expected when the S3 backup didn't include the `mysql.host` table.
- Fixed an issue which might cause repeated failovers due to updates of virtual columns with secondary indexes.
- Fixed an issue related to transaction lock memory management with long-running write transactions resulting in a database restart.
- Fixed multiple issues where the engine might crash during zero-downtime patching while checking for safe point for patching.
- Fixed an issue to skip redo logging for temporary tables, which was previously causing a crash.
- Fixed a race condition in the lock manager between killing connection/query and the session killed.
- Fixed an issue where the database could crash if it is a binlog replica and receives a DDL event over the MySQL `time_zone` table.

Global databases:

- MySQL `INFORMATION_SCHEMA.REPLICA_HOST_STATUS` view in a secondary Region now shows the entries for the replicas belonging to that Region.
- Fixed unexpected query failures that could occur in a Global DB secondary Region after temporary network connectivity issues between the primary and secondary Regions.
-

Parallel query:

- Fixed the `EXPLAIN` plan for a Parallel Query query, which is incorrect for a simple single-table query.

- Fixed self-deadlock that may occur when Parallel Query is enabled.

General improvements:

- Export to S3 now supports the ENCRYPTION keyword.
- The `aurora_binlog_replication_max_yield_seconds` parameter now has a max value of 36,000. The previous maximum accepted value was 45. This parameter works only when the parameter `aurora_binlog_use_large_read_buffer` is set to 1.
- Changed the behavior to map MIXED binlog_format to ROW instead of STATEMENT when executing `LOAD DATA FROM INFILE | S3`.
- Fixed an issue where a binlog replica connected to an Aurora MySQL binlog primary might show incomplete data when the primary executed `LOAD DATA FROM S3` and `binlog_format` is set to STATEMENT.
- Increased maximum allowable length for audit system variables `server_audit_incl_users` and `server_audit_excl_users` from 1024 bytes to 2000 bytes.
- Fixed an issue where users may lose access to the database when lowering the `max_connections` parameter in the parameter group when the current connections is greater than the value being set.
- Fixed an issue in Data Activity Streams where a single quote and backslash were not escaped properly.

Integration of MySQL community edition bug fixes

- Bug #27659490: SELECT USING DYNAMIC RANGE AND INDEX MERGE USE TOO MUCH MEMORY(OOM)
- Bug #26881508: MYSQL #1: DISABLE_ABORT_ON_ERROR IN AUTH_COMMON.H
- Bug #24437124: POSSIBLE BUFFER OVERFLOW ON CREATE TABLE
- Bug #27158030: INNODB ONLINE ALTER CRASHES WITH CONCURRENT DML
- Bug #29770705: SERVER CRASHED WHILE EXECUTING SELECT WITH SPECIFIC WHERE CLAUSE
- Bug #26502135: MYSQLD SEGFAULTS IN MDL_CONTEXT::TRY_ACQUIRE_LOCK_IMPL
- Bug #26935001: ALTER TABLE AUTO_INCREMENT TRIES TO READ INDEX FROM DISCARDED TABLESPACE
- Bug #28491099: [FATAL] MEMORY BLOCK IS INVALID | INNODB: ASSERTION FAILURE: UTOUT.CC:670
- Bug #30499288: GCC 9.2.1 REPORTS A NEW WARNING FOR OS_FILE_GET_PARENT_DIR
- Bug #29952565: where MYSQLD GOT SIGNAL 11 WHILE EXECUTING A QUERY(UNION + ORDER BY + SUB-QUERY)
- Bug #30628268: OUT OF MEMORY CRASH
- Bug #30441969: BUG #29723340: MYSQL SERVER CRASH AFTER SQL QUERY WITH DATA ?AST
- Bug #30569003: 5.7 REPLICATION BREAKAGE WITH SYNTAX ERROR WITH GRANT MANAGEMENT
- Bug #29915479: RUNNING COM_REGISTER_SLAVE WITHOUT COM_BINLOG_DUMP CAN RESULTS IN SERVER EXIT
- Bug #30569003: 5.7 REPLICATION BREAKAGE WITH SYNTAX ERROR WITH GRANT MANAGEMENT
- Bug #29915479: RUNNING COM_REGISTER_SLAVE WITHOUT COM_BINLOG_DUMP CAN RESULTS IN SERVER EXIT
- Bug #20712046: SHOW PROCESSLIST AND PERFORMANCE_SCHEMA TABLES DO NOT MASK PASSWORD FROM QUERY
- Bug #18898433: EXTREMELY SLOW PERFORMANCE WITH OUTER JOINS AND JOIN BUFFER (fixed in 5.7.21)
- Bug #26402045: MYSQLD CRASHES ON QUERY (fixed in MySQL 5.7.23)
- Bug #23103937: PS_TRUNCATE_ALL_TABLES() DOES NOT WORK IN SUPER_READ_ONLY MODE
- Bug #26666274: INFINITE LOOP IN PERFORMANCE SCHEMA BUFFER CONTAINER

- Bug #26997096: relay_log_space value is not updated in a synchronized manner so that its value sometimes much higher than the actual disk space used by relay logs. (<https://github.com/mysql/mysql-server/commit/78f25d2809ad457e81f90342239c9bc32a36cdaf>)
- Bug #25082593: FOREIGN KEY VALIDATION DOESN'T NEED TO ACQUIRE GAP LOCK IN READ COMMITTED
- Bug #24764800: REPLICATION FAILING ON SLAVE WITH XAER_RMFAIL ERROR.
- Bug #81441: WARNING ABOUT LOCALHOST WHEN USING SKIP-NAME-RESOLVE.

Comparison with Aurora MySQL version 1

The following Amazon Aurora MySQL features are supported in Aurora MySQL Version 1 (compatible with MySQL 5.6), but these features are currently not supported in Aurora MySQL Version 2 (compatible with MySQL 5.7).

- Asynchronous key prefetch (AKP). For more information, see [Optimizing Amazon Aurora indexed join queries with asynchronous key prefetch \(p. 1036\)](#).
- Hash joins. For more information, see [Optimizing large Aurora MySQL join queries with hash joins \(p. 1038\)](#).
- Native functions for synchronously invoking AWS Lambda functions. For more information, see [Invoking a Lambda function with an Aurora MySQL native function \(p. 1012\)](#).
- Scan batching. For more information, see [Aurora MySQL database engine updates 2017-12-11 \(p. 1225\)](#).
- Migrating data from MySQL using an Amazon S3 bucket. For more information, see [Migrating data from MySQL by using an Amazon S3 bucket \(p. 784\)](#).

MySQL 5.7 compatibility

This Aurora MySQL version is wire-compatible with MySQL 5.7 and includes features such as JSON support, spatial indexes, and generated columns. Aurora MySQL uses a native implementation of spatial indexing using z-order curves to deliver >20x better write performance and >10x better read performance than MySQL 5.7 for spatial datasets.

This Aurora MySQL version does not currently support the following MySQL 5.7 features:

- Group replication plugin
- Increased page size
- InnoDB buffer pool loading at startup
- InnoDB full-text parser plugin
- Multisource replication
- Online buffer pool resizing
- Password validation plugin
- Query rewrite plugins
- Replication filtering
- The CREATE TABLESPACE SQL statement

Aurora MySQL database engine updates 2022-01-06 (version 2.08.4)

Version: 2.08.4

Aurora MySQL 2.08.4 is generally available. Aurora MySQL 2.x versions are compatible with MySQL 5.7 and Aurora MySQL 1.x versions are compatible with MySQL 5.6.

If you have any questions or concerns, AWS Support is available on the community forums and through [AWS Premium Support](#). For more information, see [Maintaining an Amazon Aurora DB cluster \(p. 443\)](#).

Note

For information on how to upgrade your Aurora MySQL database cluster, see [Upgrading the minor version or patch level of an Aurora MySQL DB cluster \(p. 1088\)](#).

Improvements

Security fixes and general improvements:

- Fixed security issues related to Aurora MySQL integration with other AWS Services such as Amazon S3, Amazon ML, and AWS Lambda.
- Fixed an issue where the value of a TIMESTAMP column of an existing row is updated to the latest timestamp when all of the following conditions are satisfied: 1. a trigger exists for the table; 2. an INSERT is performed on the table that has an ON DUPLICATE KEY UPDATE clause; 3. the inserted row can cause a duplicate value violation in a UNIQUE index or PRIMARY KEY; and 4. one or more columns are of TIMESTAMP data type and have a default value of CURRENT_TIMESTAMP.
- Fixed an issue which, in rare conditions, causes a writer instance to restart when an in-memory data-integrity check fails.
- Fixed an issue with parallel query which could cause the database to restart when executing SQL statements with a LIMIT clause.

Comparison with Aurora MySQL version 1

The following Amazon Aurora MySQL features are supported in Aurora MySQL Version 1 (compatible with MySQL 5.6), but these features are currently not supported in Aurora MySQL Version 2 (compatible with MySQL 5.7).

- Asynchronous key prefetch (AKP). For more information, see [Optimizing Amazon Aurora indexed join queries with asynchronous key prefetch \(p. 1036\)](#).
- Hash joins. For more information, see [Optimizing large Aurora MySQL join queries with hash joins \(p. 1038\)](#).
- Native functions for synchronously invoking AWS Lambda functions. For more information, see [Invoking a Lambda function with an Aurora MySQL native function \(p. 1012\)](#).
- Scan batching. For more information, see [Aurora MySQL database engine updates 2017-12-11 \(p. 1225\)](#).
- Migrating data from MySQL using an Amazon S3 bucket. For more information, see [Migrating data from MySQL by using an Amazon S3 bucket \(p. 784\)](#).

MySQL 5.7 compatibility

This Aurora MySQL version is wire-compatible with MySQL 5.7 and includes features such as JSON support, spatial indexes, and generated columns. Aurora MySQL uses a native implementation of spatial indexing using z-order curves to deliver >20x better write performance and >10x better read performance than MySQL 5.7 for spatial datasets.

This Aurora MySQL version does not currently support the following MySQL 5.7 features:

- Group replication plugin
- Increased page size

- InnoDB buffer pool loading at startup
- InnoDB full-text parser plugin
- Multisource replication
- Online buffer pool resizing
- Password validation plugin
- Query rewrite plugins
- Replication filtering
- The `CREATE TABLESPACE` SQL statement

Aurora MySQL database engine updates 2020-11-12 (version 2.08.3)

Version: 2.08.3

Aurora MySQL 2.08.3 is generally available. Aurora MySQL 2.x versions are compatible with MySQL 5.7 and Aurora MySQL 1.x versions are compatible with MySQL 5.6.

Currently supported Aurora MySQL releases for upgrade to 2.08.3 are: 1.14.* , 1.15.* , 1.16.* , 1.17.* , 1.18.* , 1.19.* , 1.20.* , 1.21.* , 1.22.* , 2.01.* , 2.02.* , 2.03.* , 2.04.* , 2.05.* , 2.06.* , 2.07.* , and 2.08.*.

You can upgrade existing Aurora MySQL 2.* database clusters directly to Aurora MySQL 2.08.3. You can upgrade an existing Aurora MySQL 1.* cluster directly to 2.07.3 or higher and then directly upgrade to 2.08.3.

To create a cluster with an older version of Aurora MySQL, specify the engine version through the AWS Management Console, the AWS CLI, or the RDS API.

If you have any questions or concerns, AWS Support is available on the community forums and through [AWS Premium Support](#). For more information, see [Maintaining an Amazon Aurora DB cluster \(p. 443\)](#).

Note

For information on how to upgrade your Aurora MySQL database cluster, see [Upgrading the minor version or patch level of an Aurora MySQL DB cluster \(p. 1088\)](#).

Improvements

Security fixes:

Fixes and other enhancements to fine-tune handling in a managed environment. Additional CVE fixes below:

- [CVE-2020-14567](#)
- [CVE-2020-14559](#)
- [CVE-2020-14553](#)
- [CVE-2020-14547](#)
- [CVE-2020-14540](#)
- [CVE-2020-2812](#)
- [CVE-2020-2806](#)
- [CVE-2020-2780](#)
- [CVE-2020-2765](#)
- [CVE-2020-2763](#)
- [CVE-2020-2760](#)

- [CVE-2020-2579](#)

Incompatible changes:

This version introduces a permission change that affects the behavior of the `mysqldump` command. Users must have the `PROCESS` privilege to access the `INFORMATION_SCHEMA.FILES` table. To run the `mysqldump` command without any changes, grant the `PROCESS` privilege to the database user that the `mysqldump` command connects to. You can also run the `mysqldump` command with the `--no-tablespaces` option. With that option, the `mysqldump` output doesn't include any `CREATE LOGFILE GROUP` or `CREATE TABLESPACE` statements. In that case, the `mysqldump` command doesn't access the `INFORMATION_SCHEMA.FILES` table, and you don't need to grant the `PROCESS` permission.

Integration of MySQL community edition bug fixes

- Bug #23762382 - INSERT VALUES QUERY WITH JOIN IN A SELECT CAUSES INCORRECT BEHAVIOR.
- Bug #25209512 - CURRENT_TIMESTAMP PRODUCES ZEROS IN TRIGGER.

Comparison with Aurora MySQL version 1

The following Amazon Aurora MySQL features are supported in Aurora MySQL Version 1 (compatible with MySQL 5.6), but these features are currently not supported in Aurora MySQL Version 2 (compatible with MySQL 5.7).

- Asynchronous key prefetch (AKP). For more information, see [Optimizing Amazon Aurora indexed join queries with asynchronous key prefetch \(p. 1036\)](#).
- Hash joins. For more information, see [Optimizing large Aurora MySQL join queries with hash joins \(p. 1038\)](#).
- Native functions for synchronously invoking AWS Lambda functions. For more information, see [Invoking a Lambda function with an Aurora MySQL native function \(p. 1012\)](#).
- Scan batching. For more information, see [Aurora MySQL database engine updates 2017-12-11 \(p. 1225\)](#).
- Migrating data from MySQL using an Amazon S3 bucket. For more information, see [Migrating data from MySQL by using an Amazon S3 bucket \(p. 784\)](#).

MySQL 5.7 compatibility

This Aurora MySQL version is wire-compatible with MySQL 5.7 and includes features such as JSON support, spatial indexes, and generated columns. Aurora MySQL uses a native implementation of spatial indexing using z-order curves to deliver >20x better write performance and >10x better read performance than MySQL 5.7 for spatial datasets.

This Aurora MySQL version does not currently support the following MySQL 5.7 features:

- Group replication plugin
- Increased page size
- InnoDB buffer pool loading at startup
- InnoDB full-text parser plugin
- Multisource replication
- Online buffer pool resizing
- Password validation plugin
- Query rewrite plugins
- Replication filtering

- The `CREATE TABLESPACE` SQL statement

Aurora MySQL database engine updates 2020-08-28 (version 2.08.2)

Version: 2.08.2

Aurora MySQL 2.08.2 is generally available. Aurora MySQL 2.x versions are compatible with MySQL 5.7 and Aurora MySQL 1.x versions are compatible with MySQL 5.6.

Currently supported Aurora MySQL releases are 1.14.* , 1.15.* , 1.16.* , 1.17.* , 1.18.* , 1.19.* , 1.20.* , 1.21.* , 1.22.* , 2.01.* , 2.02.* , 2.03.* , 2.04.* , 2.05.* , 2.06.* , 2.07.* , and 2.08.*.

You can restore a snapshot from a currently supported Aurora MySQL release into Aurora MySQL 2.08.2. You also have the option to upgrade existing Aurora MySQL 2.* database clusters to Aurora MySQL 2.08.2. You can't upgrade an existing Aurora MySQL 1.* cluster directly to 2.08.2; however, you can restore its snapshot to Aurora MySQL 2.08.2. See [Restoring from a DB cluster snapshot \(p. 497\)](#) for more information about restoring snapshots.

To create a cluster with an older version of Aurora MySQL, specify the engine version through the AWS Management Console, the AWS CLI, or the RDS API.

If you have any questions or concerns, AWS Support is available on the community forums and through [AWS Premium Support](#). For more information, see [Maintaining an Amazon Aurora DB cluster \(p. 443\)](#).

Note

For information on how to upgrade your Aurora MySQL database cluster, see [Upgrading the minor version or patch level of an Aurora MySQL DB cluster \(p. 1088\)](#).

Improvements

Critical fixes:

- Fixed an issue that might cause an unplanned outage and affect database availability.

Availability fixes:

- Fixed an issue where the Aurora MySQL database could restart if it is a binlog replica and replicates a DDL event over the `mysql.time_zone` table.

Comparison with Aurora MySQL version 1

The following Amazon Aurora MySQL features are supported in Aurora MySQL Version 1 (compatible with MySQL 5.6), but these features are currently not supported in Aurora MySQL Version 2 (compatible with MySQL 5.7).

- Asynchronous key prefetch (AKP). For more information, see [Optimizing Amazon Aurora indexed join queries with asynchronous key prefetch \(p. 1036\)](#).
- Hash joins. For more information, see [Optimizing large Aurora MySQL join queries with hash joins \(p. 1038\)](#).
- Native functions for synchronously invoking AWS Lambda functions. For more information, see [Invoking a Lambda function with an Aurora MySQL native function \(p. 1012\)](#).
- Scan batching. For more information, see [Aurora MySQL database engine updates 2017-12-11 \(p. 1225\)](#).
- Migrating data from MySQL using an Amazon S3 bucket. For more information, see [Migrating data from MySQL by using an Amazon S3 bucket \(p. 784\)](#).

MySQL 5.7 compatibility

This Aurora MySQL version is wire-compatible with MySQL 5.7 and includes features such as JSON support, spatial indexes, and generated columns. Aurora MySQL uses a native implementation of spatial indexing using z-order curves to deliver >20x better write performance and >10x better read performance than MySQL 5.7 for spatial datasets.

This Aurora MySQL version does not currently support the following MySQL 5.7 features:

- Group replication plugin
- Increased page size
- InnoDB buffer pool loading at startup
- InnoDB full-text parser plugin
- Multisource replication
- Online buffer pool resizing
- Password validation plugin
- Query rewrite plugins
- Replication filtering
- The `CREATE TABLESPACE` SQL statement

Aurora MySQL database engine updates 2020-06-18 (version 2.08.1)

Version: 2.08.1

Aurora MySQL 2.08.1 is generally available. Aurora MySQL 2.x versions are compatible with MySQL 5.7 and Aurora MySQL 1.x versions are compatible with MySQL 5.6.

Currently supported Aurora MySQL releases are 1.14.* , 1.15.* , 1.16.* , 1.17.* , 1.18.* , 1.19.* , 1.20.* , 1.21.* , 1.22.* , 2.01.* , 2.02.* , 2.03.* , 2.04.* , 2.05.* , 2.06.* , 2.07.* , and 2.08.* .

You can restore a snapshot from a currently supported Aurora MySQL release into Aurora MySQL 2.08.1. You also have the option to upgrade existing Aurora MySQL 2.* database clusters to Aurora MySQL 2.08.1. You can't upgrade an existing Aurora MySQL 1.* cluster directly to 2.08.1; however, you can restore its snapshot to Aurora MySQL 2.08.1.

To create a cluster with an older version of Aurora MySQL, specify the engine version through the AWS Management Console, the AWS CLI, or the RDS API.

If you have any questions or concerns, AWS Support is available on the community forums and through [AWS Premium Support](#). For more information, see [Maintaining an Amazon Aurora DB cluster \(p. 443\)](#).

Note

For information on how to upgrade your Aurora MySQL database cluster, see [Upgrading the minor version or patch level of an Aurora MySQL DB cluster \(p. 1088\)](#).

Improvements

New features:

- Global database write forwarding. In an Aurora global database, now you can perform certain write operations, such as DML statements, while connected to a secondary cluster. The write operations are forwarded to the primary cluster, and any changes are replicated back to the secondary clusters. For more information, see [Using write forwarding in an Amazon Aurora global database \(p. 255\)](#).

General stability fixes:

- Fixed an issue where restoring a cluster from Amazon S3 to Aurora MySQL version 2.08.0 took longer than expected if the S3 backup didn't include the `mysql.host` table.

Comparison with Aurora MySQL version 1

The following Amazon Aurora MySQL features are supported in Aurora MySQL Version 1 (compatible with MySQL 5.6), but these features are currently not supported in Aurora MySQL Version 2 (compatible with MySQL 5.7).

- Asynchronous key prefetch (AKP). For more information, see [Optimizing Amazon Aurora indexed join queries with asynchronous key prefetch \(p. 1036\)](#).
- Hash joins. For more information, see [Optimizing large Aurora MySQL join queries with hash joins \(p. 1038\)](#).
- Native functions for synchronously invoking AWS Lambda functions. For more information, see [Invoking a Lambda function with an Aurora MySQL native function \(p. 1012\)](#).
- Scan batching. For more information, see [Aurora MySQL database engine updates 2017-12-11 \(p. 1225\)](#).
- Migrating data from MySQL using an Amazon S3 bucket. For more information, see [Migrating data from MySQL by using an Amazon S3 bucket \(p. 784\)](#).

MySQL 5.7 compatibility

This Aurora MySQL version is wire-compatible with MySQL 5.7 and includes features such as JSON support, spatial indexes, and generated columns. Aurora MySQL uses a native implementation of spatial indexing using z-order curves to deliver >20x better write performance and >10x better read performance than MySQL 5.7 for spatial datasets.

This Aurora MySQL version does not currently support the following MySQL 5.7 features:

- Group replication plugin
- Increased page size
- InnoDB buffer pool loading at startup
- InnoDB full-text parser plugin
- Multisource replication
- Online buffer pool resizing
- Password validation plugin
- Query rewrite plugins
- Replication filtering
- The `CREATE TABLESPACE` SQL statement

Aurora MySQL database engine updates 2020-06-02 (version 2.08.0)

Version: 2.08.0

Aurora MySQL 2.08.0 is generally available. Aurora MySQL 2.x versions are compatible with MySQL 5.7 and Aurora MySQL 1.x versions are compatible with MySQL 5.6.

Currently supported Aurora MySQL releases are 1.14.* , 1.15.* , 1.16.* , 1.17.* , 1.18.* , 1.19.* , 1.20.* , 1.21.* , 1.22.* , 2.01.* , 2.02.* , 2.03.* , 2.04.* , 2.05.* , 2.06.* , 2.07.* , and 2.08.* .

You can restore a snapshot from a currently supported Aurora MySQL release into Aurora MySQL 2.08.0. You also have the option to upgrade existing Aurora MySQL 2.* database clusters to Aurora MySQL 2.08.0. You can't upgrade an existing Aurora MySQL 1.* cluster directly to 2.08.0; however, you can restore its snapshot to Aurora MySQL 2.08.0.

To create a cluster with an older version of Aurora MySQL, specify the engine version through the AWS Management Console, the AWS CLI, or the RDS API.

If you have any questions or concerns, AWS Support is available on the community forums and through [AWS Premium Support](#). For more information, see [Maintaining an Amazon Aurora DB cluster \(p. 443\)](#).

Note

For information on how to upgrade your Aurora MySQL database cluster, see [Upgrading the minor version or patch level of an Aurora MySQL DB cluster \(p. 1088\)](#).

Improvements

New features:

- Improved binary log (binlog) processing to reduce crash recovery time and commit time latency when very large transactions are involved.
- Launching Database Activity Streams (DAS) feature for Aurora MySQL. This feature provides a near-real-time data stream of the database activity in your relational database to help you monitor activity. For more information, see [Monitoring Amazon Aurora with Database Activity Streams \(p. 714\)](#).
- Updated timezone files to support the latest Brazil timezone change.
- Introduced new keywords in SQL to exercise the hash join functionality for a specific table and/or inner table: `HASH_JOIN`, `HASH_JOIN_PROBING`, and `HASH_JOIN_BUILDING`. For additional details, see [Aurora MySQL hints \(p. 1074\)](#).
- Introduced join order hint support in Aurora MySQL 5.7 by backporting [a MySQL 8.0 feature](#). The new hints are `JOIN_FIXED_ORDER`, `JOIN_ORDER`, `JOIN_PREFIX`, and `JOIN_SUFFIX`. For detailed documentation of join order hint support, see [WL#9158: Join order hints](#).
- Aurora Machine Learning now supports user-defined functions with `MEDIUMINT` as the return type.
- The `lambda_async()` stored procedure now supports all MySQL `utf8` characters.

High priority fixes:

- Fixed an issue that could cause a reader DB instance to return incomplete results for an FTS query after the `INFORMATION_SCHEMA.INNODB_SYS_TABLES` table is queried on the writer DB instance.
- [CVE-2019-5443](#)
- [CVE-2019-3822](#)

Availability improvements:

- Fixed an issue that resulted in a database restart after a multi-query statement that accesses multiple tables or databases is executed with the query cache enabled.
- Fixed a race condition in the lock manager that resulted in a database restart or failover during transaction rollback.
- Fixed an issue that triggered database restart or failover when multiple connections are trying to update the same table with a Full-Text Search index.
- Fixed an issue that could trigger a database restart or failover during a `kill session` command. If you encounter this issue, contact AWS support to enable this fix on your instance.
- Fixed an issue that caused reader DB instance to restart during a multi-statement transaction with multiple `SELECT` statements and a heavy write workload on the writer DB instance with `AUTOCOMMIT` enabled.

- Fixed an issue that caused reader DB instance to restart after executing long-running queries while the writer DB instance is under a heavy OLTP write workload.

General improvements:

- Improved database recovery time and commit latency for long running transactions when binlog is enabled.
- Improved the algorithm to generate better statistics for estimating distinct value counts on indexed columns, including columns with skewed data distributions.
- Reduced the response time and CPU utilization of join queries that access MyISAM temporary tables and the results spill to local storage.
- Fixed an issue that prevented Aurora MySQL 5.6 snapshots with database or table names containing spaces from being restored to a new Aurora MySQL 5.7 cluster.
- Included victim transaction info when deadlock is resolved in `show engine innodb status`.
- Fixed an issue that caused connections to get stuck when clients of multiple different versions are connected to the same database and are accessing the query cache.
- Fixed a memory leak resulting from multiple invocations of the Zero-Downtime Patch (ZDP) or Zero-Downtime Restart (ZDR) workflow throughout the lifetime of a database instance.
- Fixed an error message in Zero-Downtime Patch (ZDP) or Zero-Downtime Restart (ZDR) operations wrongly stating that the last transaction was aborted if the auto-commit flag is turned off.
- Fixed an issue in Zero-Downtime Patch (ZDP) operations that could lead to a server failure error message when restoring user session variables in the new database process.
- Fixed an issue in Zero Downtime Patch (ZDP) operations that might cause intermittent database failures when there are long running queries during patching.
- Fixed an issue where queries including an Aurora Machine Learning function returned empty error messages due to an incorrectly handled error response from Machine Learning services such as Amazon Sagemaker and Amazon Comprehend.
- Fixed an issue in the out-of-memory monitoring functionality that did not honor a custom value of the `table_definition_cache` parameter.
- The error message "Query execution was interrupted" is returned if an Aurora Machine Learning query is interrupted. Previously, the generic message "Internal error in processing ML request" was returned instead.
- Fixed an issue that could cause a binlog worker to experience a connection timeout when the `slave_net_timeout` parameter is less than the `aurora_binlog_replication_max_yield_seconds` parameter and there is low workload on the binlog master cluster.
- Improved monitoring of the binlog recovery progress by outputting informational messages in the error log at a frequency of one message per minute.
- Fixed an issue that could cause active transactions not to be reported by the `SHOW ENGINE INNODB STATUS` query.

Integration of MySQL community edition bug fixes

- [Bug #25289359](#): A full-text cache lock taken when data is synchronized was not released if the full-text cache size exceeded the full-text cache size limit.
- [Bug #29138644](#): Manually changing the system time while the MySQL server was running caused page cleaner thread delays.
- [Bug #25222337](#): A NULL virtual column field name in a virtual index caused a server exit during a field name comparison that occurs while populating virtual columns affected by a foreign key constraint.
- [Bug #25053286](#): Executing a stored procedure containing a query that accessed a view could allocate memory that was not freed until the session ended.

- [Bug #25586773](#): Executing a stored procedure containing a statement that created a table from the contents of certain `SELECT` statements could result in a memory leak.
- [Bug #28834208](#): During log application, after an `OPTIMIZE TABLE` operation, InnoDB did not populate virtual columns before checking for virtual column index updates.
- [Bug #26666274](#): Infinite loop in performance schema buffer container due to 32-bit unsigned integer overflow.

Comparison with Aurora MySQL version 1

The following Amazon Aurora MySQL features are supported in Aurora MySQL Version 1 (compatible with MySQL 5.6), but these features are currently not supported in Aurora MySQL Version 2 (compatible with MySQL 5.7).

- Asynchronous key prefetch (AKP). For more information, see [Optimizing Amazon Aurora indexed join queries with asynchronous key prefetch \(p. 1036\)](#).
- Hash joins. For more information, see [Optimizing large Aurora MySQL join queries with hash joins \(p. 1038\)](#).
- Native functions for synchronously invoking AWS Lambda functions. For more information, see [Invoking a Lambda function with an Aurora MySQL native function \(p. 1012\)](#).
- Scan batching. For more information, see [Aurora MySQL database engine updates 2017-12-11 \(p. 1225\)](#).
- Migrating data from MySQL using an Amazon S3 bucket. For more information, see [Migrating data from MySQL by using an Amazon S3 bucket \(p. 784\)](#).

MySQL 5.7 compatibility

This Aurora MySQL version is wire-compatible with MySQL 5.7 and includes features such as JSON support, spatial indexes, and generated columns. Aurora MySQL uses a native implementation of spatial indexing using z-order curves to deliver >20x better write performance and >10x better read performance than MySQL 5.7 for spatial datasets.

This Aurora MySQL version does not currently support the following MySQL 5.7 features:

- Group replication plugin
- Increased page size
- InnoDB buffer pool loading at startup
- InnoDB full-text parser plugin
- Multisource replication
- Online buffer pool resizing
- Password validation plugin
- Query rewrite plugins
- Replication filtering
- The `CREATE TABLESPACE` SQL statement

Aurora MySQL database engine updates 2021-11-24 (version 2.07.7)

Version: 2.07.7

Aurora MySQL 2.07.7 is generally available. Aurora MySQL 2.* versions are compatible with MySQL 5.7 and Aurora MySQL 1.* versions are compatible with MySQL 5.6.

Note

This version is designated as a long-term support (LTS) release. For more information, see [Aurora MySQL long-term support \(LTS\) releases \(p. 1085\)](#).

If you have any questions or concerns, AWS Support is available on the community forums and through [AWS Premium Support](#). For more information, see [Maintaining an Amazon Aurora DB cluster \(p. 443\)](#).

Improvements

Security fixes:

Fixes and other enhancements to fine-tune handling in a managed environment. Additional CVE fixes below:

- [CVE-2019-17543](#)
- [CVE-2019-2960](#)

General improvements:

- Fixed security issues related to Aurora MySQL integration with other AWS Services such as Amazon S3, Amazon ML, Lambda.
- Fixed an issue with incorrect reporting of an Aurora replication lag.
- Fixed an issue which can cause a database instance restart to fail when the database has a large number of user and privilege combinations.
- Fixed an issue which can cause general_log and slow_log tables to become inaccessible after in-place major version upgrade from Aurora MySQL 1.x (based on MySQL 5.6) to Aurora MySQL 2.x (based on MySQL 5.7).
- Fixed an issue which, in rare conditions, can cause a reader instance to restart due to an incorrect check processing.
- Fixed an issue which, in rare conditions, shows the "Database Load" chart in Performance Insights (PI) sessions as actively using CPU even though the sessions have finished processing and are idle.
- Fixed an issue with parallel query which can cause the database to restart when executing SQL statements with a LIMIT clause.
- Fixed an issue where the value of a TIMESTAMP column of an existing row is updated to the latest timestamp when all of the following conditions are satisfied: 1. A trigger exists for the table; 2. an INSERT is performed on the table that has an ON DUPLICATE KEY UPDATE clause; 3. the inserted row can cause a duplicate value violation in a UNIQUE index or PRIMARY KEY; and, 4. one or more columns are of TIMESTAMP data type and have a default value of CURRENT_TIMESTAMP.
- Fixed an issue which, in rare conditions, can cause the database instance to restart when using XA transactions in READ COMMITTED isolation level.

Integration of MySQL community edition bug fixes

- INSERTING 64K SIZE RECORDS TAKE TOO MUCH TIME. ([Bug#23031146](#))

Comparison with Aurora MySQL version 1

The following Amazon Aurora MySQL features are supported in Aurora MySQL Version 1 (compatible with MySQL 5.6), but these features are currently not supported in Aurora MySQL Version 2 (compatible with MySQL 5.7).

- Asynchronous key prefetch (AKP). For more information, see [Optimizing Amazon Aurora indexed join queries with asynchronous key prefetch \(p. 1036\)](#).

- Hash joins. For more information, see [Optimizing large Aurora MySQL join queries with hash joins \(p. 1038\)](#).
- Native functions for synchronously invoking AWS Lambda functions. For more information, see [Invoking a Lambda function with an Aurora MySQL native function \(p. 1012\)](#).
- Scan batching. For more information, see [Aurora MySQL database engine updates 2017-12-11 \(p. 1225\)](#).
- Migrating data from MySQL using an Amazon S3 bucket. For more information, see [Migrating data from MySQL by using an Amazon S3 bucket \(p. 784\)](#).

MySQL 5.7 compatibility

This Aurora MySQL version is wire-compatible with MySQL 5.7 and includes features such as JSON support, spatial indexes, and generated columns. Aurora MySQL uses a native implementation of spatial indexing using z-order curves to deliver >20x better write performance and >10x better read performance than MySQL 5.7 for spatial datasets.

This Aurora MySQL version does not currently support the following MySQL 5.7 features:

- Group replication plugin
- Increased page size
- InnoDB buffer pool loading at startup
- InnoDB full-text parser plugin
- Multisource replication
- Online buffer pool resizing
- Password validation plugin
- Query rewrite plugins
- Replication filtering
- The `CREATE TABLESPACE` SQL statement

Aurora MySQL database engine updates 2021-09-02 (version 2.07.6)

Version: 2.07.6

Aurora MySQL 2.07.6 is generally available. Aurora MySQL 2.* versions are compatible with MySQL 5.7 and Aurora MySQL 1.* versions are compatible with MySQL 5.6.

Currently supported Aurora MySQL releases are 1.14.*, 1.15.*, 1.16.*, 1.17.*, 1.18.*, 1.19.*, 1.20.*, 1.21.*, 1.22.*, 1.23.*, 2.01.*, 2.02.*, 2.03.*, 2.04.*, 2.05.*, 2.06.*, 2.07.*, 2.08.*, 2.09.*, and 2.10.*.

You can restore a snapshot from a currently supported Aurora MySQL release into Aurora MySQL 2.07.6. You also have the option to upgrade existing Aurora MySQL 2.* database clusters to Aurora MySQL 2.07.6. You can't upgrade an existing Aurora MySQL 1.* cluster directly to 2.07.6; however, you can restore its snapshot to Aurora MySQL 2.07.6.

To create a cluster with an older version of Aurora MySQL, please specify the engine version through the AWS Management Console, the AWS CLI, or the RDS API.

If you have any questions or concerns, AWS Support is available on the community forums and through [AWS Premium Support](#). For more information, see [Maintaining an Amazon Aurora DB cluster \(p. 443\)](#).

Integration of MySQL community edition bug fixes

- INSERTING 64K SIZE RECORDS TAKE TOO MUCH TIME. ([Bug#23031146](#))

Comparison with Aurora MySQL version 1

The following Amazon Aurora MySQL features are supported in Aurora MySQL Version 1 (compatible with MySQL 5.6), but these features are currently not supported in Aurora MySQL Version 2 (compatible with MySQL 5.7).

- Asynchronous key prefetch (AKP). For more information, see [Optimizing Amazon Aurora indexed join queries with asynchronous key prefetch \(p. 1036\)](#).
- Hash joins. For more information, see [Optimizing large Aurora MySQL join queries with hash joins \(p. 1038\)](#).
- Native functions for synchronously invoking AWS Lambda functions. For more information, see [Invoking a Lambda function with an Aurora MySQL native function \(p. 1012\)](#).
- Scan batching. For more information, see [Aurora MySQL database engine updates 2017-12-11 \(p. 1225\)](#).
- Migrating data from MySQL using an Amazon S3 bucket. For more information, see [Migrating data from MySQL by using an Amazon S3 bucket \(p. 784\)](#).

MySQL 5.7 compatibility

This Aurora MySQL version is wire-compatible with MySQL 5.7 and includes features such as JSON support, spatial indexes, and generated columns. Aurora MySQL uses a native implementation of spatial indexing using z-order curves to deliver >20x better write performance and >10x better read performance than MySQL 5.7 for spatial datasets.

This Aurora MySQL version does not currently support the following MySQL 5.7 features:

- Group replication plugin
- Increased page size
- InnoDB buffer pool loading at startup
- InnoDB full-text parser plugin
- Multisource replication
- Online buffer pool resizing
- Password validation plugin
- Query rewrite plugins
- Replication filtering
- The `CREATE TABLESPACE` SQL statement

Aurora MySQL database engine updates 2021-07-06 (version 2.07.5)

Version: 2.07.5

Aurora MySQL 2.07.5 is generally available. Aurora MySQL 2.* versions are compatible with MySQL 5.7 and Aurora MySQL 1.* versions are compatible with MySQL 5.6.

Currently supported Aurora MySQL releases are 1.14.*, 1.15.*, 1.16.*, 1.17.*, 1.18.*, 1.19.*, 1.20.*, 1.21.*, 1.22.*, 1.23.*, 2.01.*, 2.02.*, 2.03.*, 2.04.*, 2.05.*, 2.06.*, 2.07*, 2.08*, 2.09*, and 2.10*.

You can restore a snapshot from a currently supported Aurora MySQL release into Aurora MySQL 2.07.5. You also have the option to upgrade existing Aurora MySQL 2.* database clusters to Aurora MySQL

2.07.5. You can't upgrade an existing Aurora MySQL 1.* cluster directly to 2.07.5; however, you can restore its snapshot to Aurora MySQL 2.07.5.

To create a cluster with an older version of Aurora MySQL, please specify the engine version through the AWS Management Console, the AWS CLI, or the RDS API.

Note

This version is designated as a long-term support (LTS) release. For more information, see [Aurora MySQL long-term support \(LTS\) releases \(p. 1085\)](#).

If you have any questions or concerns, AWS Support is available on the community forums and through [AWS Premium Support](#). For more information, see [Maintaining an Amazon Aurora DB cluster \(p. 443\)](#).

Improvements

Availability improvements:

- Fixed an issue that user-level locks are not allowed on an Aurora Replica.
- Fixed an issue that could cause a restart of a database when using XA transactions in `READ COMMITTED` isolation level.
- Extended maximum allowable length to 2000 for the `server_audit_incl_users` and `server_audit_excl_users` global parameters.

Comparison with Aurora MySQL version 1

The following Amazon Aurora MySQL features are supported in Aurora MySQL Version 1 (compatible with MySQL 5.6), but these features are currently not supported in Aurora MySQL Version 2 (compatible with MySQL 5.7).

- Asynchronous key prefetch (AKP). For more information, see [Optimizing Amazon Aurora indexed join queries with asynchronous key prefetch \(p. 1036\)](#).
- Hash joins. For more information, see [Optimizing large Aurora MySQL join queries with hash joins \(p. 1038\)](#).
- Native functions for synchronously invoking AWS Lambda functions. For more information, see [Invoking a Lambda function with an Aurora MySQL native function \(p. 1012\)](#).
- Scan batching. For more information, see [Aurora MySQL database engine updates 2017-12-11 \(p. 1225\)](#).
- Migrating data from MySQL using an Amazon S3 bucket. For more information, see [Migrating data from MySQL by using an Amazon S3 bucket \(p. 784\)](#).

MySQL 5.7 compatibility

This Aurora MySQL version is wire-compatible with MySQL 5.7 and includes features such as JSON support, spatial indexes, and generated columns. Aurora MySQL uses a native implementation of spatial indexing using z-order curves to deliver >20x better write performance and >10x better read performance than MySQL 5.7 for spatial datasets.

This Aurora MySQL version does not currently support the following MySQL 5.7 features:

- Group replication plugin
- Increased page size
- InnoDB buffer pool loading at startup
- InnoDB full-text parser plugin
- Multisource replication

- Online buffer pool resizing
- Password validation plugin
- Query rewrite plugins
- Replication filtering
- The `CREATE TABLESPACE` SQL statement

Aurora MySQL database engine updates 2021-03-04 (version 2.07.4)

Version: 2.07.4

Aurora MySQL 2.07.4 is generally available. Aurora MySQL 2.* versions are compatible with MySQL 5.7 and Aurora MySQL 1.* versions are compatible with MySQL 5.6.

Currently supported Aurora MySQL releases are 1.14.*, 1.15.*, 1.16.*, 1.17.*, 1.18.*, 1.19.*, 1.20.*, 1.21.*, 1.22.*, 1.23.*, 2.01.*, 2.02.*, 2.03.*, 2.04.*, 2.05.*, 2.06.*, 2.07.*, 2.08.*, and 2.09.*.

You can restore a snapshot from a currently supported Aurora MySQL release into Aurora MySQL 2.07.4. You also have the option to upgrade existing Aurora MySQL 2.* database clusters to Aurora MySQL 2.07.4. You can't upgrade an existing Aurora MySQL 1.* cluster directly to 2.07.4; however, you can restore its snapshot to Aurora MySQL 2.07.4.

To create a cluster with an older version of Aurora MySQL, please specify the engine version through the AWS Management Console, the AWS CLI, or the RDS API.

Note

This version is designated as a long-term support (LTS) release. For more information, see [Aurora MySQL long-term support \(LTS\) releases \(p. 1085\)](#).

If you have any questions or concerns, AWS Support is available on the community forums and through [AWS Premium Support](#). For more information, see [Maintaining an Amazon Aurora DB cluster \(p. 443\)](#).

Improvements

Security fixes:

- [CVE-2020-14812](#)
- [CVE-2020-14793](#)
- [CVE-2020-14790](#)
- [CVE-2020-14775](#)
- [CVE-2020-14769](#)
- [CVE-2020-14765](#)
- [CVE-2020-14760](#)
- [CVE-2020-14672](#)
- [CVE-2020-1971](#)

Availability improvements:

- Fixed an issue that could cause a client to hang in case of a network error while reading or writing a network packet.
- Improved engine restart times in some cases after interrupted DDL.
- Fixed an issue where a DDL or DML could cause engine restart during a page prefetch request.

- Fixed an issue where a replica could restart while performing a reverse scan of a table/index on an Aurora Read Replica.
- Fixed an issue in clone cluster operation that could cause the clone to take longer.
- Fixed an issue that could cause a restart of a database when using parallel query optimization for geo-spatial column.
- Fixed an issue that caused a binlog replica to stop with an `HA_ERR_KEY_NOT_FOUND` error.

Integration of MySQL community edition bug fixes

- Fixed an issue in the Full-text ngram parser when dealing with tokens containing '' (space), '%', or '.'. Customers should rebuild their FTS indexes if using ngram parser. (Bug #25873310)
- Fixed an issue that could cause engine restart during query execution with nested SQL views. (Bug #27214153, Bug #26864199)

Comparison with Aurora MySQL version 1

The following Amazon Aurora MySQL features are supported in Aurora MySQL Version 1 (compatible with MySQL 5.6), but these features are currently not supported in Aurora MySQL Version 2 (compatible with MySQL 5.7).

- Asynchronous key prefetch (AKP). For more information, see [Optimizing Amazon Aurora indexed join queries with asynchronous key prefetch \(p. 1036\)](#).
- Hash joins. For more information, see [Optimizing large Aurora MySQL join queries with hash joins \(p. 1038\)](#).
- Native functions for synchronously invoking AWS Lambda functions. For more information, see [Invoking a Lambda function with an Aurora MySQL native function \(p. 1012\)](#).
- Scan batching. For more information, see [Aurora MySQL database engine updates 2017-12-11 \(p. 1225\)](#).
- Migrating data from MySQL using an Amazon S3 bucket. For more information, see [Migrating data from MySQL by using an Amazon S3 bucket \(p. 784\)](#).

MySQL 5.7 compatibility

This Aurora MySQL version is wire-compatible with MySQL 5.7 and includes features such as JSON support, spatial indexes, and generated columns. Aurora MySQL uses a native implementation of spatial indexing using z-order curves to deliver >20x better write performance and >10x better read performance than MySQL 5.7 for spatial datasets.

This Aurora MySQL version does not currently support the following MySQL 5.7 features:

- Group replication plugin
- Increased page size
- InnoDB buffer pool loading at startup
- InnoDB full-text parser plugin
- Multisource replication
- Online buffer pool resizing
- Password validation plugin
- Query rewrite plugins
- Replication filtering
- The `CREATE TABLESPACE` SQL statement

Aurora MySQL database engine updates 2020-11-10 (version 2.07.3)

Version: 2.07.3

Aurora MySQL 2.07.3 is generally available. Aurora MySQL 2.* versions are compatible with MySQL 5.7 and Aurora MySQL 1.* versions are compatible with MySQL 5.6.

Currently supported Aurora MySQL releases are 1.14.*, 1.15.*, 1.16.*, 1.17.*, 1.18.*, 1.19.*, 1.20.*, 1.21.*, 1.22.*, 1.23.*, 2.01.*, 2.02.*, 2.03.*, 2.04.*, 2.05.*, 2.06.*, 2.07.*, 2.08.*, and 2.09.*.

You can restore a snapshot from a currently supported Aurora MySQL release into Aurora MySQL 2.07.3. You also have the option to upgrade existing Aurora MySQL 2.* database clusters to Aurora MySQL 2.07.3. You can't upgrade an existing Aurora MySQL 1.* cluster directly to 2.07.3; however, you can restore its snapshot to Aurora MySQL 2.07.3.

To create a cluster with an older version of Aurora MySQL, please specify the engine version through the AWS Management Console, the AWS CLI, or the RDS API.

Note

This version is designated as a long-term support (LTS) release. For more information, see [Aurora MySQL long-term support \(LTS\) releases \(p. 1085\)](#).

If you have any questions or concerns, AWS Support is available on the community forums and through [AWS Premium Support](#). For more information, see [Maintaining an Amazon Aurora DB cluster \(p. 443\)](#).

Improvements

Security fixes:

Fixes and other enhancements to fine-tune handling in a managed environment. Additional CVE fixes below:

- [CVE-2020-14567](#)
- [CVE-2020-14559](#)
- [CVE-2020-14553](#)
- [CVE-2020-14547](#)
- [CVE-2020-14540](#)
- [CVE-2020-2812](#)
- [CVE-2020-2806](#)
- [CVE-2020-2780](#)
- [CVE-2020-2765](#)
- [CVE-2020-2763](#)
- [CVE-2020-2760](#)
- [CVE-2020-2579](#)
- [CVE-2019-2740](#)

Incompatible changes:

This version introduces a permission change that affects the behavior of the `mysqldump` command. Users must have the `PROCESS` privilege to access the `INFORMATION_SCHEMA.FILES` table. To run the `mysqldump` command without any changes, grant the `PROCESS` privilege to the database user that the `mysqldump` command connects to. You can also run the `mysqldump` command with the `--no-tablespaces` option. With that option, the `mysqldump` output doesn't include any `CREATE LOGFILE`

GROUP or CREATE TABLESPACE statements. In that case, the `mysqldump` command doesn't access the `INFORMATION_SCHEMA.FILES` table, and you don't need to grant the `PROCESS` permission.

Availability improvements:

- Fixed a race condition in the lock manager between the killing of a connection/query and the termination of the session resulting in a database restart.
- Fixed an issue that results in a database restart after a multi-query statement that accesses multiple tables or databases is executed with the query cache enabled.
- Fixed an issue that might cause repeated restarts due to updates of virtual columns with secondary indexes.

Integration of MySQL community edition bug fixes

- *InnoDB*: Concurrent XA transactions that ran successfully to the XA prepare stage on the master conflicted when replayed on the slave, resulting in a lock wait timeout in the applier thread. The conflict was due to the GAP lock range which differed when the transactions were replayed serially on the slave. To prevent this type of conflict, GAP locks taken by XA transactions in `READ COMMITTED` isolation level are now released (and no longer inherited) when XA transactions reach the prepare stage. (Bug #27189701, Bug #25866046)
- *InnoDB*: A gap lock was taken unnecessarily during foreign key validation while using the `READ COMMITTED` isolation level. (Bug #25082593)
- *Replication*: When using XA transactions, if a lock wait timeout or deadlock occurred for the applier (SQL) thread on a replication slave, the automatic retry did not work. The cause was that while the SQL thread would do a rollback, it would not roll the XA transaction back. This meant that when the transaction was retried, the first event was XA START which was invalid as the XA transaction was already in progress, leading to an XAER_RMFAIL error. (Bug #24764800)
- *Replication*: Interleaved transactions could sometimes deadlock the slave applier when the transaction isolation level was set to `REPEATABLE READ`. (Bug #25040331)
- *Replication*: The value returned by a `SHOW SLAVE STATUS` statement for the total combined size of all existing relay log files (`Relay_Log_Space`) could become much larger than the actual disk space used by the relay log files. The I/O thread did not lock the variable while it updated the value, so the SQL thread could automatically delete a relay log file and write a reduced value before the I/O thread finished updating the value. The I/O thread then wrote its original size calculation, ignoring the SQL thread's update and so adding back the space for the deleted file. The `Relay_Log_Space` value is now locked during updates to prevent concurrent updates and ensure an accurate calculation. (Bug #26997096, Bug #87832)
- For an `INSERT` statement for which the `VALUES` list produced values for the second or later row using a subquery containing a join, the server could exit after failing to resolve the required privileges. (Bug #23762382)
- For a table having a `TIMESTAMP` or `DATETIME` column having a default of `CURRENT_TIMESTAMP`, the column could be initialized to `0000-00-00 00:00:00` if the table had a `BEFORE INSERT` trigger. (Bug #25209512, Bug #84077)
- A server exit could result from simultaneous attempts by multiple threads to register and deregister metadata Performance Schema objects. (Bug #26502135)
- Executing a stored procedure containing a statement that created a table from the contents of certain `SELECT` statements could result in a memory leak. (Bug #25586773)
- Executing a stored procedure containing a query that accessed a view could allocate memory that was not freed until the session ended. (Bug #25053286)
- Certain cases of subquery materialization could cause a server exit. These queries now produce an error suggesting that materialization be disabled. (Bug #26402045)
- Queries with many left joins were slow if join buffering was used (for example, using the block nested loop algorithm). (Bug #18898433, Bug #72854)

- The optimizer skipped the second column in a composite index when executing an inner join with a `LIKE` clause against the second column. (Bug #28086754)

Comparison with Aurora MySQL version 1

The following Amazon Aurora MySQL features are supported in Aurora MySQL Version 1 (compatible with MySQL 5.6), but these features are currently not supported in Aurora MySQL Version 2 (compatible with MySQL 5.7).

- Asynchronous key prefetch (AKP). For more information, see [Optimizing Amazon Aurora indexed join queries with asynchronous key prefetch \(p. 1036\)](#).
- Hash joins. For more information, see [Optimizing large Aurora MySQL join queries with hash joins \(p. 1038\)](#).
- Native functions for synchronously invoking AWS Lambda functions. For more information, see [Invoking a Lambda function with an Aurora MySQL native function \(p. 1012\)](#).
- Scan batching. For more information, see [Aurora MySQL database engine updates 2017-12-11 \(p. 1225\)](#).
- Migrating data from MySQL using an Amazon S3 bucket. For more information, see [Migrating data from MySQL by using an Amazon S3 bucket \(p. 784\)](#).

MySQL 5.7 compatibility

This Aurora MySQL version is wire-compatible with MySQL 5.7 and includes features such as JSON support, spatial indexes, and generated columns. Aurora MySQL uses a native implementation of spatial indexing using z-order curves to deliver >20x better write performance and >10x better read performance than MySQL 5.7 for spatial datasets.

This Aurora MySQL version does not currently support the following MySQL 5.7 features:

- Group replication plugin
- Increased page size
- InnoDB buffer pool loading at startup
- InnoDB full-text parser plugin
- Multisource replication
- Online buffer pool resizing
- Password validation plugin
- Query rewrite plugins
- Replication filtering
- The `CREATE TABLESPACE` SQL statement

Aurora MySQL database engine updates 2020-04-17 (version 2.07.2)

Version: 2.07.2

Aurora MySQL 2.07.2 is generally available. Aurora MySQL 2.* versions are compatible with MySQL 5.7 and Aurora MySQL 1.* versions are compatible with MySQL 5.6.

Currently supported Aurora MySQL releases are 1.14.*, 1.15.*, 1.16.*, 1.17.*, 1.18.*, 1.19.*, 1.20.*, 1.21.*, 1.22.*, 2.01.*, 2.02.*, 2.03.*, 2.04.*, 2.05.*, 2.06.*, and 2.07.*.

You can restore a snapshot from a currently supported Aurora MySQL release into Aurora MySQL 2.07.2. You also have the option to upgrade existing Aurora MySQL 2.* database clusters to Aurora MySQL 2.07.2. You can't upgrade an existing Aurora MySQL 1.* cluster directly to 2.07.2; however, you can restore its snapshot to Aurora MySQL 2.07.2.

To create a cluster with an older version of Aurora MySQL, please specify the engine version through the AWS Management Console, the AWS CLI, or the RDS API.

Note

This version is currently not available in the following AWS Region: [us-gov-east-1], AWS GovCloud (US-West) [us-gov-west-1]. There will be a separate announcement once it is made available.

This version is designated as a long-term support (LTS) release. For more information, see [Aurora MySQL long-term support \(LTS\) releases \(p. 1085\)](#).

If you have any questions or concerns, AWS Support is available on the community forums and through [AWS Premium Support](#). For more information, see [Maintaining an Amazon Aurora DB cluster \(p. 443\)](#).

Improvements

High priority fixes:

- Fixed an issue that caused cloning to take longer on some database clusters with high write loads.
- Fixed an issue that could cause queries on a reader DB instance with execution plans using secondary indexes to return uncommitted data. The issue is limited to data affected by data manipulation language (DML) operations that modify primary or secondary index key columns.

General improvements:

- Fixed an issue that resulted in a slow restore of an Aurora 1.x DB cluster containing FTS (Full Text Search) indexes to an Aurora 2.x DB cluster.
- Fixed an issue that caused slower restores of an Aurora 1.x database snapshot containing partitioned tables with special characters in table names to an Aurora 2.x DB cluster.
- Fixed an issue that caused errors when querying slow query logs and general logs in reader DB instances.

Integration of MySQL community edition bug fixes

- Bug #23104498: Fixed an issue in Performance Schema in reporting total memory used. (<https://github.com/mysql/mysql-server/commit/20b6840df5452f47313c6f9a6ca075bfbc00a96b>)
- Bug #22551677: Fixed an issue in Performance Schema that could lead to the database engine crashing when attempting to take it offline. (<https://github.com/mysql/mysql-server/commit/05e2386eccd32b6b444b900c9f8a87a1d8d531e9>)
- Bug #23550835, Bug #23298025, Bug #81464: Fixed an issue in Performance Schema that causes a database engine crash due to exceeding the capacity of an internal buffer. (<https://github.com/mysql/mysql-server/commit/b4287f93857bf2f99b18fd06f555bbe5b12debfc>)

Comparison with Aurora MySQL version 1

The following Amazon Aurora MySQL features are supported in Aurora MySQL Version 1 (compatible with MySQL 5.6), but these features are currently not supported in Aurora MySQL Version 2 (compatible with MySQL 5.7).

- Asynchronous key prefetch (AKP). For more information, see [Optimizing Amazon Aurora indexed join queries with asynchronous key prefetch \(p. 1036\)](#).

- Hash joins. For more information, see [Optimizing large Aurora MySQL join queries with hash joins \(p. 1038\)](#).
- Native functions for synchronously invoking AWS Lambda functions. For more information, see [Invoking a Lambda function with an Aurora MySQL native function \(p. 1012\)](#).
- Scan batching. For more information, see [Aurora MySQL database engine updates 2017-12-11 \(p. 1225\)](#).
- Migrating data from MySQL using an Amazon S3 bucket. For more information, see [Migrating data from MySQL by using an Amazon S3 bucket \(p. 784\)](#).

MySQL 5.7 compatibility

This Aurora MySQL version is wire-compatible with MySQL 5.7 and includes features such as JSON support, spatial indexes, and generated columns. Aurora MySQL uses a native implementation of spatial indexing using z-order curves to deliver >20x better write performance and >10x better read performance than MySQL 5.7 for spatial datasets.

This Aurora MySQL version does not currently support the following MySQL 5.7 features:

- Group replication plugin
- Increased page size
- InnoDB buffer pool loading at startup
- InnoDB full-text parser plugin
- Multisource replication
- Online buffer pool resizing
- Password validation plugin
- Query rewrite plugins
- Replication filtering
- The `CREATE TABLESPACE` SQL statement

Aurora MySQL database engine updates 2019-12-23 (version 2.07.1)

Version: 2.07.1

Aurora MySQL 2.07.1 is generally available. Aurora MySQL 2.x versions are compatible with MySQL 5.7 and Aurora MySQL 1.x versions are compatible with MySQL 5.6.

Currently supported Aurora MySQL releases are 1.14.* , 1.15.* , 1.16.* , 1.17.* , 1.18.* , 1.19.* , 1.20.* , 1.21.* , 1.22.* , 2.01.* , 2.02.* , 2.03.* , 2.04.* , 2.05.* , 2.06.* , and 2.07.* .

You can restore a snapshot from a currently supported Aurora MySQL release into Aurora MySQL 2.07.1. You also have the option to upgrade existing Aurora MySQL 2.* database clusters to Aurora MySQL 2.07.1. You cannot upgrade an existing Aurora MySQL 1.* cluster directly to 2.07.1; however, you can restore its snapshot to Aurora MySQL 2.07.1.

To create a cluster with an older version of Aurora MySQL, please specify the engine version through the AWS Management Console, the AWS CLI, or the RDS API.

Note

This version is currently not available in the following AWS Regions: AWS GovCloud (US-East) [us-gov-east-1], AWS GovCloud (US-West) [us-gov-west-1], China (Ningxia) [cn-northwest-1],

Asia Pacific (Hong Kong) [ap-east-1], and Middle East (Bahrain) [me-south-1]. There will be a separate announcement once it is made available.

If you have any questions or concerns, AWS Support is available on the community forums and through [AWS Premium Support](#). For more information, see [Maintaining an Amazon Aurora DB cluster \(p. 443\)](#).

Note

For information on how to upgrade your Aurora MySQL database cluster, see [Upgrading the minor version or patch level of an Aurora MySQL DB cluster \(p. 1088\)](#).

Improvements

High priority fixes:

- Fixed a slow memory leak in Aurora specific database tracing and logging sub-system that lowers the freeable memory.

General Stability fixes:

- Fixed a crash during execution of a complex query involving multi-table joins and aggregation that use intermediate tables internally.

Comparison with Aurora MySQL Version 1

The following Amazon Aurora MySQL features are supported in Aurora MySQL Version 1 (compatible with MySQL 5.6), but these features are currently not supported in Aurora MySQL Version 2 (compatible with MySQL 5.7).

- Asynchronous key prefetch (AKP). For more information, see [Optimizing Amazon Aurora indexed join queries with asynchronous key prefetch \(p. 1036\)](#).
- Scan batching. For more information, see [Aurora MySQL database engine updates 2017-12-11 \(p. 1225\)](#).
- Migrating data from MySQL using an Amazon S3 bucket. For more information, see [Migrating data from MySQL by using an Amazon S3 bucket \(p. 784\)](#).

MySQL 5.7 compatibility

This Aurora MySQL version is wire-compatible with MySQL 5.7 and includes features such as JSON support, spatial indexes, and generated columns. Aurora MySQL uses a native implementation of spatial indexing using z-order curves to deliver >20x better write performance and >10x better read performance than MySQL 5.7 for spatial datasets.

This Aurora MySQL version does not currently support the following MySQL 5.7 features:

- Group replication plugin
- Increased page size
- InnoDB buffer pool loading at startup
- InnoDB full-text parser plugin
- Multisource replication
- Online buffer pool resizing
- Password validation plugin
- Query rewrite plugins
- Replication filtering

- The `CREATE TABLESPACE` SQL statement

Aurora MySQL database engine updates 2019-11-25 (version 2.07.0)

Version: 2.07.0

Aurora MySQL 2.07.0 is generally available. Aurora MySQL 2.x versions are compatible with MySQL 5.7 and Aurora MySQL 1.x versions are compatible with MySQL 5.6.

Currently supported Aurora MySQL releases are 1.14.* , 1.15.* , 1.16.* , 1.17.* , 1.18.* , 1.19.* , 1.20.* , 1.21.* , 1.22.* , 2.01.* , 2.02.* , 2.03.* , 2.04.* , 2.05.* , 2.06.* , and 2.07.*.

You can restore a snapshot from a currently supported Aurora MySQL release into Aurora MySQL 2.07.0. You also have the option to upgrade existing Aurora MySQL 2.* database clusters to Aurora MySQL 2.07.0. You cannot upgrade an existing Aurora MySQL 1.* cluster directly to 2.07.0; however, you can restore its snapshot to Aurora MySQL 2.07.0.

To create a cluster with an older version of Aurora MySQL, please specify the engine version through the AWS Management Console, the AWS CLI, or the RDS API.

Note

This version is currently not available in the following AWS Regions: AWS GovCloud (US-East) [us-gov-east-1], AWS GovCloud (US-West) [us-gov-west-1], China (Ningxia) [cn-northwest-1], Asia Pacific (Hong Kong) [ap-east-1], Middle East (Bahrain) [me-south-1], and South America (São Paulo) [sa-east-1]. There will be a separate announcement once it is made available.

If you have any questions or concerns, AWS Support is available on the community forums and through [AWS Premium Support](#). For more information, see [Maintaining an Amazon Aurora DB cluster \(p. 443\)](#).

Note

For information on how to upgrade your Aurora MySQL database cluster, see [Upgrading the minor version or patch level of an Aurora MySQL DB cluster \(p. 1088\)](#).

Improvements

New features:

- Global Databases now allow adding secondary read-only replica regions for database clusters deployed in these AWS Regions: regions: US East (N. Virginia) [us-east-1], US East (Ohio) [us-east-2], US West (N. California) [us-west-1], US West (Oregon) [us-west-2], Europe (Ireland) [eu-west-1], Europe (London) [eu-west-2], Europe (Paris) [eu-west-3], Asia Pacific (Tokyo) [ap-northeast-1], Asia Pacific (Seoul) [ap-northeast-2], Asia Pacific (Singapore) [ap-southeast-1], Asia Pacific (Sydney) [ap-southeast-2], Canada (Central) [ca-central-1], Europe (Frankfurt) [eu-central-1], and Asia Pacific (Mumbai) [ap-south-1].
- Amazon Aurora machine learning is a highly optimized integration between the Aurora MySQL database and AWS machine learning (ML) services. Aurora machine learning allows developers to add a variety of ML-based predictions to their database applications by invoking ML models using the familiar SQL programming language they already use for database development, without having to build custom integrations or learn separate tools. For more information, see [Using machine learning \(ML\) capabilities with Amazon Aurora](#).
- Added support for the `ANSI READ COMMITTED` isolation level on the read replicas. This isolation level enables long-running queries on the read replica to execute without impacting the high throughput of writes on the writer node. For more information, see [Aurora MySQL isolation levels](#).

Critical fixes:

- [CVE-2019-2922](#)
- [CVE-2019-2923](#)
- [CVE-2019-2924](#)
- [CVE-2019-2910](#)

High-priority fixes:

- Fixed an issue in the DDL recovery that resulted in prolonged database downtime. Clusters that become unavailable after executing multi-table drop statement, for example `DROP TABLE t1, t2, t3`, should be updated to this version.
- Fixed an issue in the DDL recovery that resulted in prolonged database downtime. Clusters that become unavailable after executing `INPLACE ALTER TABLE` DDL statements should be updated to this version.

General stability fixes:

- Fixed an issue that generated inconsistent data in the `information_schema.replica_host_status` table.

Integration of MySQL community edition bug fixes

- Bug #26251621: INCORRECT BEHAVIOR WITH TRIGGER AND GCOL
- Bug #22574695: ASSERTION `!TABLE || (!TABLE->READ_SET || BITMAP_IS_SET(TABLE->READ_SET, FILE))`
- Bug #25966845: INSERT ON DUPLICATE KEY GENERATE A DEADLOCK
- Bug #23070734: CONCURRENT TRUNCATE TABLES CAUSE STALL
- Bug #26191879: FOREIGN KEY CASCADES USE EXCESSIVE MEMORY
- Bug #20989615: INNODB AUTO_INCREMENT PRODUCES SAME VALUE TWICE

Comparison with Aurora MySQL version 1

The following Amazon Aurora MySQL features are supported in Aurora MySQL Version 1 (compatible with MySQL 5.6), but these features are currently not supported in Aurora MySQL Version 2 (compatible with MySQL 5.7).

- Asynchronous key prefetch (AKP). For more information, see [Optimizing Amazon Aurora indexed join queries with asynchronous key prefetch \(p. 1036\)](#).
- Scan batching. For more information, see [Aurora MySQL database engine updates 2017-12-11 \(p. 1225\)](#).
- Migrating data from MySQL using an Amazon S3 bucket. For more information, see [Migrating data from MySQL by using an Amazon S3 bucket \(p. 784\)](#).

MySQL 5.7 compatibility

Aurora MySQL 2.07.0 is wire-compatible with MySQL 5.7 and includes features such as JSON support, spatial indexes, and generated columns. Aurora MySQL uses a native implementation of spatial indexing using z-order curves to deliver >20x better write performance and >10x better read performance than MySQL 5.7 for spatial datasets.

Aurora MySQL 2.07.0 does not currently support the following MySQL 5.7 features:

- Group replication plugin
- Increased page size
- InnoDB buffer pool loading at startup
- InnoDB full-text parser plugin
- Multisource replication
- Online buffer pool resizing
- Password validation plugin
- Query rewrite plugins
- Replication filtering
- The `CREATE TABLESPACE` SQL statement

Aurora MySQL database engine updates 2019-11-22 (version 2.06.0) (deprecated)

Version: 2.06.0

Aurora MySQL 2.06.0 is generally available. Aurora MySQL 2.x versions are compatible with MySQL 5.7 and Aurora MySQL 1.x versions are compatible with MySQL 5.6.

Currently supported Aurora MySQL releases are 1.14.*, 1.15.*, 1.16.*, 1.17.*, 1.18.*, 1.19.*, 2.01.*, 2.02.*, 2.03.*, 2.04.*, 2.05.*, and 2.06.*.

You can restore a snapshot from a currently supported Aurora MySQL release into Aurora MySQL 2.06.0. You also have the option to upgrade existing Aurora MySQL 2.* database clusters to Aurora MySQL 2.06.0. You cannot upgrade an existing Aurora MySQL 1.* cluster directly to 2.06.0; however, you can restore its snapshot to Aurora MySQL 2.06.0.

To create a cluster with an older version of Aurora MySQL, please specify the engine version through the AWS Management Console, the AWS CLI, or the RDS API.

Note

This version is currently not available in the following AWS Regions: AWS GovCloud (US-East) [us-gov-east-1], AWS GovCloud (US-West) [us-gov-west-1], China (Ningxia) [cn-northwest-1], Asia Pacific (Hong Kong) [ap-east-1], and Middle East (Bahrain) [me-south-1]. There will be a separate announcement once it is made available.

If you have any questions or concerns, AWS Support is available on the community forums and through [AWS Premium Support](#). For more information, see [Maintaining an Amazon Aurora DB cluster \(p. 443\)](#).

Note

For information on how to upgrade your Aurora MySQL database cluster, see [Upgrading the minor version or patch level of an Aurora MySQL DB cluster \(p. 1088\)](#).

Improvements

New features:

- Aurora MySQL clusters now support the instance types db.r5.8xlarge, db.r5.16xlarge, and db.r5.24xlarge. For more information about instance types for Aurora MySQL clusters, see [Aurora DB instance classes \(p. 54\)](#).
- The hash join feature is now generally available and does not require the Aurora lab mode setting to be ON. This feature can improve query performance when you need to join a large amount of data by using an equi-join. For more information about using this feature, see [Optimizing large Aurora MySQL join queries with hash joins \(p. 1038\)](#).

- The hot row contention feature is now generally available and does not require the Aurora lab mode setting to be ON. This feature substantially improves throughput for workloads with many transactions contending for rows on the same page.
- Aurora MySQL 2.06 and higher support "rewinding" a DB cluster to a specific time, without restoring data from a backup. This feature, known as Backtrack, provides a quick way to recover from user errors, such as dropping the wrong table or deleting the wrong row. Backtrack completes within seconds, even for large databases. Read the [AWS blog](#) for an overview, and refer to [Backtracking an Aurora DB cluster \(p. 816\)](#) for more details.
- Aurora 2.06 and higher support synchronous AWS Lambda invocations through the native function `lambda_sync()`. Also available is native function `lambda_async()`, which can be used as an alternative to the existing stored procedure for asynchronous Lambda invocation. For information about calling Lambda functions, see [Invoking a Lambda function from an Amazon Aurora MySQL DB cluster \(p. 1010\)](#).

Critical fixes:

None.

High-priority fixes:

CVE fixes

- [CVE-2019-2805](#)
- [CVE-2019-2730](#)
- [CVE-2019-2739](#)
- [CVE-2019-2778](#)
- [CVE-2019-2758](#)
- [CVE-2018-3064](#)
- [CVE-2018-3058](#)
- [CVE-2018-2786](#)
- [CVE-2017-3653](#)
- [CVE-2017-3455](#)
- [CVE-2017-3465](#)
- [CVE-2017-3244](#)
- [CVE-2016-5612](#)

Connection handling

- Database availability has been improved to better service a surge in client connections while executing one or more DDLs. It is handled by temporarily creating additional threads when needed. You are advised to upgrade if the database becomes unresponsive following a surge in connections while processing DDL.

Engine restart

- Fixed an issue of prolonged unavailability while restarting the engine. This addresses an issue in the buffer pool initialization. This issue occurs rarely but can potentially impact any supported release.
- Fixed an issue that causes a database configured as a binlog master to restart while a heavy write workload is running.

General stability fixes:

- Made improvements where queries accessing uncached data could be slower than usual. Customers experiencing unexplained elevated read latency while accessing uncached data are encouraged to upgrade as they may be experiencing this issue.
- Fixed an issue that failed to restore partitioned tables from a database snapshot. Customers who encounter errors when accessing partitioned tables in a database that has been restored from the snapshot of an Aurora MySQL 1.* database are advised to use this version.
- Improved stability of the Aurora Replicas by fixing lock contention between threads serving read queries and the one applying schema changes while a DDL query is in progress on the writer DB instance.
- Fixed a stability issue related to `mysql.innodb_table_stats` table update triggered by DDL operations.
- Fixed an issue that incorrectly reported `ERROR 1836` when a nested query is executed against a temporary table on the Aurora Replica.

Performance enhancements:

- Improved performance of binlog replication by preventing unnecessary API calls to the cache if the query cache has been disabled on the binlog worker.

Comparison with Aurora MySQL version 1

The following Amazon Aurora MySQL features are supported in Aurora MySQL Version 1 (compatible with MySQL 5.6), but these features are currently not supported in Aurora MySQL Version 2 (compatible with MySQL 5.7).

- Asynchronous key prefetch (AKP). For more information, see [Optimizing Amazon Aurora indexed join queries with asynchronous key prefetch \(p. 1036\)](#).
- Scan batching. For more information, see [Aurora MySQL database engine updates 2017-12-11 \(p. 1225\)](#).
- Migrating data from MySQL using an Amazon S3 bucket. For more information, see [Migrating data from MySQL by using an Amazon S3 bucket \(p. 784\)](#).

MySQL 5.7 compatibility

Aurora MySQL 2.06.0 is wire-compatible with MySQL 5.7 and includes features such as JSON support, spatial indexes, and generated columns. Aurora MySQL uses a native implementation of spatial indexing using z-order curves to deliver >20x better write performance and >10x better read performance than MySQL 5.7 for spatial datasets.

Aurora MySQL 2.06.0 does not currently support the following MySQL 5.7 features:

- Group replication plugin
- Increased page size
- InnoDB buffer pool loading at startup
- InnoDB full-text parser plugin
- Multisource replication
- Online buffer pool resizing
- Password validation plugin
- Query rewrite plugins
- Replication filtering

- The `CREATE TABLESPACE` SQL statement

Aurora MySQL database engine updates 2019-11-11 (version 2.05.0) (deprecated)

Version: 2.05.0

Aurora MySQL 2.05.0 is generally available. Aurora MySQL 2.x versions are compatible with MySQL 5.7 and Aurora MySQL 1.x versions are compatible with MySQL 5.6.

Currently supported Aurora MySQL releases are 1.14.*, 1.15.*, 1.16.*, 1.17.*, 1.18.*, 1.19.*, 2.01.*, 2.02.*, 2.03.* and 2.04.*.

You can restore a snapshot from a currently supported Aurora MySQL release into Aurora MySQL 2.05.0. You also have the option to upgrade existing Aurora MySQL 2.* database clusters, up to 2.04.6, to Aurora MySQL 2.05.0. You cannot upgrade an existing Aurora MySQL 1.* cluster directly to 2.05.0; however, you can restore its snapshot to Aurora MySQL 2.05.0.

To create a cluster with an older version of Aurora MySQL, please specify the engine version through the AWS Management Console, the AWS CLI, or the RDS API.

Note

This version is currently not available in the following AWS Regions: AWS GovCloud (US-East) [us-gov-east-1], AWS GovCloud (US-West) [us-gov-west-1], China (Ningxia) [cn-northwest-1], Asia Pacific (Hong Kong) [ap-east-1], Europe (Stockholm) [eu-north-1], and Middle East (Bahrain) [me-south-1]. There will be a separate announcement once it is made available.

If you have any questions or concerns, AWS Support is available on the community forums and through [AWS Premium Support](#). For more information, see [Maintaining an Amazon Aurora DB cluster \(p. 443\)](#).

Note

For information on how to upgrade your Aurora MySQL database cluster, see [Upgrading the minor version or patch level of an Aurora MySQL DB cluster \(p. 1088\)](#).

Improvements

Critical fixes:

- [CVE-2018-0734](#)
- [CVE-2019-2534](#)
- [CVE-2018-3155](#)
- [CVE-2018-2612](#)
- [CVE-2017-3599](#)
- [CVE-2018-3056](#)
- [CVE-2018-2562](#)
- [CVE-2017-3329](#)
- [CVE-2018-2696](#)
- Fixed an issue where the events in current binlog file on the master were not replicated on the worker if the value of the parameter `sync_binlog` was not set to 1.

High-priority fixes:

- Customers with database size close to 64 tebibytes (TiB) are strongly advised to upgrade to this version to avoid downtime due to stability bugs affecting volumes close to the Aurora storage limit.

- The default value of the parameter `aurora_binlog_replication_max_yield_seconds` has been changed to zero to prevent an increase in replication lag in favor of foreground query performance on the binlog master.

Integration of MySQL bug fixes

- Bug#23054591: PURGE BINARY LOGS TO is reading the whole binlog file and causing MySQL to Stall

Comparison with Aurora MySQL version 1

The following Amazon Aurora MySQL features are supported in Aurora MySQL Version 1 (compatible with MySQL 5.6), but these features are currently not supported in Aurora MySQL Version 2 (compatible with MySQL 5.7).

- Asynchronous key prefetch (AKP). For more information, see [Optimizing Amazon Aurora indexed join queries with asynchronous key prefetch \(p. 1036\)](#).
- Hash joins. For more information, see [Optimizing large Aurora MySQL join queries with hash joins \(p. 1038\)](#).
- Native functions for synchronously invoking AWS Lambda functions. For more information, see [Invoking a Lambda function with an Aurora MySQL native function \(p. 1012\)](#).
- Scan batching. For more information, see [Aurora MySQL database engine updates 2017-12-11 \(p. 1225\)](#).
- Migrating data from MySQL using an Amazon S3 bucket. For more information, see [Migrating data from MySQL by using an Amazon S3 bucket \(p. 784\)](#).

MySQL 5.7 compatibility

Aurora MySQL 2.05.0 is wire-compatible with MySQL 5.7 and includes features such as JSON support, spatial indexes, and generated columns. Aurora MySQL uses a native implementation of spatial indexing using z-order curves to deliver >20x better write performance and >10x better read performance than MySQL 5.7 for spatial datasets.

Aurora MySQL 2.05.0 does not currently support the following MySQL 5.7 features:

- Group replication plugin
- Increased page size
- InnoDB buffer pool loading at startup
- InnoDB full-text parser plugin
- Multisource replication
- Online buffer pool resizing
- Password validation plugin
- Query rewrite plugins
- Replication filtering
- The `CREATE TABLESPACE` SQL statement

Aurora MySQL database engine updates 2020-08-14 (version 2.04.9)

Version: 2.04.9

Aurora MySQL 2.04.9 is generally available. Aurora MySQL 2.x versions are compatible with MySQL 5.7 and Aurora MySQL 1.x versions are compatible with MySQL 5.6.

Currently supported Aurora MySQL releases are 1.14.* , 1.15.* , 1.16.* , 1.17.* , 1.18.* , 1.19.* , 2.01.* , 2.02.* , 2.03.* , 2.04.* , and 2.05.* . You can restore a snapshot of any 2.* Aurora MySQL release into Aurora MySQL 2.04.9. You also have the option to upgrade existing Aurora MySQL 2.* database clusters to Aurora MySQL 2.04.9.

To create a cluster with an older version of Aurora MySQL, please specify the engine version through the AWS Management Console, the AWS CLI, or the RDS API.

If you have any questions or concerns, AWS Support is available on the community forums and through [AWS Premium Support](#). For more information, see [Maintaining an Amazon Aurora DB cluster \(p. 443\)](#).

Note

This version is currently not available in the following AWS Regions: AWS GovCloud (US-East) [us-gov-east-1], AWS GovCloud (US-West) [us-gov-west-1], Asia Pacific (Hong Kong) [ap-east-1], and Middle East (Bahrain) [me-south-1]. There will be a separate announcement once it is made available.

Note

For information on how to upgrade your Aurora MySQL database cluster, see [Upgrading the minor version or patch level of an Aurora MySQL DB cluster \(p. 1088\)](#).

Improvements

High-priority fixes:

CVE fixes

- [CVE-2019-2805](#)
- [CVE-2019-2730](#)
- [CVE-2019-2739](#)
- [CVE-2019-2778](#)
- [CVE-2019-2758](#)
- [CVE-2018-3064](#)
- [CVE-2018-3058](#)
- [CVE-2018-2786](#)
- [CVE-2017-3653](#)
- [CVE-2017-3455](#)
- [CVE-2017-3464](#)
- [CVE-2017-3465](#)
- [CVE-2017-3244](#)
- [CVE-2016-5612](#)
- [CVE-2019-2628](#)
- [CVE-2019-2740](#)
- [CVE-2019-2922](#)
- [CVE-2019-2923](#)
- [CVE-2019-2924](#)
- [CVE-2019-2910](#)
- [CVE-2019-5443](#)
- [CVE-2019-3822](#)

- [CVE-2020-2760](#)
- [CVE-2019-2911](#)
- [CVE-2018-2813](#)

Availability improvements:

- Fixed an issue that could cause a database restart or failover due to execution of a `kill session` command. If you encounter this issue, contact AWS support to enable this fix on your instance.
- Fixed an issue that causes a database restart during execution of a complex query involving multi-table joins and aggregation that use intermediate tables internally.
- Fixed an issue that causes database restarts due to an interrupted `DROP TABLE` on multiple tables.
- Fixed an issue that causes a database failover during database recovery.
- Fixed a database restart caused by incorrect reporting of `threads_running` when audit and slow query logs are enabled.
- Fixed an issue where a `kill query` command might get stuck during execution.
- Fixed a race condition in the lock manager that resulted in a database restart or failover during transaction rollback.
- Fixed an issue that triggered database restart or failover when multiple connections are trying to update the same table with a Full-Text Search index.
- Fixed an issue that can cause a deadlock when purging an index resulting in a failover or restart.

General improvements:

- Fixed issues that could cause queries on read replicas to use data from an uncommitted transaction. This issue is limited to the transactions that are started immediately after a database restart.
- Fixed an issue encountered during `INPLACE ALTER TABLE` for a table with triggers defined and when the DDL did not contain a `RENAME` clause.
- Fixed an issue that caused cloning to take longer on some database clusters with high writeload.
- Fixed an issue encountered during an upgrade when a partitioned table has embedded spaces in the name.
- Fixed an issue where the read replica might transiently see partial results of a recently committed transaction on the writer.
- Fixed an issue where queries on a read replica against an FTS table may produce stale results. This will only occur when the FTS query on the read replica closely follows a query on `INFORMATION_SCHEMA.INNODB_SYS_TABLES` for the same FTS table on the writer.
- Fixed an issue that resulted in a slow restore of Aurora 1.x database cluster containing FTS (Full-Text Search) indexes to an Aurora 2.x database cluster.
- Extended maximum allowable length to 2000 for `server_audit_incl_users` and `server_audit_excl_users` global parameters.
- Fixed an issue where Aurora 1.x to Aurora 2.x restore might take an extended time to complete.
- Fixed an issue where a `lambda_async` invocation through stored procedure doesn't work with Unicode.
- Fixed an issue encountered when a spatial index does not properly handle an off-record geometry column.
- Fixed an issue that might cause a query to fail on a reader DB instance with `InternalFailureException` error with message "Operation terminated (internal error)".

Integration of MySQL bug fixes

- Bug #23070734, Bug #80060: Concurrent TRUNCATE TABLEs cause stalls

- Bug #23103937: PS_TRUNCATE_ALL_TABLES() DOES NOT WORK IN SUPER_READ_ONLY MODE
- Bug#22551677: When taking the server offline, a race condition within the Performance Schema could lead to a server exit.
- Bug #27082268: Invalid FTS sync synchronization.
- BUG #12589870: Fixed an issues which causes a restart with multi-query statement when query cache is enabled.
- Bug #26402045: Certain cases of subquery materialization could cause a server exit. These queries now produce an error suggesting that materialization be disabled.
- Bug #18898433: Queries with many left joins were slow if join buffering was used (for example, using the block nested loop algorithm).
- Bug #25222337: A NULL virtual column field name in a virtual index caused a server exit during a field name comparison that occurs while populating virtual columns affected by a foreign key constraint. (<https://github.com/mysql/mysql-server/commit/273d5c9d7072c63b6c47dbef6963d7dc491d5131>)
- Bug #25053286: Executing a stored procedure containing a query that accessed a view could allocate memory that was not freed until the session ended. (<https://github.com/mysql/mysql-server/commit/d7b37d4d141a95f577916448650c429f0d6e193d>)
- Bug #25586773: Executing a stored procedure containing a statement that created a table from the contents of certain SELECT (<https://dev.mysql.com/doc/refman/5.7/en/select.html>) statements could result in a memory leak. (<https://github.com/mysql/mysql-server/commit/88301e5adab65f6750f66af284be410c4369d0c1>)
- Bug #26666274: INFINITE LOOP IN PERFORMANCE SCHEMA BUFFER CONTAINER.
- Bug #23550835, Bug #23298025, Bug #81464: A SELECT Performance Schema tables when an internal buffer was full could cause a server exit.

Comparison with Aurora MySQL version 1

The following Amazon Aurora MySQL features are supported in Aurora MySQL Version 1 (compatible with MySQL 5.6), but these features are currently not supported in Aurora MySQL Version 2 (compatible with MySQL 5.7).

- Asynchronous key prefetch (AKP). For more information, see [Optimizing Amazon Aurora indexed join queries with asynchronous key prefetch \(p. 1036\)](#).
- Hash joins. For more information, see [Optimizing large Aurora MySQL join queries with hash joins \(p. 1038\)](#).
- Native functions for synchronously invoking AWS Lambda functions. For more information, see [Invoking a Lambda function with an Aurora MySQL native function \(p. 1012\)](#).
- Scan batching. For more information, see [Aurora MySQL database engine updates 2017-12-11 \(p. 1225\)](#).
- Migrating data from MySQL using an Amazon S3 bucket. For more information, see [Migrating data from MySQL by using an Amazon S3 bucket \(p. 784\)](#).

MySQL 5.7 compatibility

Aurora MySQL 2.04.9 is wire-compatible with MySQL 5.7 and includes features such as JSON support, spatial indexes, and generated columns. Aurora MySQL uses a native implementation of spatial indexing using z-order curves to deliver >20x better write performance and >10x better read performance than MySQL 5.7 for spatial datasets.

Aurora MySQL 2.04.9 does not currently support the following MySQL 5.7 features:

- Group replication plugin
- Increased page size

- InnoDB buffer pool loading at startup
- InnoDB full-text parser plugin
- Multisource replication
- Online buffer pool resizing
- Password validation plugin
- Query rewrite plugins
- Replication filtering
- The `CREATE TABLESPACE` SQL statement

Aurora MySQL database engine updates 2019-11-20 (version 2.04.8)

Version: 2.04.8

Aurora MySQL 2.04.8 is generally available. Aurora MySQL 2.x versions are compatible with MySQL 5.7 and Aurora MySQL 1.x versions are compatible with MySQL 5.6.

Currently supported Aurora MySQL releases are 1.14.*, 1.15.*, 1.16.*, 1.17.*, 1.18.*, 1.19.*, 2.01.*, 2.02.*, 2.03.*, 2.04.*, and 2.05.*. You can restore a snapshot of any 2.* Aurora MySQL release into Aurora MySQL 2.04.8. You also have the option to upgrade existing Aurora MySQL 2.* database clusters to Aurora MySQL 2.04.8.

To create a cluster with an older version of Aurora MySQL, please specify the engine version through the AWS Management Console, the AWS CLI, or the RDS API.

If you have any questions or concerns, AWS Support is available on the community forums and through [AWS Premium Support](#). For more information, see [Maintaining an Amazon Aurora DB cluster \(p. 443\)](#).

Note

This version is currently not available in the following AWS Regions: AWS GovCloud (US-East) [us-gov-east-1], AWS GovCloud (US-West) [us-gov-west-1], Asia Pacific (Hong Kong) [ap-east-1], and Middle East (Bahrain) [me-south-1]. There will be a separate announcement once it is made available.

Note

For information on how to upgrade your Aurora MySQL database cluster, see [Upgrading the minor version or patch level of an Aurora MySQL DB cluster \(p. 1088\)](#).

Improvements

New features:

- **Read replica improvements:**

- Reduced network traffic from the writer instance by efficiently transmitting data to reader instances within the Aurora DB cluster. This improvement is enabled by default, because it helps prevent replicas from falling behind and restarting. The parameter for this feature is `aurora_enable_repl_bin_log_filtering`.
- Reduced network traffic from the writer to reader instances within the Aurora DB cluster using compression. This improvement is enabled by default for 8xlarge and 16xlarge instance classes only, because these instances can tolerate additional CPU overhead for compression. The parameter for this feature is `aurora_enable_replica_log_compression`.

High-priority fixes:

- Improved memory management in the Aurora writer instance that prevents restart of writer due to out of memory conditions during heavy workload in presence of reader instances within the Aurora DB cluster.
- Fix for a non-deterministic condition in the scheduler that results in engine restart while accessing the performance schema object concurrently.

Comparison with Aurora MySQL version 1

The following Amazon Aurora MySQL features are supported in Aurora MySQL Version 1 (compatible with MySQL 5.6), but these features are currently not supported in Aurora MySQL Version 2 (compatible with MySQL 5.7).

- Asynchronous key prefetch (AKP). For more information, see [Optimizing Amazon Aurora indexed join queries with asynchronous key prefetch \(p. 1036\)](#).
- Hash joins. For more information, see [Optimizing large Aurora MySQL join queries with hash joins \(p. 1038\)](#).
- Native functions for synchronously invoking AWS Lambda functions. For more information, see [Invoking a Lambda function with an Aurora MySQL native function \(p. 1012\)](#).
- Scan batching. For more information, see [Aurora MySQL database engine updates 2017-12-11 \(p. 1225\)](#).
- Migrating data from MySQL using an Amazon S3 bucket. For more information, see [Migrating data from MySQL by using an Amazon S3 bucket \(p. 784\)](#).

MySQL 5.7 compatibility

Aurora MySQL 2.04.8 is wire-compatible with MySQL 5.7 and includes features such as JSON support, spatial indexes, and generated columns. Aurora MySQL uses a native implementation of spatial indexing using z-order curves to deliver >20x better write performance and >10x better read performance than MySQL 5.7 for spatial datasets.

Aurora MySQL 2.04.8 does not currently support the following MySQL 5.7 features:

- Group replication plugin
- Increased page size
- InnoDB buffer pool loading at startup
- InnoDB full-text parser plugin
- Multisource replication
- Online buffer pool resizing
- Password validation plugin
- Query rewrite plugins
- Replication filtering
- The `CREATE TABLESPACE` SQL statement

Aurora MySQL database engine updates 2019-11-14 (version 2.04.7)

Version: 2.04.7

Aurora MySQL 2.04.7 is generally available. Aurora MySQL 2.x versions are compatible with MySQL 5.7 and Aurora MySQL 1.x versions are compatible with MySQL 5.6.

Currently supported Aurora MySQL releases are 1.14.* , 1.15.* , 1.16.* , 1.17.* , 1.18.* , 1.19.* , 2.01.* , 2.02.* , 2.03.* and 2.04.*.

You can restore a snapshot from a currently supported Aurora MySQL release into Aurora MySQL 2.04.7. You also have the option to upgrade existing Aurora MySQL 2.* database clusters to Aurora MySQL 2.04.7. You can't upgrade an existing Aurora MySQL 1.* cluster directly to 2.04.7; however, you can restore its snapshot to Aurora MySQL 2.04.7.

To create a cluster with an older version of Aurora MySQL, please specify the engine version through the AWS Management Console, the AWS CLI, or the RDS API.

If you have any questions or concerns, AWS Support is available on the community forums and through [AWS Premium Support](#). For more information, see [Maintaining an Amazon Aurora DB cluster \(p. 443\)](#).

Note

This version is currently not available in the following AWS Regions: AWS GovCloud (US-East) [us-gov-east-1], AWS GovCloud (US-West) [us-gov-west-1], Asia Pacific (Hong Kong) [ap-east-1], and Middle East (Bahrain) [me-south-1]. There will be a separate announcement once it is made available.

Note

For information on how to upgrade your Aurora MySQL database cluster, see [Upgrading the minor version or patch level of an Aurora MySQL DB cluster \(p. 1088\)](#).

Improvements

High-priority fixes:

Connection Handling

- Database availability has been improved to better service a surge in client connections while executing one or more DDLs. It is handled by temporarily creating additional threads when needed. You are advised to upgrade if the database becomes unresponsive following a surge in connections while processing DDL.
- Fixed an issue that resulted in an incorrect value for the `Threads_running` global status variable.

Engine Restart

- Fixed an issue of prolonged unavailability while restarting the engine. This addresses an issue in the buffer pool initialization. This issue occurs rarely but can potentially impact any supported release.

General stability fixes:

- Made improvements where queries accessing uncached data could be slower than usual. Customers experiencing unexplained elevated read latencies while accessing uncached data are encouraged to upgrade as they may be experiencing this issue.

Comparison with Aurora MySQL version 1

The following Amazon Aurora MySQL features are supported in Aurora MySQL Version 1 (compatible with MySQL 5.6), but these features are currently not supported in Aurora MySQL Version 2 (compatible with MySQL 5.7).

- Asynchronous key prefetch (AKP). For more information, see [Optimizing Amazon Aurora indexed join queries with asynchronous key prefetch \(p. 1036\)](#).
- Hash joins. For more information, see [Optimizing large Aurora MySQL join queries with hash joins \(p. 1038\)](#).

- Native functions for synchronously invoking AWS Lambda functions. For more information, see [Invoking a Lambda function with an Aurora MySQL native function \(p. 1012\)](#).
- Scan batching. For more information, see [Aurora MySQL database engine updates 2017-12-11 \(p. 1225\)](#).
- Migrating data from MySQL using an Amazon S3 bucket. For more information, see [Migrating data from MySQL by using an Amazon S3 bucket \(p. 784\)](#).

MySQL 5.7 compatibility

Aurora MySQL 2.04.7 is wire-compatible with MySQL 5.7 and includes features such as JSON support, spatial indexes, and generated columns. Aurora MySQL uses a native implementation of spatial indexing using z-order curves to deliver >20x better write performance and >10x better read performance than MySQL 5.7 for spatial datasets.

Aurora MySQL 2.04.7 does not currently support the following MySQL 5.7 features:

- Group replication plugin
- Increased page size
- InnoDB buffer pool loading at startup
- InnoDB full-text parser plugin
- Multisource replication
- Online buffer pool resizing
- Password validation plugin
- Query rewrite plugins
- Replication filtering
- The `CREATE TABLESPACE` SQL statement

Aurora MySQL database engine updates 2019-09-19 (version 2.04.6)

Version: 2.04.6

Aurora MySQL 2.04.6 is generally available. Aurora MySQL 2.x versions are compatible with MySQL 5.7 and Aurora MySQL 1.x versions are compatible with MySQL 5.6.

You have the option to upgrade existing Aurora MySQL 2.* database clusters to Aurora MySQL 2.04.6. We do not allow in-place upgrade of Aurora MySQL 1.* clusters. This restriction will be lifted in a later Aurora MySQL 2.* release. You can restore snapshots of Aurora MySQL 1.14.*, 1.15.*, 1.16.*, 1.17.*, 1.18.*, 1.19.*, 2.01.*, 2.02.*, 2.03.* and 2.04.* into Aurora MySQL 2.04.6.

To use an older version of Aurora MySQL, you can create new database clusters by specifying the engine version through the AWS Management Console, the AWS CLI, or the Amazon RDS API.

If you have any questions or concerns, AWS Support is available on the community forums and through [AWS Premium Support](#). For more information, see [Maintaining an Amazon Aurora DB cluster \(p. 443\)](#).

Note

This version is currently not available in the following AWS Regions: Europe (London) [eu-west-2], AWS GovCloud (US-East) [us-gov-east-1], AWS GovCloud (US-West) [us-gov-west-1], China (Ningxia) [cn-northwest-1], and Asia Pacific (Hong Kong) [ap-east-1]. There will be a separate announcement once it is made available.

Note

For information on how to upgrade your Aurora MySQL database cluster, see [Upgrading the minor version or patch level of an Aurora MySQL DB cluster \(p. 1088\)](#).

Improvements

- Fixed an issue where the events in current binlog file on the master were not replicated on the worker if the value of the parameter `sync_binlog` was not set to 1.
- The default value of the parameter `aurora_binlog_replication_max_yield_seconds` has been changed to zero to prevent an increase in replication lag in favor of foreground query performance on the binlog master.

Integration of MySQL bug fixes

- Bug#23054591: PURGE BINARY LOGS TO is reading the whole binlog file and causing MySql to Stall

Comparison with Aurora MySQL version 1

The following Amazon Aurora MySQL features are supported in Aurora MySQL Version 1 (compatible with MySQL 5.6), but these features are currently not supported in Aurora MySQL Version 2 (compatible with MySQL 5.7).

- Asynchronous key prefetch (AKP). For more information, see [Optimizing Amazon Aurora indexed join queries with asynchronous key prefetch \(p. 1036\)](#).
- Hash joins. For more information, see [Optimizing large Aurora MySQL join queries with hash joins \(p. 1038\)](#).
- Native functions for synchronously invoking AWS Lambda functions. For more information, see [Invoking a Lambda function with an Aurora MySQL native function \(p. 1012\)](#).
- Scan batching. For more information, see [Aurora MySQL database engine updates 2017-12-11 \(p. 1225\)](#).
- Migrating data from MySQL using an Amazon S3 bucket. For more information, see [Migrating data from MySQL by using an Amazon S3 bucket \(p. 784\)](#).

MySQL 5.7 compatibility

Aurora MySQL 2.04.6 is wire-compatible with MySQL 5.7 and includes features such as JSON support, spatial indexes, and generated columns. Aurora MySQL uses a native implementation of spatial indexing using z-order curves to deliver >20x better write performance and >10x better read performance than MySQL 5.7 for spatial datasets.

Aurora MySQL 2.04.6 does not currently support the following MySQL 5.7 features:

- Group replication plugin
- Increased page size
- InnoDB buffer pool loading at startup
- InnoDB full-text parser plugin
- Multisource replication
- Online buffer pool resizing
- Password validation plugin
- Query rewrite plugins
- Replication filtering
- The `CREATE TABLESPACE` SQL statement

Aurora MySQL database engine updates 2019-07-08 (version 2.04.5)

Version: 2.04.5

Aurora MySQL 2.04.5 is generally available. Aurora MySQL 2.x versions are compatible with MySQL 5.7 and Aurora MySQL 1.x versions are compatible with MySQL 5.6.

You have the option to upgrade existing Aurora MySQL 2.* database clusters to Aurora MySQL 2.04.5. We do not allow in-place upgrade of Aurora MySQL 1.* clusters. This restriction will be lifted in a later Aurora MySQL 2.* release. You can restore snapshots of Aurora MySQL 1.14.*, 1.15.*, 1.16.*, 1.17.*, 1.18.*, 1.19.*, 2.01.*, 2.02.*, 2.03.* and 2.04.* into Aurora MySQL 2.04.5.

If you have any questions or concerns, AWS Support is available on the community forums and through [AWS Premium Support](#). For more information, see [Maintaining an Amazon Aurora DB cluster \(p. 443\)](#).

Note

For information on how to upgrade your Aurora MySQL database cluster, see [Upgrading the minor version or patch level of an Aurora MySQL DB cluster \(p. 1088\)](#).

Improvements

- Fixed a race condition during storage volume growth that caused the database to restart.
- Fixed an internal communication failure during volume open that caused the database to restart.
- Added DDL recovery support for `ALTER TABLE ALGORITHM=INPLACE` on partitioned tables.
- Fixed an issue with DDL recovery of `ALTER TABLE ALGORITHM=COPY` that caused the database to restart.
- Improved Aurora Replica stability under heavy delete workload on the writer.
- Fixed a database restart caused by a deadlock between the thread performing full-text search index sync and the thread performing eviction of full-text search table from dictionary cache.
- Fixed a stability issue on the binlog worker during DDL replication while the connection to the binlog master is unstable.
- Fixed an out-of-memory issue in the full-text search code that caused the database to restart.
- Fixed an issue on the Aurora Writer that caused it to restart when the entire 64 tebibyte (TiB) volume is used.
- Fixed a race condition in the Performance Schema feature that caused the database to restart.
- Fixed an issue that caused aborted connections when handling an error in network protocol management.

Comparison with Aurora MySQL version 1

The following Amazon Aurora MySQL features are supported in Aurora MySQL Version 1 (compatible with MySQL 5.6), but these features are currently not supported in Aurora MySQL Version 2 (compatible with MySQL 5.7).

- Asynchronous key prefetch (AKP). For more information, see [Optimizing Amazon Aurora indexed join queries with asynchronous key prefetch \(p. 1036\)](#).
- Hash joins. For more information, see [Optimizing large Aurora MySQL join queries with hash joins \(p. 1038\)](#).
- Native functions for synchronously invoking AWS Lambda functions. For more information, see [Invoking a Lambda function with an Aurora MySQL native function \(p. 1012\)](#).
- Scan batching. For more information, see [Aurora MySQL database engine updates 2017-12-11 \(p. 1225\)](#).

- Migrating data from MySQL using an Amazon S3 bucket. For more information, see [Migrating data from MySQL by using an Amazon S3 bucket \(p. 784\)](#).

MySQL 5.7 compatibility

Aurora MySQL 2.04.5 is wire-compatible with MySQL 5.7 and includes features such as JSON support, spatial indexes, and generated columns. Aurora MySQL uses a native implementation of spatial indexing using z-order curves to deliver >20x better write performance and >10x better read performance than MySQL 5.7 for spatial datasets.

Aurora MySQL 2.04.5 does not currently support the following MySQL 5.7 features:

- Group replication plugin
- Increased page size
- InnoDB buffer pool loading at startup
- InnoDB full-text parser plugin
- Multisource replication
- Online buffer pool resizing
- Password validation plugin
- Query rewrite plugins
- Replication filtering
- The `CREATE TABLESPACE` SQL statement

Aurora MySQL database engine updates 2019-05-29 (version 2.04.4)

Version: 2.04.4

Aurora MySQL 2.04.4 is generally available. Aurora MySQL 2.x versions are compatible with MySQL 5.7 and Aurora MySQL 1.x versions are compatible with MySQL 5.6.

When creating a new Aurora MySQL DB cluster (including restoring a snapshot), you have the option of choosing compatibility with either MySQL 5.7 or MySQL 5.6. We do not allow in-place upgrade of Aurora MySQL 1.* clusters or restore of Aurora MySQL 1.* clusters from an Amazon S3 backup into Aurora MySQL 2.04.4. We plan to remove these restrictions in a later Aurora MySQL 2.* release.

You can restore snapshots of Aurora MySQL 1.14.*, 1.15.*, 1.16.*, 1.17.*, 1.18.*, 1.19.*, 2.01.*, 2.02.*, 2.03.*, and 2.04.* into Aurora MySQL 2.04.4.

If you have any questions or concerns, AWS Support is available on the community forums and through [AWS Premium Support](#). For more information, see [Maintaining an Amazon Aurora DB cluster \(p. 443\)](#).

Note

This version is currently not available in the AWS GovCloud (US-West) [us-gov-west-1], Europe (Stockholm) [eu-north-1], China (Ningxia) [cn-northwest-1], and Asia Pacific (Hong Kong) [ap-east-1] AWS Regions. There will be a separate announcement once it is made available.

Note

For information on how to upgrade your Aurora MySQL database cluster, see [Upgrading the minor version or patch level of an Aurora MySQL DB cluster \(p. 1088\)](#).

Improvements

- Fixed an issue that could cause failures when loading data into Aurora from S3.

- Fixed an issue that could cause failures when upload data from Aurora to S3.
- Fixed an issue that caused aborted connections when handling an error in network protocol management.
- Fixed an issue that could cause a crash when dealing with partitioned tables.
- Fixed an issue with the Performance Insights feature being unavailable in some regions.

Comparison with Aurora MySQL version 1

The following Amazon Aurora MySQL features are supported in Aurora MySQL Version 1 (compatible with MySQL 5.6), but these features are currently not supported in Aurora MySQL Version 2 (compatible with MySQL 5.7).

- Asynchronous key prefetch (AKP). For more information, see [Optimizing Amazon Aurora indexed join queries with asynchronous key prefetch \(p. 1036\)](#).
- Hash joins. For more information, see [Optimizing large Aurora MySQL join queries with hash joins \(p. 1038\)](#).
- Native functions for synchronously invoking AWS Lambda functions. For more information, see [Invoking a Lambda function with an Aurora MySQL native function \(p. 1012\)](#).
- Scan batching. For more information, see [Aurora MySQL database engine updates 2017-12-11 \(p. 1225\)](#).
- Migrating data from MySQL using an Amazon S3 bucket. For more information, see [Migrating data from MySQL by using an Amazon S3 bucket \(p. 784\)](#).

MySQL 5.7 compatibility

Aurora MySQL 2.04.4 is wire-compatible with MySQL 5.7 and includes features such as JSON support, spatial indexes, and generated columns. Aurora MySQL uses a native implementation of spatial indexing using z-order curves to deliver >20x better write performance and >10x better read performance than MySQL 5.7 for spatial datasets.

Aurora MySQL 2.04.4 does not currently support the following MySQL 5.7 features:

- Group replication plugin
- Increased page size
- InnoDB buffer pool loading at startup
- InnoDB full-text parser plugin
- Multisource replication
- Online buffer pool resizing
- Password validation plugin
- Query rewrite plugins
- Replication filtering
- The `CREATE TABLESPACE` SQL statement

Aurora MySQL database engine updates 2019-05-09 (version 2.04.3)

Version: 2.04.3

Aurora MySQL 2.04.3 is generally available. Aurora MySQL 2.x versions are compatible with MySQL 5.7 and Aurora MySQL 1.x versions are compatible with MySQL 5.6.

When creating a new Aurora MySQL DB cluster (including restoring a snapshot), you have the option of choosing compatibility with either MySQL 5.7 or MySQL 5.6. We do not allow in-place upgrade of Aurora MySQL 1.* clusters or restore of Aurora MySQL 1.* clusters from an Amazon S3 backup into Aurora MySQL 2.04.3. We plan to remove these restrictions in a later Aurora MySQL 2.* release.

You can restore snapshots of Aurora MySQL 1.14.*, 1.15.*, 1.16.*, 1.17.*, 1.18.*, 1.19.*, 2.01.*, 2.02.*, 2.03.*, and 2.04.* into Aurora MySQL 2.04.3.

If you have any questions or concerns, AWS Support is available on the community forums and through [AWS Premium Support](#). For more information, see [Maintaining an Amazon Aurora DB cluster \(p. 443\)](#).

Note

This version is currently not available in the AWS GovCloud (US-West) [us-gov-west-1] and China (Ningxia) [cn-northwest-1] AWS Regions. There will be a separate announcement once it is made available.

Note

For information on how to upgrade your Aurora MySQL database cluster, see [Upgrading the minor version or patch level of an Aurora MySQL DB cluster \(p. 1088\)](#).

Improvements

- Fixed a bug in binlog replication that can cause an issue on Aurora instances configured as binlog worker.
- Fixed an out-of-memory condition when handling large stored routines.
- Fixed an error in handling certain kinds of `ALTER TABLE` commands.
- Fixed an issue with aborted connections because of an error in network protocol management.

Comparison with Aurora MySQL version 1

The following Amazon Aurora MySQL features are supported in Aurora MySQL Version 1 (compatible with MySQL 5.6), but these features are currently not supported in Aurora MySQL Version 2 (compatible with MySQL 5.7).

- Asynchronous key prefetch (AKP). For more information, see [Optimizing Amazon Aurora indexed join queries with asynchronous key prefetch \(p. 1036\)](#).
- Hash joins. For more information, see [Optimizing large Aurora MySQL join queries with hash joins \(p. 1038\)](#).
- Native functions for synchronously invoking AWS Lambda functions. For more information, see [Invoking a Lambda function with an Aurora MySQL native function \(p. 1012\)](#).
- Scan batching. For more information, see [Aurora MySQL database engine updates 2017-12-11 \(p. 1225\)](#).
- Migrating data from MySQL using an Amazon S3 bucket. For more information, see [Migrating data from MySQL by using an Amazon S3 bucket \(p. 784\)](#).

MySQL 5.7 compatibility

Aurora MySQL 2.04.3 is wire-compatible with MySQL 5.7 and includes features such as JSON support, spatial indexes, and generated columns. Aurora MySQL uses a native implementation of spatial indexing using z-order curves to deliver >20x better write performance and >10x better read performance than MySQL 5.7 for spatial datasets.

Aurora MySQL 2.04.3 does not currently support the following MySQL 5.7 features:

- Group replication plugin

- Increased page size
- InnoDB buffer pool loading at startup
- InnoDB full-text parser plugin
- Multisource replication
- Online buffer pool resizing
- Password validation plugin
- Query rewrite plugins
- Replication filtering
- The `CREATE TABLESPACE` SQL statement

Aurora MySQL database engine updates 2019-05-02 (version 2.04.2)

Version: 2.04.2

Aurora MySQL 2.04.2 is generally available. Aurora MySQL 2.x versions are compatible with MySQL 5.7 and Aurora MySQL 1.x versions are compatible with MySQL 5.6.

When creating a new Aurora MySQL DB cluster (including restoring a snapshot), you have the option of choosing compatibility with either MySQL 5.7 or MySQL 5.6. We do not allow in-place upgrade of Aurora MySQL 1.* clusters or restore of Aurora MySQL 1.* clusters from an Amazon S3 backup into Aurora MySQL 2.04.2. We plan to remove these restrictions in a later Aurora MySQL 2.* release.

You can restore snapshots of Aurora MySQL 1.14.*, 1.15.*, 1.16.*, 1.17.*, 1.18.*, 1.19.*, 2.01.*, 2.02.*, 2.03.*, 2.04.0, and 2.04.1 into Aurora MySQL 2.04.2.

If you have any questions or concerns, AWS Support is available on the community forums and through [AWS Premium Support](#). For more information, see [Maintaining an Amazon Aurora DB cluster \(p. 443\)](#).

Note

This version is currently not available in the AWS GovCloud (US-West) [us-gov-west-1] and China (Ningxia) [cn-northwest-1] AWS Regions. There will be a separate announcement once it is made available.

Note

For information on how to upgrade your Aurora MySQL database cluster, see [Upgrading the minor version or patch level of an Aurora MySQL DB cluster \(p. 1088\)](#).

Improvements

- Added support for SSL binlog replication using custom certificates. For information on using SSL binlog replication in Aurora MySQL, see [mysql_rds_import_binlog_ssl_material](#).
- Fixed a deadlock on the Aurora primary instance that occurs when a table with a Full Text Search index is being optimized.
- Fixed an issue on the Aurora Replicas where performance of certain queries using `SELECT(*)` could be impacted on tables that have secondary indexes.
- Fixed a condition that resulted in Error 1032 being posted.
- Improved the stability of Aurora Replicas by fixing multiple deadlocks.

Integration of MySQL bug fixes

- Bug #24829050 - INDEX_MERGE_INTERSECTION OPTIMIZATION CAUSES WRONG QUERY RESULTS

Comparison with Aurora MySQL version 1

The following Amazon Aurora MySQL features are supported in Aurora MySQL Version 1 (compatible with MySQL 5.6), but these features are currently not supported in Aurora MySQL Version 2 (compatible with MySQL 5.7).

- Asynchronous key prefetch (AKP). For more information, see [Optimizing Amazon Aurora indexed join queries with asynchronous key prefetch \(p. 1036\)](#).
- Hash joins. For more information, see [Optimizing large Aurora MySQL join queries with hash joins \(p. 1038\)](#).
- Native functions for synchronously invoking AWS Lambda functions. For more information, see [Invoking a Lambda function with an Aurora MySQL native function \(p. 1012\)](#).
- Scan batching. For more information, see [Aurora MySQL database engine updates 2017-12-11 \(p. 1225\)](#).
- Migrating data from MySQL using an Amazon S3 bucket. For more information, see [Migrating data from MySQL by using an Amazon S3 bucket \(p. 784\)](#).

MySQL 5.7 compatibility

Aurora MySQL 2.04.2 is wire-compatible with MySQL 5.7 and includes features such as JSON support, spatial indexes, and generated columns. Aurora MySQL uses a native implementation of spatial indexing using z-order curves to deliver >20x better write performance and >10x better read performance than MySQL 5.7 for spatial datasets.

Aurora MySQL 2.04.2 does not currently support the following MySQL 5.7 features:

- Group replication plugin
- Increased page size
- InnoDB buffer pool loading at startup
- InnoDB full-text parser plugin
- Multisource replication
- Online buffer pool resizing
- Password validation plugin
- Query rewrite plugins
- Replication filtering
- The `CREATE TABLESPACE` SQL statement

Aurora MySQL database engine updates 2019-03-25 (version 2.04.1)

Version: 2.04.1

Aurora MySQL 2.04.1 is generally available. Aurora MySQL 2.x versions are compatible with MySQL 5.7 and Aurora MySQL 1.x versions are compatible with MySQL 5.6.

When creating a new Aurora MySQL DB cluster (including restoring a snapshot), you have the option of choosing compatibility with either MySQL 5.7 or MySQL 5.6. We do not allow in-place upgrade of Aurora MySQL 1.* clusters or restore of Aurora MySQL 1.* clusters from an Amazon S3 backup into Aurora MySQL 2.04.1. We plan to remove these restrictions in a later Aurora MySQL 2.* release.

You can restore snapshots of Aurora MySQL 1.14.*, 1.15.*, 1.16.*, 1.17.*, 1.18.*, 1.19.*, 2.01.*, 2.02.*, 2.03.*, 2.04.0 into Aurora MySQL 2.04.1.

If you have any questions or concerns, AWS Support is available on the community forums and through [AWS Premium Support](#). For more information, see [Maintaining an Amazon Aurora DB cluster \(p. 443\)](#).

Note

This version is currently not available in the AWS GovCloud (US-West) [us-gov-west-1] region. There will be a separate announcement once it is made available.

Note

The procedure to upgrade your DB cluster has changed. For more information, see [Upgrading the minor version or patch level of an Aurora MySQL DB cluster \(p. 1088\)](#).

Improvements

- Fixed an issue where an Aurora MySQL 5.6 snapshot for versions lower than 1.16 could not be restored to the latest Aurora MySQL 5.7 cluster.

Comparison with Aurora MySQL version 1

The following Amazon Aurora MySQL features are supported in Aurora MySQL Version 1 (compatible with MySQL 5.6), but these features are currently not supported in Aurora MySQL Version 2 (compatible with MySQL 5.7).

- Asynchronous key prefetch (AKP). For more information, see [Optimizing Amazon Aurora indexed join queries with asynchronous key prefetch \(p. 1036\)](#).
- Hash joins. For more information, see [Optimizing large Aurora MySQL join queries with hash joins \(p. 1038\)](#).
- Native functions for synchronously invoking AWS Lambda functions. For more information, see [Invoking a Lambda function with an Aurora MySQL native function \(p. 1012\)](#).
- Scan batching. For more information, see [Aurora MySQL database engine updates 2017-12-11 \(p. 1225\)](#).
- Migrating data from MySQL using an Amazon S3 bucket. For more information, see [Migrating data from MySQL by using an Amazon S3 bucket \(p. 784\)](#).

MySQL 5.7 compatibility

This Aurora MySQL version is wire-compatible with MySQL 5.7 and includes features such as JSON support, spatial indexes, and generated columns. Aurora MySQL uses a native implementation of spatial indexing using z-order curves to deliver >20x better write performance and >10x better read performance than MySQL 5.7 for spatial datasets.

This Aurora MySQL version does not currently support the following MySQL 5.7 features:

- Group replication plugin
- Increased page size
- InnoDB buffer pool loading at startup
- InnoDB full-text parser plugin
- Multisource replication
- Online buffer pool resizing
- Password validation plugin
- Query rewrite plugins
- Replication filtering
- The `CREATE TABLESPACE` SQL statement

Aurora MySQL database engine updates 2019-03-25 (version 2.04.0)

Version: 2.04

Aurora MySQL 2.04 is generally available. Aurora MySQL 2.x versions are compatible with MySQL 5.7 and Aurora MySQL 1.x versions are compatible with MySQL 5.6.

When creating a new Aurora MySQL DB cluster (including restoring a snapshot), you have the option of choosing compatibility with either MySQL 5.7 or MySQL 5.6. We do not allow in-place upgrade of Aurora MySQL 1.* clusters or restore of Aurora MySQL 1.* clusters from an Amazon S3 backup into Aurora MySQL 2.04.0. We plan to remove these restrictions in a later Aurora MySQL 2.* release.

You can restore snapshots of Aurora MySQL 1.19.*, 2.01.*, 2.02.*, and 2.03.* into Aurora MySQL 2.04.0. You cannot restore snapshots of Aurora MySQL 1.14.* or lower, 1.15.*, 1.16.*, 1.17.*, 1.18.* into Aurora MySQL 2.04.0. This restriction is removed in Aurora MySQL 2.04.1.

If you have any questions or concerns, AWS Support is available on the community forums and through [AWS Premium Support](#). For more information, see [Maintaining an Amazon Aurora DB cluster \(p. 443\)](#).

Note

This version is currently not available in the AWS GovCloud (US-West) [us-gov-west-1] region. There will be a separate announcement once it is made available.

Note

The procedure to upgrade your DB cluster has changed. For more information, see [Upgrading the minor version or patch level of an Aurora MySQL DB cluster \(p. 1088\)](#).

Improvements

- Supports GTID-based replication. For information about using GTID-based replication with Aurora MySQL, see [Using GTID-based replication for Aurora MySQL \(p. 954\)](#).
- Fixed an issue where an Aurora Replica incorrectly throws a running in read-only mode error when a statement deleting or updating rows in a temporary table contains an InnoDB subquery.

Integration of MySQL bug fixes

- Bug #26225783: MYSQL CRASH ON CREATE TABLE (REPRODUCABLE) -> INNODB: ALONG SEMAPHORE WAIT.

Comparison with Aurora MySQL version 1

The following Amazon Aurora MySQL features are supported in Aurora MySQL Version 1 (compatible with MySQL 5.6), but these features are currently not supported in Aurora MySQL Version 2 (compatible with MySQL 5.7).

- Asynchronous key prefetch (AKP). For more information, see [Optimizing Amazon Aurora indexed join queries with asynchronous key prefetch \(p. 1036\)](#).
- Hash joins. For more information, see [Optimizing large Aurora MySQL join queries with hash joins \(p. 1038\)](#).
- Native functions for synchronously invoking AWS Lambda functions. For more information, see [Invoking a Lambda function with an Aurora MySQL native function \(p. 1012\)](#).
- Scan batching. For more information, see [Aurora MySQL database engine updates 2017-12-11 \(p. 1225\)](#).

- Migrating data from MySQL using an Amazon S3 bucket. For more information, see [Migrating data from MySQL by using an Amazon S3 bucket \(p. 784\)](#).

MySQL 5.7 compatibility

This Aurora MySQL version is wire-compatible with MySQL 5.7 and includes features such as JSON support, spatial indexes, and generated columns. Aurora MySQL uses a native implementation of spatial indexing using z-order curves to deliver >20x better write performance and >10x better read performance than MySQL 5.7 for spatial datasets.

This Aurora MySQL version does not currently support the following MySQL 5.7 features:

- Group replication plugin
- Increased page size
- InnoDB buffer pool loading at startup
- InnoDB full-text parser plugin
- Multisource replication
- Online buffer pool resizing
- Password validation plugin
- Query rewrite plugins
- Replication filtering
- The `CREATE TABLESPACE` SQL statement

Aurora MySQL database engine updates 2019-02-07 (version 2.03.4) (deprecated)

Version: 2.03.4

Aurora MySQL 2.03.4 is generally available. Aurora MySQL 2.x versions are compatible with MySQL 5.7 and Aurora MySQL 1.x versions are compatible with MySQL 5.6.

When creating a new Aurora MySQL DB cluster (including restoring a snapshot), you can choose compatibility with either MySQL 5.7 or MySQL 5.6.

We don't allow in-place upgrade of Aurora MySQL 1.* clusters into Aurora MySQL 2.03.4 or restore to Aurora MySQL 2.03.4 from an Amazon S3 backup. We plan to remove these restrictions in a later Aurora MySQL 2.* release.

If you have any questions or concerns, AWS Support is available on the community forums and through [AWS Premium Support](#). For more information, see [Maintaining an Amazon Aurora DB cluster \(p. 443\)](#).

Note

This version is currently not available in the AWS GovCloud (US-West) [us-gov-west-1] and China (Beijing) [cn-north-1] regions. There will be a separate announcement once it is made available.

Note

The procedure to upgrade your DB cluster has changed. For more information, see [Upgrading the minor version or patch level of an Aurora MySQL DB cluster \(p. 1088\)](#).

Improvements

- Support for UTF8MB4 Unicode 9.0 accent-sensitive and case-insensitive collation, `utf8mb4_0900_as_ci`.

Comparison with Aurora MySQL version 1

The following Amazon Aurora MySQL features are supported in Aurora MySQL version 1 (compatible with MySQL 5.6), but these features are currently not supported in Aurora MySQL version 2 (compatible with MySQL 5.7).

- Asynchronous key prefetch (AKP). For more information, see [Optimizing Amazon Aurora indexed join queries with asynchronous key prefetch \(p. 1036\)](#).
- Hash joins. For more information, see [Optimizing large Aurora MySQL join queries with hash joins \(p. 1038\)](#).
- Native functions for synchronously invoking AWS Lambda functions. These functions are available for MySQL 5.7-compatible clusters in Aurora MySQL 2.06 and higher. For more information, see [Invoking a Lambda function with an Aurora MySQL native function \(p. 1012\)](#).
- Scan batching. For more information, see [Aurora MySQL database engine updates 2017-12-11 \(p. 1225\)](#).
- Migrating data from MySQL using an Amazon S3 bucket. For more information, see [Migrating data from MySQL by using an Amazon S3 bucket \(p. 784\)](#).

MySQL 5.7 compatibility

Aurora MySQL 2.03.4 is wire-compatible with MySQL 5.7 and includes features such as JSON support, spatial indexes, and generated columns. Aurora MySQL uses a native implementation of spatial indexing using z-order curves to deliver >20x better write performance and >10x better read performance than MySQL 5.7 for spatial datasets.

Aurora MySQL 2.03.4 does not currently support the following MySQL 5.7 features:

- Global transaction identifiers (GTIDs). Aurora MySQL supports GTIDs in version 2.04 and higher.
- Group replication plugin
- Increased page size
- InnoDB buffer pool loading at startup
- InnoDB full-text parser plugin
- Multisource replication
- Online buffer pool resizing
- Password validation plugin
- Query rewrite plugins
- Replication filtering
- The CREATE TABLESPACE SQL statement
- X Protocol

Aurora MySQL database engine updates 2019-01-18 (version 2.03.3) (deprecated)

Version: 2.03.3

Aurora MySQL 2.03.3 is generally available. Aurora MySQL 2.x versions are compatible with MySQL 5.7 and Aurora MySQL 1.x versions are compatible with MySQL 5.6.

When creating a new Aurora MySQL DB cluster (including restoring a snapshot), you can choose compatibility with either MySQL 5.7 or MySQL 5.6.

We don't allow in-place upgrade of Aurora MySQL 1.* clusters into Aurora MySQL 2.03.3 or restore to Aurora MySQL 2.03.3 from an Amazon S3 backup. We plan to remove these restrictions in a later Aurora MySQL 2.* release.

If you have any questions or concerns, AWS Support is available on the community forums and through [AWS Premium Support](#). For more information, see [Maintaining an Amazon Aurora DB cluster \(p. 443\)](#).

Note

This version is currently not available in the AWS GovCloud (US-West) [us-gov-west-1] and China (Beijing) [cn-north-1] regions. There will be a separate announcement once it is made available.

Note

The procedure to upgrade your DB cluster has changed. For more information, see [Upgrading the minor version or patch level of an Aurora MySQL DB cluster \(p. 1088\)](#).

Improvements

- Fixed an issue where an Aurora Replica might become dead-latched when running a backward scan on an index.
- Fixed an issue where an Aurora Replica might restart when the Aurora primary instance runs in-place DDL operations on partitioned tables.
- Fixed an issue where an Aurora Replica might restart during query cache invalidation after a DDL operation on the Aurora primary instance.
- Fixed an issue where an Aurora Replica might restart during a `SELECT` query on a table while the Aurora primary instance runs truncation on that table.
- Fixed a wrong result issue with MyISAM temporary tables where only indexed columns are accessed.
- Fixed an issue in slow logs that generated incorrect large values for `query_time` and `lock_time` periodically after approximately 40,000 queries.
- Fixed an issue where a schema named "tmp" could cause migration from RDS for MySQL to Aurora MySQL to become stuck.
- Fixed an issue where the audit log might have missing events during log rotation.
- Fixed an issue where the Aurora primary instance restored from an Aurora 5.6 snapshot might restart when the Fast DDL feature in the lab mode is enabled.
- Fixed an issue where the CPU usage is 100% caused by the dictionary stats thread.
- Fixed an issue where an Aurora Replica might restart when running a `CHECK TABLE` statement.

Integration of MySQL bug fixes

- Bug #25361251: INCORRECT BEHAVIOR WITH INSERT ON DUPLICATE KEY IN SP
- Bug #26734162: INCORRECT BEHAVIOR WITH INSERT OF BLOB + ON DUPLICATE KEY UPDATE
- Bug #27460607: INCORRECT BEHAVIOR OF IODKU WHEN INSERT SELECT's SOURCE TABLE IS EMPTY
- A query using a `DISTINCT` or `GROUP BY` clause could return incorrect results. (MySQL 5.7 Bug #79591, Bug #22343910)
- A `DELETE` from joined tables using a derived table in the `WHERE` clause fails with error 1093 (Bug #23074801).
- GCOLS: INCORRECT BEHAVIOR WITH CHARSET CHANGES (Bug #25287633).

Comparison with Aurora MySQL version 1

The following Amazon Aurora MySQL features are supported in Aurora MySQL version 1 (compatible with MySQL 5.6), but these features are currently not supported in Aurora MySQL version 2 (compatible with MySQL 5.7).

- Asynchronous key prefetch (AKP). For more information, see [Optimizing Amazon Aurora indexed join queries with asynchronous key prefetch \(p. 1036\)](#).
- Hash joins. For more information, see [Optimizing large Aurora MySQL join queries with hash joins \(p. 1038\)](#).
- Native functions for synchronously invoking AWS Lambda functions. These functions are available for MySQL 5.7-compatible clusters in Aurora MySQL 2.06 and higher. For more information, see [Invoking a Lambda function with an Aurora MySQL native function \(p. 1012\)](#).
- Scan batching. For more information, see [Aurora MySQL database engine updates 2017-12-11 \(p. 1225\)](#).
- Migrating data from MySQL using an Amazon S3 bucket. For more information, see [Migrating data from MySQL by using an Amazon S3 bucket \(p. 784\)](#).

MySQL 5.7 compatibility

Aurora MySQL 2.03.3 is wire-compatible with MySQL 5.7 and includes features such as JSON support, spatial indexes, and generated columns. Aurora MySQL uses a native implementation of spatial indexing using z-order curves to deliver >20x better write performance and >10x better read performance than MySQL 5.7 for spatial datasets.

Aurora MySQL 2.03.3 does not currently support the following MySQL 5.7 features:

- Global transaction identifiers (GTIDs). Aurora MySQL supports GTIDs in version 2.04 and higher.
- Group replication plugin
- Increased page size
- InnoDB buffer pool loading at startup
- InnoDB full-text parser plugin
- Multisource replication
- Online buffer pool resizing
- Password validation plugin
- Query rewrite plugins
- Replication filtering
- The `CREATE TABLESPACE` SQL statement
- X Protocol

Aurora MySQL database engine updates 2019-01-09 (version 2.03.2) (deprecated)

Version: 2.03.2

Aurora MySQL 2.03.2 is generally available. Aurora MySQL 2.x versions are compatible with MySQL 5.7 and Aurora MySQL 1.x versions are compatible with MySQL 5.6.

When creating a new Aurora MySQL DB cluster (including restoring a snapshot), you can choose compatibility with either MySQL 5.7 or MySQL 5.6.

We don't allow in-place upgrade of Aurora MySQL 1.* clusters into Aurora MySQL 2.03.2 or restore to Aurora MySQL 2.03.2 from an Amazon S3 backup. We plan to remove these restrictions in a later Aurora MySQL 2.* release.

If you have any questions or concerns, AWS Support is available on the community forums and through [AWS Premium Support](#). For more information, see [Maintaining an Amazon Aurora DB cluster \(p. 443\)](#).

Note

This version is currently not available in the AWS GovCloud (US-West) [us-gov-west-1] and China (Beijing) [cn-north-1] regions. There will be a separate announcement once it is made available.

Note

The procedure to upgrade your DB cluster has changed. For more information, see [Upgrading the minor version or patch level of an Aurora MySQL DB cluster \(p. 1088\)](#).

Improvements

- **Aurora Version Selector** – Starting with Aurora MySQL 2.03.2, you can choose from among multiple versions of MySQL 5.7-compatible Aurora on the AWS Management Console. For more information, see [Checking or specifying Aurora MySQL engine versions through AWS \(p. 1083\)](#).

Comparison with Aurora MySQL version 1

The following Amazon Aurora MySQL features are supported in Aurora MySQL version 1 (compatible with MySQL 5.6), but these features are currently not supported in Aurora MySQL version 2 (compatible with MySQL 5.7).

- Asynchronous key prefetch (AKP). For more information, see [Optimizing Amazon Aurora indexed join queries with asynchronous key prefetch \(p. 1036\)](#).
- Hash joins. For more information, see [Optimizing large Aurora MySQL join queries with hash joins \(p. 1038\)](#).
- Native functions for synchronously invoking AWS Lambda functions. These functions are available for MySQL 5.7-compatible clusters in Aurora MySQL 2.06 and higher. For more information, see [Invoking a Lambda function with an Aurora MySQL native function \(p. 1012\)](#).
- Scan batching. For more information, see [Aurora MySQL database engine updates 2017-12-11 \(p. 1225\)](#).
- Migrating data from MySQL using an Amazon S3 bucket. For more information, see [Migrating data from MySQL by using an Amazon S3 bucket \(p. 784\)](#).

MySQL 5.7 compatibility

Aurora MySQL 2.03.2 is wire-compatible with MySQL 5.7 and includes features such as JSON support, spatial indexes, and generated columns. Aurora MySQL uses a native implementation of spatial indexing using z-order curves to deliver >20x better write performance and >10x better read performance than MySQL 5.7 for spatial datasets.

Aurora MySQL 2.03.2 does not currently support the following MySQL 5.7 features:

- Global transaction identifiers (GTIDs). Aurora MySQL supports GTIDs in version 2.04 and higher.
- Group replication plugin
- Increased page size
- InnoDB buffer pool loading at startup
- InnoDB full-text parser plugin
- Multisource replication
- Online buffer pool resizing
- Password validation plugin
- Query rewrite plugins
- Replication filtering
- The `CREATE TABLESPACE` SQL statement

- X Protocol

Aurora MySQL database engine updates 2018-10-24 (version 2.03.1) (deprecated)

Version: 2.03.1

Aurora MySQL 2.03.1 is generally available. Aurora MySQL 2.x versions are compatible with MySQL 5.7 and Aurora MySQL 1.x versions are compatible with MySQL 5.6.

When creating a new Aurora MySQL DB cluster, you can choose compatibility with either MySQL 5.7 or MySQL 5.6. When restoring a MySQL 5.6-compatible snapshot, you can choose compatibility with either MySQL 5.7 or MySQL 5.6.

You can restore snapshots of Aurora MySQL 1.14.*, 1.15.*, 1.16.*, 1.17.*, 1.18.*, 2.01.*, 2.02.*, and 2.03 into Aurora MySQL 2.03.1.

We don't allow in-place upgrade of Aurora MySQL 1.* clusters into Aurora MySQL 2.03.1 or restore to Aurora MySQL 2.03.1 from an Amazon S3 backup. We plan to remove these restrictions in a later Aurora MySQL 2.* release.

If you have any questions or concerns, AWS Support is available on the community forums and through [AWS Premium Support](#). For more information, see [Maintaining an Amazon Aurora DB cluster \(p. 443\)](#).

Note

This version is currently not available in the AWS GovCloud (US-West) [us-gov-west-1] and China (Beijing) [cn-north-1] regions. There will be a separate announcement once it is made available.

Improvements

- Fix an issue where the Aurora Writer might restart when running transaction deadlock detection.

Comparison with Aurora MySQL version 1

The following Amazon Aurora MySQL features are supported in Aurora MySQL version 1 (compatible with MySQL 5.6), but these features are currently not supported in Aurora MySQL version 2 (compatible with MySQL 5.7).

- Asynchronous key prefetch (AKP). For more information, see [Optimizing Amazon Aurora indexed join queries with asynchronous key prefetch \(p. 1036\)](#).
- Hash joins. For more information, see [Optimizing large Aurora MySQL join queries with hash joins \(p. 1038\)](#).
- Native functions for synchronously invoking AWS Lambda functions. These functions are available for MySQL 5.7-compatible clusters in Aurora MySQL 2.06 and higher. For more information, see [Invoking a Lambda function with an Aurora MySQL native function \(p. 1012\)](#).
- Scan batching. For more information, see [Aurora MySQL database engine updates 2017-12-11 \(p. 1225\)](#).
- Migrating data from MySQL using an Amazon S3 bucket. For more information, see [Migrating data from MySQL by using an Amazon S3 bucket \(p. 784\)](#).

MySQL 5.7 compatibility

Aurora MySQL 2.03.1 is wire-compatible with MySQL 5.7 and includes features such as JSON support, spatial indexes, and generated columns. Aurora MySQL uses a native implementation of spatial indexing

using z-order curves to deliver >20x better write performance and >10x better read performance than MySQL 5.7 for spatial datasets.

Aurora MySQL 2.03.1 does not currently support the following MySQL 5.7 features:

- Global transaction identifiers (GTIDs). Aurora MySQL supports GTIDs in version 2.04 and higher.
- Group replication plugin
- Increased page size
- InnoDB buffer pool loading at startup
- InnoDB full-text parser plugin
- Multisource replication
- Online buffer pool resizing
- Password validation plugin
- Query rewrite plugins
- Replication filtering
- The `CREATE TABLESPACE` SQL statement
- X Protocol

Aurora MySQL database engine updates 2018-10-11 (version 2.03) (deprecated)

Version: 2.03

Aurora MySQL 2.03 is generally available. Aurora MySQL 2.x versions are compatible with MySQL 5.7 and Aurora MySQL 1.x versions are compatible with MySQL 5.6.

When creating a new Aurora MySQL DB cluster, you can choose compatibility with either MySQL 5.7 or MySQL 5.6. When restoring a MySQL 5.6-compatible snapshot, you can choose compatibility with either MySQL 5.7 or MySQL 5.6.

You can restore snapshots of Aurora MySQL 1.14.* , 1.15.* , 1.16.* , 1.17.* , 1.18.* , 2.01.* , and 2.02.* into Aurora MySQL 2.03.

We don't allow in-place upgrade of Aurora MySQL 1.* clusters into Aurora MySQL 2.03 or restore to Aurora MySQL 2.03 from an Amazon S3 backup. We plan to remove these restrictions in a later Aurora MySQL 2.* release.

Note

This version is currently not available in the AWS GovCloud (US-West) [us-gov-west-1] and China (Beijing) [cn-north-1] regions. There will be a separate announcement once it is made available.

If you have any questions or concerns, AWS Support is available on the community forums and through [AWS Premium Support](#). For more information, see [Maintaining an Amazon Aurora DB cluster \(p. 443\)](#).

Improvements

- Performance schema is available.
- Fixed an issue where zombie sessions with killed state might consume more CPU.
- Fixed a dead latch issue when a read-only transaction is acquiring a lock on a record on the Aurora Writer.
- Fixed an issue where the Aurora Replica without customer workload might have high CPU utilization.

- Multiple fixes on issues that might cause the Aurora Replica or the Aurora writer to restart.
- Added capability to skip diagnostic logging when the disk throughput limit is reached.
- Fixed a memory leak issue when binlog is enabled on the Aurora Writer.

Integration of MySQL community edition bug fixes

- REVERSE SCAN ON A PARTITIONED TABLE DOES ICP - ORDER BY DESC (Bug #24929748).
- JSON_OBJECT CREATES INVALID JSON CODE (Bug#26867509).
- INSERTING LARGE JSON DATA TAKES AN INORDINATE AMOUNT OF TIME (Bug #22843444).
- PARTITIONED TABLES USE MORE MEMORY IN 5.7 THAN 5.6 (Bug #25080442).

Comparison with Aurora MySQL version 1

The following Amazon Aurora MySQL features are supported in Aurora MySQL version 1 (compatible with MySQL 5.6), but these features are currently not supported in Aurora MySQL version 2 (compatible with MySQL 5.7).

- Asynchronous key prefetch (AKP). For more information, see [Optimizing Amazon Aurora indexed join queries with asynchronous key prefetch \(p. 1036\)](#).
- Hash joins. For more information, see [Optimizing large Aurora MySQL join queries with hash joins \(p. 1038\)](#).
- Native functions for synchronously invoking AWS Lambda functions. These functions are available for MySQL 5.7-compatible clusters in Aurora MySQL 2.06 and higher. For more information, see [Invoking a Lambda function with an Aurora MySQL native function \(p. 1012\)](#).
- Scan batching. For more information, see [Aurora MySQL database engine updates 2017-12-11 \(p. 1225\)](#).
- Migrating data from MySQL using an Amazon S3 bucket. For more information, see [Migrating data from MySQL by using an Amazon S3 bucket \(p. 784\)](#).

MySQL 5.7 compatibility

Aurora MySQL 2.03 is wire-compatible with MySQL 5.7 and includes features such as JSON support, spatial indexes, and generated columns. Aurora MySQL uses a native implementation of spatial indexing using z-order curves to deliver >20x better write performance and >10x better read performance than MySQL 5.7 for spatial datasets.

Aurora MySQL 2.03 does not currently support the following MySQL 5.7 features:

- Global transaction identifiers (GTIDs). Aurora MySQL supports GTIDs in version 2.04 and higher.
- Group replication plugin
- Increased page size
- InnoDB buffer pool loading at startup
- InnoDB full-text parser plugin
- Multisource replication
- Online buffer pool resizing
- Password validation plugin
- Query rewrite plugins
- Replication filtering
- The CREATE TABLESPACE SQL statement

- X Protocol

Aurora MySQL database engine updates 2018-10-08 (version 2.02.5) (deprecated)

Version: 2.02.5

Aurora MySQL 2.02.5 is generally available. Aurora MySQL 2.x versions are compatible with MySQL 5.7 and Aurora MySQL 1.x versions are compatible with MySQL 5.6.

When creating a new Aurora MySQL DB cluster, you can choose compatibility with either MySQL 5.7 or MySQL 5.6. When restoring a MySQL 5.6-compatible snapshot, you can choose compatibility with either MySQL 5.7 or MySQL 5.6.

You can restore snapshots of Aurora MySQL 1.14.* , 1.15.* , 1.16.* , 1.17.* , 1.18.* , 2.01.* , and 2.02.* into Aurora MySQL 2.02.5. You can also perform an in-place upgrade from Aurora MySQL 2.01.* or 2.02.* to Aurora MySQL 2.02.5.

We don't allow in-place upgrade of Aurora MySQL 1.* clusters into Aurora MySQL 2.02.5 or restore to Aurora MySQL 2.02.5 from an Amazon S3 backup. We plan to remove these restrictions in a later Aurora MySQL 2.* release.

The performance schema is disabled for this release of Aurora MySQL 5.7. Upgrade to Aurora 2.03 for performance schema support.

Note

This version is currently not available in the AWS GovCloud (US-West) [us-gov-west-1] and China (Beijing) [cn-north-1] regions. There will be a separate announcement once it is made available.

If you have any questions or concerns, AWS Support is available on the community forums and through [AWS Premium Support](#). For more information, see [Maintaining an Amazon Aurora DB cluster \(p. 443\)](#).

Improvements

- Fix an issue where an Aurora Replica might restart when it is doing a reverse scan on a table.

Comparison with Aurora MySQL version 1

The following Amazon Aurora MySQL features are supported in Aurora MySQL version 1 (compatible with MySQL 5.6), but these features are currently not supported in Aurora MySQL version 2 (compatible with MySQL 5.7).

- Asynchronous key prefetch (AKP). For more information, see [Optimizing Amazon Aurora indexed join queries with asynchronous key prefetch \(p. 1036\)](#).
- Hash joins. For more information, see [Optimizing large Aurora MySQL join queries with hash joins \(p. 1038\)](#).
- Native functions for synchronously invoking AWS Lambda functions. These functions are available for MySQL 5.7-compatible clusters in Aurora MySQL 2.06 and higher. For more information, see [Invoking a Lambda function with an Aurora MySQL native function \(p. 1012\)](#).
- Scan batching. For more information, see [Aurora MySQL database engine updates 2017-12-11 \(p. 1225\)](#).
- Migrating data from MySQL using an Amazon S3 bucket. For more information, see [Migrating data from MySQL by using an Amazon S3 bucket \(p. 784\)](#).

MySQL 5.7 compatibility

Aurora MySQL 2.02.5 is wire-compatible with MySQL 5.7 and includes features such as JSON support, spatial indexes, and generated columns. Aurora MySQL uses a native implementation of spatial indexing using z-order curves to deliver >20x better write performance and >10x better read performance than MySQL 5.7 for spatial datasets.

Aurora MySQL 2.02.5 does not currently support the following MySQL 5.7 features:

- Global transaction identifiers (GTIDs). Aurora MySQL supports GTIDs in version 2.04 and higher.
- Group replication plugin
- Increased page size
- InnoDB buffer pool loading at startup
- InnoDB full-text parser plugin
- Multisource replication
- Online buffer pool resizing
- Password validation plugin
- Query rewrite plugins
- Replication filtering
- The `CREATE TABLESPACE` SQL statement
- X Protocol

Aurora MySQL database engine updates 2018-09-21 (version 2.02.4) (deprecated)

Version: 2.02.4

Aurora MySQL 2.02.4 is generally available. Aurora MySQL 2.x versions are compatible with MySQL 5.7 and Aurora MySQL 1.x versions are compatible with MySQL 5.6.

When creating a new Aurora MySQL DB cluster, you can choose compatibility with either MySQL 5.7 or MySQL 5.6. When restoring a MySQL 5.6-compatible snapshot, you can choose compatibility with either MySQL 5.7 or MySQL 5.6.

You can restore snapshots of Aurora MySQL 1.14.* , 1.15.* , 1.16.* , 1.17.* , 1.18.* , 2.01.* , and 2.02.* into Aurora MySQL 2.02.4. You can also perform an in-place upgrade from Aurora MySQL 2.01.* or 2.02.* to Aurora MySQL 2.02.4.

We don't allow in-place upgrade of Aurora MySQL 1.* clusters into Aurora MySQL 2.02.4 or restore to Aurora MySQL 2.02.4 from an Amazon S3 backup. We plan to remove these restrictions in a later Aurora MySQL 2.* release.

The performance schema is disabled for this release of Aurora MySQL 5.7. Upgrade to Aurora 2.03 for performance schema support.

If you have any questions or concerns, AWS Support is available on the community forums and through [AWS Premium Support](#). For more information, see [Maintaining an Amazon Aurora DB cluster \(p. 443\)](#).

Improvements

- Fixed a stability issue related to Full Text Search indexes on tables restored from an Aurora MySQL 5.6 snapshot.

Integration of MySQL community edition bug fixes

- BUG#13651665 INNODB MAY BE UNABLE TO LOAD TABLE DEFINITION AFTER RENAME
- BUG#21371070 INNODB: CANNOT ALLOCATE 0 BYTES.
- BUG#21378944 FTS ASSERT ENC.SRC_ILIST_PTR != NULL, FTS_OPTIMIZE_WORD(), OPTIMIZE TABLE
- BUG#21508537 ASSERTION FAILURE UT_A(!VICTIM_TRX->READ_ONLY)
- BUG#21983865 UNEXPECTED DEADLOCK WITH INNODB_AUTOINC_LOCK_MODE=0
- BUG#22679185 INVALID INNODB FTS DOC ID DURING INSERT
- BUG#22899305 GCOLS: ASSERTION: !(COL->PRTYPE & 256).
- BUG#22956469 MEMORY LEAK INTRODUCED IN 5.7.8 IN MEMORY/INNODB/OSOFILE
- BUG#22996488 CRASH IN FTS_SYNC_INDEX WHEN DOING DDL IN A LOOP
- BUG#23014521 GCOL:INNODB: ASSERTION: !IS_V
- BUG#23021168 REPLICATION STOPS AFTER TRX IS ROLLED BACK ASYNC
- BUG#23052231 ASSERTION: ADD_AUTOINC < DICT_TABLE_GET_N_USER_COLS
- BUG#23149683 ROTATE INNODB MASTER KEY WITH KEYRING_OKV_CONF_DIR MISSING: SIGSEGV; SIGNAL 11
- BUG#23762382 INSERT VALUES QUERY WITH JOIN IN A SELECT CAUSES INCORRECT BEHAVIOR
- BUG#25209512 CURRENT_TIMESTAMP PRODUCES ZEROS IN TRIGGER
- BUG#26626277 BUG IN "INSERT... ON DUPLICATE KEY UPDATE" QUERY
- BUG#26734162 INCORRECT BEHAVIOR WITH INSERT OF BLOB + ON DUPLICATE KEY UPDATE
- BUG#27460607 INCORRECT WHEN INSERT SELECT's SOURCE TABLE IS EMPTY

Comparison with Aurora MySQL version 1

The following Amazon Aurora MySQL features are supported in Aurora MySQL version 1 (compatible with MySQL 5.6), but these features are currently not supported in Aurora MySQL version 2 (compatible with MySQL 5.7).

- Asynchronous key prefetch (AKP). For more information, see [Optimizing Amazon Aurora indexed join queries with asynchronous key prefetch \(p. 1036\)](#).
- Hash joins. For more information, see [Optimizing large Aurora MySQL join queries with hash joins \(p. 1038\)](#).
- Native functions for synchronously invoking AWS Lambda functions. These functions are available for MySQL 5.7-compatible clusters in Aurora MySQL 2.06 and higher. For more information, see [Invoking a Lambda function with an Aurora MySQL native function \(p. 1012\)](#).
- Scan batching. For more information, see [Aurora MySQL database engine updates 2017-12-11 \(p. 1225\)](#).
- Migrating data from MySQL using an Amazon S3 bucket. For more information, see [Migrating data from MySQL by using an Amazon S3 bucket \(p. 784\)](#).

MySQL 5.7 compatibility

Aurora MySQL 2.02.4 is wire-compatible with MySQL 5.7 and includes features such as JSON support, spatial indexes, and generated columns. Aurora MySQL uses a native implementation of spatial indexing using z-order curves to deliver >20x better write performance and >10x better read performance than MySQL 5.7 for spatial datasets.

Aurora MySQL 2.02.4 does not currently support the following MySQL 5.7 features:

- Global transaction identifiers (GTIDs). Aurora MySQL supports GTIDs in version 2.04 and higher.
- Group replication plugin
- Increased page size
- InnoDB buffer pool loading at startup
- InnoDB full-text parser plugin
- Multisource replication
- Online buffer pool resizing
- Password validation plugin
- Query rewrite plugins
- Replication filtering
- The `CREATE TABLESPACE` SQL statement
- X Protocol

Aurora MySQL database engine updates 2018-08-23 (version 2.02.3) (deprecated)

Version: 2.02.3

Aurora MySQL 2.02.3 is generally available. Aurora MySQL 2.x versions are compatible with MySQL 5.7 and Aurora MySQL 1.x versions are compatible with MySQL 5.6.

When creating a new Aurora MySQL DB cluster, you can choose compatibility with either MySQL 5.7 or MySQL 5.6. When restoring a MySQL 5.6-compatible snapshot, you can choose compatibility with either MySQL 5.7 or MySQL 5.6.

You can restore snapshots of Aurora MySQL 1.14.* , 1.15.* , 1.16.* , 1.17.* , 2.01.* , and 2.02.* into Aurora MySQL 2.02.3. You can also perform an in-place upgrade from Aurora MySQL 2.01.* or 2.02.* to Aurora MySQL 2.02.3.

We don't allow in-place upgrade of Aurora MySQL 1.* clusters into Aurora MySQL 2.02.3 or restore to Aurora MySQL 2.02.3 from an Amazon S3 backup. We plan to remove these restrictions in a later Aurora MySQL 2.* release.

The performance schema is disabled for this release of Aurora MySQL 5.7. Upgrade to Aurora 2.03 for performance schema support.

Note

This version is currently not available in the AWS GovCloud (US-West) [us-gov-west-1] and China (Beijing) [cn-north-1] regions. There will be a separate announcement once it is made available.

If you have any questions or concerns, AWS Support is available on the community forums and through [AWS Premium Support](#). For more information, see [Maintaining an Amazon Aurora DB cluster \(p. 443\)](#).

Comparison with Aurora MySQL version 1

The following Amazon Aurora MySQL features are supported in Aurora MySQL version 1 (compatible with MySQL 5.6), but these features are currently not supported in Aurora MySQL version 2 (compatible with MySQL 5.7).

- Asynchronous key prefetch (AKP). For more information, see [Optimizing Amazon Aurora indexed join queries with asynchronous key prefetch \(p. 1036\)](#).

- Hash joins. For more information, see [Optimizing large Aurora MySQL join queries with hash joins \(p. 1038\)](#).
- Native functions for synchronously invoking AWS Lambda functions. These functions are available for MySQL 5.7-compatible clusters in Aurora MySQL 2.06 and higher. For more information, see [Invoking a Lambda function with an Aurora MySQL native function \(p. 1012\)](#).
- Scan batching. For more information, see [Aurora MySQL database engine updates 2017-12-11 \(p. 1225\)](#).
- Migrating data from MySQL using an Amazon S3 bucket. For more information, see [Migrating data from MySQL by using an Amazon S3 bucket \(p. 784\)](#).

Currently, Aurora MySQL 2.01 does not support features added in Aurora MySQL version 1.16 and later. For information about Aurora MySQL version 1.16, see [Aurora MySQL database engine updates 2017-12-11 \(p. 1225\)](#).

MySQL 5.7 compatibility

Aurora MySQL 2.02.3 is wire-compatible with MySQL 5.7 and includes features such as JSON support, spatial indexes, and generated columns. Aurora MySQL uses a native implementation of spatial indexing using z-order curves to deliver >20x better write performance and >10x better read performance than MySQL 5.7 for spatial datasets.

Aurora MySQL 2.02.3 does not currently support the following MySQL 5.7 features:

- Global transaction identifiers (GTIDs). Aurora MySQL supports GTIDs in version 2.04 and higher.
- Group replication plugin
- Increased page size
- InnoDB buffer pool loading at startup
- InnoDB full-text parser plugin
- Multisource replication
- Online buffer pool resizing
- Password validation plugin
- Query rewrite plugins
- Replication filtering
- The `CREATE TABLESPACE` SQL statement
- X Protocol

CLI differences between Aurora MySQL 2.x and Aurora MySQL 1.x

- The engine name for Aurora MySQL 2.x is `aurora-mysql`; the engine name for Aurora MySQL 1.x continues to be `aurora`.
- The engine version for Aurora MySQL 2.x is `5.7.12`; the engine version for Aurora MySQL 1.x continues to be `5.6.10ann`.
- The default parameter group for Aurora MySQL 2.x is `default.aurora-mysql5.7`; the default parameter group for Aurora MySQL 1.x continues to be `default.aurora5.6`.
- The DB cluster parameter group family name for Aurora MySQL 2.x is `aurora-mysql5.7`; the DB cluster parameter group family name for Aurora MySQL 1.x continues to be `aurora5.6`.

Refer to the Aurora documentation for the full set of CLI commands and differences between Aurora MySQL 2.x and Aurora MySQL 1.x.

Improvements

- Fixed an issue where an Aurora Replica can restart when using optimistic cursor restores while reading records.
- Updated the default value of the parameter `innodb_stats_persistent_sample_pages` to 128 to improve index statistics.
- Fixed an issue where an Aurora Replica might restart when it accesses a small table that is being concurrently modified on the Aurora primary instance.
- Fixed `ANALYZE TABLE` to stop flushing the table definition cache.
- Fixed an issue where the Aurora primary instance or an Aurora Replica might restart when converting a point query for geospatial to a search range.

Aurora MySQL database engine updates 2018-06-04 (version 2.02.2) (deprecated)

Version: 2.02.2

Aurora MySQL 2.02.2 is generally available. Aurora MySQL 2.x versions are compatible with MySQL 5.7 and Aurora MySQL 1.x versions are compatible with MySQL 5.6.

When creating a new Aurora MySQL DB cluster, you can choose compatibility with either MySQL 5.7 or MySQL 5.6. When restoring a MySQL 5.6-compatible snapshot, you can choose compatibility with either MySQL 5.7 or MySQL 5.6.

You can restore snapshots of Aurora MySQL 1.14*, 1.15*, 1.16*, 1.17*, 2.01*, and 2.02 into Aurora MySQL 2.02.2. You can also perform an in-place upgrade from Aurora MySQL 2.01* or 2.02 to Aurora MySQL 2.02.2.

We don't allow in-place upgrade of Aurora MySQL 1.* clusters into Aurora MySQL 2.02.2 or restore to Aurora MySQL 2.02.2 from an Amazon S3 backup. We plan to remove these restrictions in a later Aurora MySQL 2.* release.

The performance schema is disabled for this release of Aurora MySQL 5.7. Upgrade to Aurora 2.03 for performance schema support.

Note

This version is currently not available in the AWS GovCloud (US-West) [us-gov-west-1] and China (Beijing) [cn-north-1] regions. There will be a separate announcement once it is made available.

If you have any questions or concerns, AWS Support is available on the community forums and through [AWS Premium Support](#). For more information, see [Maintaining an Amazon Aurora DB cluster \(p. 443\)](#).

Comparison with Aurora MySQL 5.6

The following Amazon Aurora MySQL features are supported in Aurora MySQL 5.6, but these features are currently not supported in Aurora MySQL 5.7.

- Asynchronous key prefetch (AKP). For more information, see [Optimizing Amazon Aurora indexed join queries with asynchronous key prefetch \(p. 1036\)](#).
- Hash joins. For more information, see [Optimizing large Aurora MySQL join queries with hash joins \(p. 1038\)](#).
- Native functions for synchronously invoking AWS Lambda functions. For more information, see [Invoking a Lambda function with an Aurora MySQL native function \(p. 1012\)](#).
- Scan batching. For more information, see [Aurora MySQL database engine updates 2017-12-11 \(p. 1225\)](#).

- Migrating data from MySQL using an Amazon S3 bucket. For more information, see [Migrating data from MySQL by using an Amazon S3 bucket \(p. 784\)](#).

Currently, Aurora MySQL 2.01 does not support features added in Aurora MySQL version 1.16 and later. For information about Aurora MySQL version 1.16, see [Aurora MySQL database engine updates 2017-12-11 \(p. 1225\)](#).

MySQL 5.7 compatibility

Aurora MySQL 2.02.2 is wire-compatible with MySQL 5.7 and includes features such as JSON support, spatial indexes, and generated columns. Aurora MySQL uses a native implementation of spatial indexing using z-order curves to deliver >20x better write performance and >10x better read performance than MySQL 5.7 for spatial datasets.

Aurora MySQL 2.02.2 does not currently support the following MySQL 5.7 features:

- Global transaction identifiers (GTIDs). Aurora MySQL supports GTIDs in version 2.04 and higher.
- Group replication plugin
- Increased page size
- InnoDB buffer pool loading at startup
- InnoDB full-text parser plugin
- Multisource replication
- Online buffer pool resizing
- Password validation plugin
- Query rewrite plugins
- Replication filtering
- The CREATE TABLESPACE SQL statement
- X Protocol

CLI differences between Aurora MySQL 2.x and Aurora MySQL 1.x

- The engine name for Aurora MySQL 2.x is `aurora-mysql`; the engine name for Aurora MySQL 1.x continues to be `aurora`.
- The engine version for Aurora MySQL 2.x is `5.7.12`; the engine version for Aurora MySQL 1.x continues to be `5.6.10ann`.
- The default parameter group for Aurora MySQL 2.x is `default.aurora-mysql5.7`; the default parameter group for Aurora MySQL 1.x continues to be `default.aurora5.6`.
- The DB cluster parameter group family name for Aurora MySQL 2.x is `aurora-mysql5.7`; the DB cluster parameter group family name for Aurora MySQL 1.x continues to be `aurora5.6`.

Refer to the Aurora documentation for the full set of CLI commands and differences between Aurora MySQL 2.x and Aurora MySQL 1.x.

Improvements

- Fixed an issue where an Aurora Writer can occasionally restart when tracking Aurora Replica progress.
- Fixed an issue where an Aurora Replica restarts or throws an error when a partitioned table is accessed after running index create or drop statements on the table on the Aurora Writer.
- Fixed an issue where a table on an Aurora Replica is inaccessible while it is applying the changes caused by running ALTER table ADD/DROP column statements on the Aurora Writer.

Aurora MySQL database engine updates 2018-05-03 (version 2.02) (deprecated)

Version: 2.02

Aurora MySQL 2.02 is generally available. Aurora MySQL 2.x versions are compatible with MySQL 5.7 and Aurora MySQL 1.x versions are compatible with MySQL 5.6.

When creating a new Aurora MySQL DB cluster, you can choose compatibility with either MySQL 5.7 or MySQL 5.6. When restoring a MySQL 5.6-compatible snapshot, you can choose compatibility with either MySQL 5.7 or MySQL 5.6.

You can restore snapshots of Aurora MySQL 1.14*, 1.15*, 1.16*, 1.17* and 2.01* into Aurora MySQL 2.02. You can also perform an in-place upgrade from Aurora MySQL 2.01* to Aurora MySQL 2.02.

We don't allow in-place upgrade of Aurora MySQL 1.x clusters into Aurora MySQL 2.02 or restore to Aurora MySQL 2.02 from an Amazon S3 backup. We plan to remove these restrictions in a later Aurora MySQL 2.x release.

The performance schema is disabled for this release of Aurora MySQL 5.7. Upgrade to Aurora 2.03 for performance schema support.

If you have any questions or concerns, AWS Support is available on the community forums and through [AWS Premium Support](#). For more information, see [Maintaining an Amazon Aurora DB cluster \(p. 443\)](#).

Comparison with Aurora MySQL 5.6

The following Amazon Aurora MySQL features are supported in Aurora MySQL 5.6, but these features are currently not supported in Aurora MySQL 5.7.

- Asynchronous key prefetch (AKP). For more information, see [Optimizing Amazon Aurora indexed join queries with asynchronous key prefetch \(p. 1036\)](#).
- Hash joins. For more information, see [Optimizing large Aurora MySQL join queries with hash joins \(p. 1038\)](#).
- Native functions for synchronously invoking AWS Lambda functions. For more information, see [Invoking a Lambda function with an Aurora MySQL native function \(p. 1012\)](#).
- Scan batching. For more information, see [Aurora MySQL database engine updates 2017-12-11 \(p. 1225\)](#).
- Migrating data from MySQL using an Amazon S3 bucket. For more information, see [Migrating data from MySQL by using an Amazon S3 bucket \(p. 784\)](#).

Currently, Aurora MySQL 2.01 does not support features added in Aurora MySQL version 1.16 and later. For information about Aurora MySQL version 1.16, see [Aurora MySQL database engine updates 2017-12-11 \(p. 1225\)](#).

MySQL 5.7 compatibility

Aurora MySQL 2.02 is wire-compatible with MySQL 5.7 and includes features such as JSON support, spatial indexes, and generated columns. Aurora MySQL uses a native implementation of spatial indexing using z-order curves to deliver >20x better write performance and >10x better read performance than MySQL 5.7 for spatial datasets.

Aurora MySQL 2.02 does not currently support the following MySQL 5.7 features:

- Global transaction identifiers (GTIDs). Aurora MySQL supports GTIDs in version 2.04 and higher.

- Group replication plugin
- Increased page size
- InnoDB buffer pool loading at startup
- InnoDB full-text parser plugin
- Multisource replication
- Online buffer pool resizing
- Password validation plugin
- Query rewrite plugins
- Replication filtering
- The `CREATE TABLESPACE` SQL statement
- X Protocol

CLI differences between Aurora MySQL 2.x and Aurora MySQL 1.x

- The engine name for Aurora MySQL 2.x is `aurora-mysql`; the engine name for Aurora MySQL 1.x continues to be `aurora`.
- The engine version for Aurora MySQL 2.x is `5.7.12`; the engine version for Aurora MySQL 1.x continues to be `5.6.10ann`.
- The default parameter group for Aurora MySQL 2.x is `default.aurora-mysql5.7`; the default parameter group for Aurora MySQL 1.x continues to be `default.aurora5.6`.
- The DB cluster parameter group family name for Aurora MySQL 2.x is `aurora-mysql5.7`; the DB cluster parameter group family name for Aurora MySQL 1.x continues to be `aurora5.6`.

Refer to the Aurora documentation for the full set of CLI commands and differences between Aurora MySQL 2.x and Aurora MySQL 1.x.

Improvements

- Fixed an issue where an Aurora Writer restarts when running `INSERT` statements and exploiting the Fast Insert optimization.
- Fixed an issue where an Aurora Replica restarts when running `ALTER DATABASE` statements on the Aurora Replica.
- Fixed an issue where an Aurora Replica restarts when running queries on tables that have just been dropped on the Aurora Writer.
- Fixed an issue where an Aurora Replica restarts when setting `innodb_adaptive_hash_index` to OFF on the Aurora Replica.
- Fixed an issue where an Aurora Replica restarts when running `TRUNCATE TABLE` queries on the Aurora Writer.
- Fixed an issue where the Aurora Writer freezes in certain circumstances when running `INSERT` statements. On a multi-node cluster, this can result in a failover.
- Fixed a memory leak associated with setting session variables.
- Fixed an issue where the Aurora Writer freezes in certain circumstances associated with purging undo for tables with generated columns.
- Fixed an issue where the Aurora Writer can sometimes restart when binary logging is enabled.

Integration of MySQL bug fixes

- Left join returns incorrect results on the outer side (Bug #22833364).

Aurora MySQL database engine updates 2018-03-13 (version 2.01.1) (deprecated)

Version: 2.01.1

Aurora MySQL 2.01.1 is generally available. Aurora MySQL 2.x versions are compatible with MySQL 5.7 and Aurora MySQL 1.x versions are compatible with MySQL 5.6.

When creating a new Aurora MySQL DB cluster, you can choose compatibility with either MySQL 5.7 or MySQL 5.6. When restoring a MySQL 5.6-compatible snapshot, you can choose compatibility with either MySQL 5.7 or MySQL 5.6.

You can restore snapshots of Aurora MySQL 1.14*, 1.15*, 1.16*, and 1.17* into Aurora MySQL 2.01.1.

We don't allow in-place upgrade of Aurora MySQL 1.x clusters into Aurora MySQL 2.01.1 or restore to Aurora MySQL 2.01.1 from an Amazon S3 backup. We plan to remove these restrictions in a later Aurora MySQL 2.x release.

The performance schema is disabled for this release of Aurora MySQL 5.7. Upgrade to Aurora 2.03 for performance schema support.

Comparison with Aurora MySQL 5.6

The following Amazon Aurora MySQL features are supported in Aurora MySQL 5.6, but these features are currently not supported in Aurora MySQL 5.7.

- Asynchronous key prefetch (AKP). For more information, see [Optimizing Amazon Aurora indexed join queries with asynchronous key prefetch \(p. 1036\)](#).
- Hash joins. For more information, see [Optimizing large Aurora MySQL join queries with hash joins \(p. 1038\)](#).
- Native functions for synchronously invoking AWS Lambda functions. For more information, see [Invoking a Lambda function with an Aurora MySQL native function \(p. 1012\)](#).
- Scan batching. For more information, see [Aurora MySQL database engine updates 2017-12-11 \(p. 1225\)](#).
- Migrating data from MySQL using an Amazon S3 bucket. For more information, see [Migrating data from MySQL by using an Amazon S3 bucket \(p. 784\)](#).

Currently, Aurora MySQL 2.01.1 does not support features added in Aurora MySQL version 1.16 and later. For information about Aurora MySQL version 1.16, see [Aurora MySQL database engine updates 2017-12-11 \(p. 1225\)](#).

MySQL 5.7 compatibility

Aurora MySQL 2.01.1 is wire-compatible with MySQL 5.7 and includes features such as JSON support, spatial indexes, and generated columns. Aurora MySQL uses a native implementation of spatial indexing using z-order curves to deliver >20x better write performance and >10x better read performance than MySQL 5.7 for spatial datasets.

Aurora MySQL 2.01.1 does not currently support the following MySQL 5.7 features:

- Global transaction identifiers (GTIDs). Aurora MySQL supports GTIDs in version 2.04 and higher.
- Group replication plugin
- Increased page size
- InnoDB buffer pool loading at startup
- InnoDB full-text parser plugin

- Multisource replication
- Online buffer pool resizing
- Password validation plugin
- Query rewrite plugins
- Replication filtering
- The `CREATE TABLESPACE` SQL statement
- X Protocol

CLI differences between Aurora MySQL 2.x and Aurora MySQL 1.x

- The engine name for Aurora MySQL 2.x is `aurora-mysql`; the engine name for Aurora MySQL 1.x continues to be `aurora`.
- The engine version for Aurora MySQL 2.x is `5.7.12`; the engine version for Aurora MySQL 1.x continues to be `5.6.10ann`.
- The default parameter group for Aurora MySQL 2.x is `default.aurora-mysql5.7`; the default parameter group for Aurora MySQL 1.x continues to be `default.aurora5.6`.
- The DB cluster parameter group family name for Aurora MySQL 2.x is `aurora-mysql5.7`; the DB cluster parameter group family name for Aurora MySQL 1.x continues to be `aurora5.6`.

Refer to the Aurora documentation for the full set of CLI commands and differences between Aurora MySQL 2.x and Aurora MySQL 1.x.

Improvements

- Fixed an issue with snapshot restore where Aurora-specific database privileges were created incorrectly when a MySQL 5.6-compatible snapshot was restored with MySQL 5.7 compatibility.
- Added support for 1.17 snapshot restores.

Aurora MySQL database engine updates 2018-02-06 (version 2.01) (deprecated)

Version: 2.01

Aurora MySQL 2.01 is generally available. Going forward, Aurora MySQL 2.x versions will be compatible with MySQL 5.7 and Aurora MySQL 1.x versions will be compatible with MySQL 5.6.

When creating a new Aurora MySQL DB cluster, including those restored from snapshots, you can choose compatibility with either MySQL 5.7 or MySQL 5.6.

You can restore snapshots of Aurora MySQL 1.14*, 1.15*, and 1.16* into Aurora MySQL 2.01.

We don't allow in-place upgrade of Aurora MySQL 1.x clusters into Aurora MySQL 2.01 or restore to Aurora MySQL 2.01 from an Amazon S3 backup. We plan to remove these restrictions in a later Aurora MySQL 2.x release.

The performance schema is disabled for this release of Aurora MySQL 5.7. Upgrade to Aurora 2.03 for performance schema support.

Comparison with Aurora MySQL 5.6

The following Amazon Aurora MySQL features are supported in Aurora MySQL 5.6, but these features are currently not supported in Aurora MySQL 5.7.

- Asynchronous key prefetch (AKP). For more information, see [Optimizing Amazon Aurora indexed join queries with asynchronous key prefetch \(p. 1036\)](#).
- Hash joins. For more information, see [Optimizing large Aurora MySQL join queries with hash joins \(p. 1038\)](#).
- Native functions for synchronously invoking AWS Lambda functions. For more information, see [Invoking a Lambda function with an Aurora MySQL native function \(p. 1012\)](#).
- Scan batching. For more information, see [Aurora MySQL database engine updates 2017-12-11 \(p. 1225\)](#).
- Migrating data from MySQL using an Amazon S3 bucket. For more information, see [Migrating data from MySQL by using an Amazon S3 bucket \(p. 784\)](#).

Currently, Aurora MySQL 2.01 does not support features added in Aurora MySQL version 1.16 and later. For information about Aurora MySQL version 1.16, see [Aurora MySQL database engine updates 2017-12-11 \(p. 1225\)](#).

MySQL 5.7 compatibility

Aurora MySQL 2.01 is wire-compatible with MySQL 5.7 and includes features such as JSON support, spatial indexes, and generated columns. Aurora MySQL uses a native implementation of spatial indexing using z-order curves to deliver >20x better write performance and >10x better read performance than MySQL 5.7 for spatial datasets.

Aurora MySQL 2.01 does not currently support the following MySQL 5.7 features:

- Global transaction identifiers (GTIDs). Aurora MySQL supports GTIDs in version 2.04 and higher.
- Group replication plugin
- Increased page size
- InnoDB buffer pool loading at startup
- InnoDB full-text parser plugin
- Multisource replication
- Online buffer pool resizing
- Password validation plugin
- Query rewrite plugins
- Replication filtering
- The `CREATE TABLESPACE` SQL statement
- X Protocol

CLI differences between Aurora MySQL 2.x and Aurora MySQL 1.x

- The engine name for Aurora MySQL 2.x is `aurora-mysql`; the engine name for Aurora MySQL 1.x continues to be `aurora`.
- The engine version for Aurora MySQL 2.x is `5.7.12`; the engine version for Aurora MySQL 1.x continues to be `5.6.10ann`.
- The default parameter group for Aurora MySQL 2.x is `default.aurora-mysql5.7`; the default parameter group for Aurora MySQL 1.x continues to be `default.aurora5.6`.
- The DB cluster parameter group family name for Aurora MySQL 2.x is `aurora-mysql5.7`; the DB cluster parameter group family name for Aurora MySQL 1.x continues to be `aurora5.6`.

Refer to the Aurora documentation for the full set of CLI commands and differences between Aurora MySQL 2.x and Aurora MySQL 1.x.

Database engine updates for Amazon Aurora MySQL version 1

The following are Amazon Aurora version 1 database engine updates:

- [Aurora MySQL database engine updates 2021-09-30 \(version 1.23.4\) \(p. 1197\)](#)
- [Aurora MySQL database engine updates 2021-06-28 \(version 1.23.3\) \(p. 1198\)](#)
- [Aurora MySQL database engine updates 2021-03-18 \(version 1.23.2\) \(p. 1198\)](#)
- [Aurora MySQL database engine updates 2020-11-24 \(version 1.23.1\) \(p. 1199\)](#)
- [Aurora MySQL database engine updates 2020-09-02 \(version 1.23.0\) \(p. 1200\)](#)
- [Aurora MySQL database engine updates 2021-06-03 \(version 1.22.5\) \(p. 1204\)](#)
- [Aurora MySQL database engine updates 2021-03-04 \(version 1.22.4\) \(p. 1204\)](#)
- [Aurora MySQL database engine updates 2020-11-09 \(version 1.22.3\) \(p. 1205\)](#)
- [Aurora MySQL database engine updates 2020-03-05 \(version 1.22.2\) \(p. 1206\)](#)
- [Aurora MySQL database engine updates 2019-12-23 \(version 1.22.1\) \(p. 1207\)](#)
- [Aurora MySQL database engine updates 2019-11-25 \(version 1.22.0\) \(p. 1208\)](#)
- [Aurora MySQL database engine updates 2019-11-25 \(version 1.21.0\) \(p. 1211\)](#)
- [Aurora MySQL database engine updates 2020-03-05 \(version 1.20.1\) \(p. 1212\)](#)
- [Aurora MySQL database engine updates 2019-11-11 \(version 1.20.0\) \(p. 1213\)](#)
- [Aurora MySQL database engine updates 2020-03-05 \(version 1.19.6\) \(p. 1214\)](#)
- [Aurora MySQL database engine updates 2019-09-19 \(version 1.19.5\) \(p. 1214\)](#)
- [Aurora MySQL database engine updates 2019-06-05 \(version 1.19.2\) \(p. 1215\)](#)
- [Aurora MySQL database engine updates 2019-05-09 \(version 1.19.1\) \(p. 1216\)](#)
- [Aurora MySQL database engine updates 2019-02-07 \(version 1.19.0\) \(p. 1217\)](#)
- [Aurora MySQL database engine updates 2018-09-20 \(p. 1218\) \(Version 1.18.0\)](#)
- [Aurora MySQL database engine updates 2020-03-05 \(p. 1219\) \(Version 1.17.9\)](#)
- [Aurora MySQL database engine updates 2019-01-17 \(p. 1219\) \(Version 1.17.8\)](#)
- [Aurora MySQL database engine updates 2018-10-08 \(p. 1220\) \(Version 1.17.7\)](#)
- [Aurora MySQL database engine updates 2018-09-06 \(p. 1221\) \(Version 1.17.6\)](#)
- [Aurora MySQL database engine updates 2018-08-14 \(p. 1221\) \(Version 1.17.5\)](#)
- [Aurora MySQL database engine updates 2018-08-07 \(p. 1222\) \(Version 1.17.4\)](#)
- [Aurora MySQL database engine updates 2018-06-05 \(p. 1223\) \(Version 1.17.3\)](#)
- [Aurora MySQL database engine updates 2018-04-27 \(p. 1223\) \(Version 1.17.2\)](#)
- [Aurora MySQL database engine updates 2018-03-23 \(p. 1224\) \(Version 1.17.1\)](#)
- [Aurora MySQL database engine updates 2018-03-13 \(p. 1224\) \(Version 1.17\)](#)
- [Aurora MySQL database engine updates 2017-12-11 \(p. 1225\) \(Version 1.16\)](#)
- [Aurora MySQL database engine updates 2017-11-20 \(p. 1226\) \(Version 1.15.1\)](#)
- [Aurora MySQL database engine updates 2017-10-24 \(p. 1227\) \(Version 1.15\)](#)
- [Aurora MySQL database engine updates: 2018-03-13 \(p. 1229\) \(Version 1.14.4\)](#)
- [Aurora MySQL database engine updates: 2017-09-22 \(p. 1229\) \(Version 1.14.1\)](#)
- [Aurora MySQL database engine updates: 2017-08-07 \(p. 1230\) \(Version 1.14\)](#)
- [Aurora MySQL database engine updates: 2017-05-15 \(p. 1231\) \(Version 1.13\)](#)

- Aurora MySQL database engine updates: 2017-04-05 (p. 1232) (Version 1.12)
- Aurora MySQL database engine updates: 2017-02-23 (p. 1234) (Version 1.11)
- Aurora MySQL database engine updates: 2017-01-12 (p. 1236) (Version 1.10.1)
- Aurora MySQL database engine updates: 2016-12-14 (p. 1236) (Version 1.10)
- Aurora MySQL database engine updates: 2016-11-10 (p. 1237) (Versions 1.9.0, 1.9.1)
- Aurora MySQL database engine updates: 2016-10-26 (p. 1238) (Version 1.8.1)
- Aurora MySQL database engine updates: 2016-10-18 (p. 1238) (Version 1.8)
- Aurora MySQL database engine updates: 2016-09-20 (p. 1240) (Version 1.7.1)
- Aurora MySQL database engine updates: 2016-08-30 (p. 1240) (Version 1.7)
- Aurora MySQL database engine updates: 2016-06-01 (p. 1241) (Version 1.6.5)
- Aurora MySQL database engine updates: 2016-04-06 (p. 1241) (Version 1.6)
- Aurora MySQL database engine updates: 2016-01-11 (p. 1243) (Version 1.5)
- Aurora MySQL database engine updates: 2015-12-03 (p. 1243) (Version 1.4)
- Aurora MySQL database engine updates: 2015-10-16 (p. 1245) (Versions 1.2, 1.3)
- Aurora MySQL database engine updates: 2015-08-24 (p. 1247) (Version 1.1)

Aurora MySQL database engine updates 2021-09-30 (version 1.23.4)

Version: 1.23.4

Aurora MySQL 1.23.4 is generally available. Aurora MySQL 2.* versions are compatible with MySQL 5.7 and Aurora MySQL 1.* versions are compatible with MySQL 5.6.

Currently supported Aurora MySQL releases for upgrade to 1.23.4 are 1.14.*, 1.15.*, 1.16.*, 1.17.*, 1.18.*, 1.19.*, 1.20.*, 1.21.*, 1.22.*, and 1.23.*.

To create a cluster with an older version of Aurora MySQL, specify the engine version through the RDS Console, the AWS CLI, or the Amazon RDS API.

If you have any questions or concerns, AWS Support is available on the community forums and through [AWS Premium Support](#). For more information, see [Maintaining an Amazon Aurora DB cluster \(p. 443\)](#).

Improvements

General improvements:

- Fixed an issue that can cause high CPU consumption on the reader instances due to excessive logging of informational messages in internal diagnostic log files.

High-priority fixes:

- [CVE-2021-2307](#)
- [CVE-2021-2226](#)
- [CVE-2021-2160](#)
- [CVE-2021-2154](#)
- [CVE-2021-2060](#)
- [CVE-2021-2032](#)

- [CVE-2021-2001](#)

Aurora MySQL database engine updates 2021-06-28 (version 1.23.3)

Version: 1.23.3

Aurora MySQL 1.23.3 is generally available. Aurora MySQL 1.* versions are compatible with MySQL 5.6 and Aurora MySQL 2.* versions are compatible with MySQL 5.7.

Currently supported Aurora MySQL releases for upgrade to 1.23.3 are 1.14.*, 1.15.*, 1.16.*, 1.17.*, 1.18.*, 1.19.*, 1.20.*, 1.21.*, 1.22.*, and 1.23.*.

To create a cluster with an older version of Aurora MySQL, specify the engine version through the RDS Console, the AWS CLI, or the Amazon RDS API.

If you have any questions or concerns, AWS Support is available on the community forums and through [AWS Premium Support](#). For more information, see [Maintaining an Amazon Aurora DB cluster \(p. 443\)](#).

Improvements

General stability and availability enhancements.

Security fixes:

- [CVE-2021-23841](#)
- [CVE-2021-3449](#)
- [CVE-2020-28196](#)

Aurora MySQL database engine updates 2021-03-18 (version 1.23.2)

Version: 1.23.2

Aurora MySQL 1.23.2 is generally available. Aurora MySQL 1.* versions are compatible with MySQL 5.6 and Aurora MySQL 2.* versions are compatible with MySQL 5.7.

Currently supported Aurora MySQL releases are 1.19.*, 1.20.*, 1.21.*, 1.22.*, 1.23.*, and 2.04.*, 2.05.*, 2.06.*, 2.07.*, 2.08.* and 2.09.*. You can restore the snapshot of an Aurora MySQL 1.* database into Aurora MySQL 1.23.2.

To create a cluster with an older version of Aurora MySQL, specify the engine version through the RDS Console, the AWS CLI, or the Amazon RDS API.

Note

This version is currently not available in the following regions: AWS GovCloud (US-East) [us-gov-east-1], AWS GovCloud (US-West) [us-gov-west-1]. There will be a separate announcement once it is made available.

If you have any questions or concerns, AWS Support is available on the community forums and through [AWS Premium Support](#). For more information, see [Maintaining an Amazon Aurora DB cluster \(p. 443\)](#).

Improvements

High priority fixes:

- [CVE-2020-14867](#)
- [CVE-2020-14812](#)
- [CVE-2020-14769](#)
- [CVE-2020-14765](#)
- [CVE-2020-14793](#)
- [CVE-2020-14672](#)
- [CVE-2020-1971](#)
- [CVE-2018-3143](#)

Availability improvements:

- Fixed an issue in the dynamic cluster storage resizing feature that could cause reader DB instances to restart.
- Fixed a failover issue due to a race condition in `RESET QUERY CACHE` statement.
- Fixed a crash in a nested stored procedure call with query cache.
- Fixed an issue to prevent repeated restart of `mysqld` when recovering from an incomplete truncation of partitioned or sub-partitioned tables.
- Fixed an issue that could cause migration from on-prem or RDS for MySQL to Aurora MySQL to not succeed.
- Fixed a rare race condition where the database can restart during the scaling of the storage volume.
- Fixed an issue in the lock manager where a race condition can cause a lock to be shared by two transactions, causing the database to restart.
- Fixed an issue related to transaction lock memory management with long-running write transactions resulting in a database restart.
- Fixed a race condition in the lock manager that resulted in a database restart or failover during transaction rollback.
- Fixed an issue during upgrade from 5.6 to 5.7 when the table had Fast Online DDL enabled in lab mode in 5.6.
- Fixed multiple issues where the engine might restart during zero-downtime patching while checking for a quiesced point in database activity for patching.
- Fixed multiple issues related to repeated restarts due to interrupted DDL operations, such as `DROP TRIGGER`, `ALTER TABLE`, and specifically `ALTER TABLE` that modifies the type of partitioning or number of partitions in a table.
- Updated the default value of `table_open_cache` on 16XL and 24XL instances to avoid repeated restarts and high CPU utilization on large instances classes (R4/R5-16XL, R5-12XL, R5-24XL). This impacted 1.21.x and 1.22.x releases.
- Fixed an issue that caused a binlog replica to stop with an `HA_ERR_KEY_NOT_FOUND` error.

Integration of MySQL community edition bug fixes

- *Replication:* While a `SHOW BINLOG EVENTS` statement was executing, any parallel transaction was blocked. The fix ensures that the `SHOW BINLOG EVENTS` process now only acquires a lock for the duration of calculating the file's end position, therefore parallel transactions are not blocked for long durations. (Bug #76618, Bug #20928790)

Aurora MySQL database engine updates 2020-11-24 (version 1.23.1)

Version: 1.23.1

Aurora MySQL 1.23.1 is generally available. Aurora MySQL 1.* versions are compatible with MySQL 5.6 and Aurora MySQL 2.* versions are compatible with MySQL 5.7.

Currently supported Aurora MySQL releases for upgrade to 1.23.1 are 1.14.*, 1.15.*, 1.16.*, 1.17.*, 1.18.*, 1.19.*, 1.20.*, 1.21.*, 1.22.*, and 1.23.*.

To create a cluster with an older version of Aurora MySQL, specify the engine version through the RDS Console, the AWS CLI, or the Amazon RDS API.

If you have any questions or concerns, AWS Support is available on the community forums and through [AWS Premium Support](#). For more information, see [Maintaining an Amazon Aurora DB cluster \(p. 443\)](#).

Improvements

Security fixes:

Fixes and other enhancements to fine-tune handling in a managed environment. Additional CVE fixes below:

- [CVE-2020-14559](#)
- [CVE-2020-14539](#)

Incompatible changes:

This version introduces a permission change that affects the behavior of the `mysqldump` command. Users must have the `PROCESS` privilege to access the `INFORMATION_SCHEMA.FILES` table. To run the `mysqldump` command without any changes, grant the `PROCESS` privilege to the database user that the `mysqldump` command connects to. You can also run the `mysqldump` command with the `--no-tablespaces` option. With that option, the `mysqldump` output doesn't include any `CREATE LOGFILE GROUP` or `CREATE TABLESPACE` statements. In that case, the `mysqldump` command doesn't access the `INFORMATION_SCHEMA.FILES` table, and you don't need to grant the `PROCESS` permission.

Availability improvements:

- Fixed an issue that causes an Aurora reader instance in a global database secondary cluster running 1.23.0 to restart repeatedly.
- Fixed an issue where a global database secondary Region's replicas might restart when upgraded to release 1.23.0 while the primary Region writer was on an older release version.
- Fixed a memory leak in dynamic resizing feature, introduced in Aurora MySQL 1.23.0.
- Fixed an issue that might cause server restart during execution of a query using the parallel query feature.
- Fixed an issue that might cause a client session to hang when the database engine encounters an error while reading from or writing to the network.

Aurora MySQL database engine updates 2020-09-02 (version 1.23.0)

Version: 1.23.0

Aurora MySQL 1.23.0 is generally available. Aurora MySQL 1.* versions are compatible with MySQL 5.6 and Aurora MySQL 2.* versions are compatible with MySQL 5.7.

Currently supported Aurora MySQL releases are 1.14.* , 1.15.* , 1.16.* , 1.17.* , 1.18.* , 1.19.* , 1.20.* , 1.21.* , 1.22.* , 1.23.* , 2.01.* , 2.02.* , 2.03.* , 2.04.* , 2.05.* , 2.06.* , 2.07.* , 2.08.* , and 2.09.* . You can restore the snapshot of an Aurora MySQL 1.* database into Aurora MySQL 1.23.0.

Important

The improvements to Aurora storage in this version limit the available upgrade paths from Aurora MySQL 1.23 to Aurora MySQL 2.* . When you upgrade an Aurora MySQL 1.23 cluster to 2.* , you must upgrade to Aurora MySQL 2.09.0 or later.

To create a cluster with an older version of Aurora MySQL, please specify the engine version through the RDS Console, the AWS CLI, or the Amazon RDS API.

Note

This version is currently not available in the following regions: AWS GovCloud (US-East) [us-gov-east-1], AWS GovCloud (US-West) [us-gov-west-1]. There will be a separate announcement once it is made available.

If you have any questions or concerns, AWS Support is available on the community forums and through [AWS Premium Support](#). For more information, see [Maintaining an Amazon Aurora DB cluster \(p. 443\)](#).

Improvements

New features:

- You can now turn parallel query on or off for an existing cluster by changing the value of the DB cluster parameter `aurora_parallel_query`. You don't need to use the `parallelquery` setting for the `--engine-mode` parameter when creating the cluster.

Parallel query is now expanded to be available in all regions where Aurora MySQL is available.

There are a number of other functionality enhancements and changes to the procedures for upgrading and enabling parallel query in an Aurora cluster. For more information, see [Working with parallel query for Amazon Aurora MySQL \(p. 881\)](#).

- With this release, you can create Amazon Aurora MySQL database instances with up to 128 tebibytes (TiB) of storage. The new storage limit is an increase from the prior 64 TiB. The 128 TiB storage size supports larger databases. This capability is not supported on small instance sizes (db.t2 or db.t3). A single tablespace cannot grow beyond 64 TiB due to [InnoDB limitations with 16 KB page size](#).

Aurora alerts you when the cluster volume size is near 128 TiB, so that you can take action prior to hitting the size limit. The alerts appear in the mysql log and RDS Events in the AWS Management Console.

- Improved binary log (binlog) processing to reduce crash recovery time and commit time latency when very large transactions are involved.
- Aurora dynamically resizes your cluster storage space. With dynamic resizing, the storage space for your Aurora DB cluster automatically decreases when you remove data from the DB cluster. For more information, see [Storage scaling \(p. 396\)](#).

Note

The dynamic resizing feature is being deployed in phases to the AWS Regions where Aurora is available. Depending on the Region where your cluster is, this feature might not be available yet. For more information, see the [What's New announcement](#).

High priority fixes:

- [CVE-2019-2911](#)
- [CVE-2019-2537](#)
- [CVE-2018-2787](#)

- [CVE-2018-2784](#)
- [CVE-2018-2645](#)
- [CVE-2018-2640](#)

Availability improvements:

- Fixed an issue in the lock manager where a race condition can cause a lock to be shared by two transactions, causing the database to restart.
- Fixed an issue related to transaction lock memory management with long-running write transactions resulting in a database restart.
- Fixed a race condition in the lock manager that resulted in a database restart or failover during transaction rollback.
- Fixed an issue during upgrade from 5.6 to 5.7 when the `innodb_file_format` changed on a table that had Fast DDL enabled.
- Fixed multiple issues where the engine might restart during zero-downtime patching while checking for a quiesced point in database activity for patching.
- Fixed an issue related to DDL recovery that impacts restart of the DB instance while recovering an interrupted `DROP TRIGGER` operation.
- Fixed a bug that might cause unavailability of the database if a crash occurs during the execution of certain partitioning operations. Specifically, an interrupted `ALTER TABLE` operation that modifies the type of partitioning or the number of partitions in a table.
- Fix default value of `table_open_cache` on 16XL and 24XL instances which could cause repeated failovers and high CPU utilization on large instances classes (R4/R5-16XL, R5-12XL, R5-24XL). This impacted 1.21.x and 1.22.x.

Global databases:

- Populate missing data in the MySQL `INFORMATION_SCHEMA.REPLICA_HOST_STATUS` view on primary and secondary AWS Regions in an Aurora global database.
- Fixed unexpected query failures that could occur in a Global DB secondary Region due to garbage collection of UNDO records in the primary Region, after temporary network connectivity issues between the primary and secondary Regions.

Parallel query:

- Fixed an issue where parallel query might cause a long-running query to return an empty result.
- Fixed an issue where a query on a small table on the Aurora read replica might take more than one second.
- Fixed an issue that might cause a restart when a parallel query and a DML statement are executing concurrently under a heavy workload.

General improvements:

- Fixed an issue where queries using spatial index might return partial results if spatial index was created on tables with already existing large spatial values.
- Increased maximum allowable length for audit system variables `server_audit_incl_users` and `server_audit_excl_users` from 1024 bytes to 2000 bytes.
- Fixed an issue where a binlog replica connected to an Aurora MySQL binlog primary might show incomplete data when the Aurora MySQL binlog primary loads data from S3 under `statement binlog_format`.

- Comply with community behavior to map `mixed binlog_format` to `row` instead of `statement` for loading data.
- Fixed an issue causing binlog replication to stop working when the user closes the connection and the session is using temporary tables.
- Improved response time of a query involving MyISAM temporary tables.
- Fix permission issue when binlog worker runs a native lambda function.
- Fixed an issue on Aurora read replicas when trying to query or rotate the slow log or general log.
- Fixed an issue that broke logical replication when the `binlog_checksum` parameter is set to different values on the master and the replica.
- Fixed an issue where the read replica might transiently see partial results of a recently committed transaction on the writer.
- Include transaction info of the rolled-back transaction in `show engine innodb status` when a deadlock is resolved.

Integration of MySQL community edition bug fixes

- Binlog events with `ALTER TABLE ADD COLUMN ALGORITHM=QUICK` will be rewritten as `ALGORITHM=DEFAULT` to be compatible with the community edition.
- BUG #22350047: IF CLIENT KILLED AFTER ROLLBACK TO SAVEPOINT PREVIOUS STMTS COMMITTED
- Bug #29915479: RUNNING COM_REGISTER_SLAVE WITHOUT COM_BINLOG_DUMP CAN RESULTS IN SERVER EXIT
- Bug #30441969: BUG #29723340: MYSQL SERVER CRASH AFTER SQL QUERY WITH DATA ?AST
- Bug #30628268: OUT OF MEMORY CRASH
- Bug #27081349: UNEXPECTED BEHAVIOUR WHEN DELETE WITH SPATIAL FUNCTION
- Bug #27230859: UNEXPECTED BEHAVIOUR WHILE HANDLING INVALID POLYGON"
- Bug #27081349: UNEXPECTED BEHAVIOUR WHEN DELETE WITH SPATIAL"
- Bug #26935001: ALTER TABLE AUTO_INCREMENT TRIES TO READ INDEX FROM DISCARDED TABLESPACE
- Bug #29770705: SERVER CRASHED WHILE EXECUTING SELECT WITH SPECIFIC WHERE CLAUSE
- Bug #27659490: SELECT USING DYNAMIC RANGE AND INDEX MERGE USE TOO MUCH MEMORY(OOM)
- Bug #24786290: REPLICATION BREAKS AFTER BUG #74145 HAPPENS IN MASTER
- Bug #27703912: EXCESSIVE MEMORY USAGE WITH MANY PREPARE
- Bug #20527363: TRUNCATE TEMPORARY TABLE CRASH: !DICT_TF2_FLAG_IS_SET(TABLE, DICT_TF2_TEMPORARY)
- Bug#23103937 PS_TRUNCATE_ALL_TABLES() DOES NOT WORK IN SUPER_READ_ONLY MODE
- Bug #25053286: USE VIEW WITH CONDITION IN PROCEDURE CAUSES INCORRECT BEHAVIOR (fixed in 5.6.36)
- Bug #25586773: INCORRECT BEHAVIOR FOR CREATE TABLE SELECT IN A LOOP IN SP (fixed in 5.6.39)
- Bug #27407480: AUTOMATIC_SP_PRIVILEGES REQUIRES NEED THE INSERT PRIVILEGES FOR MYSQL.USER TABLE
- Bug #26997096: `relay_log_space` value is not updated in a synchronized manner so that its value is sometimes much higher than the actual disk space used by relay logs.
- Bug#15831300 SLAVE_TYPE_CONVERSIONS=ALL_NON_LOSSY NOT WORKING AS EXPECTED
- SSL Bug backport Bug #17087862, Bug #20551271
- Bug #16894092: PERFORMANCE REGRESSION IN 5.6.6+ FOR INSERT INTO ... SELECT ... FROM (fixed in 5.6.15).
- Port a bug fix related to `SLAVE_TYPE_CONVERSIONS`.

Aurora MySQL database engine updates 2021-06-03 (version 1.22.5)

Version: 1.22.5

Aurora MySQL 1.22.5 is generally available. Aurora MySQL 1.* versions are compatible with MySQL 5.6 and Aurora MySQL 2.* versions are compatible with MySQL 5.7.

Currently supported Aurora MySQL releases for upgrade to 1.22.5 are 1.14.*, 1.15.*, 1.16.*, 1.17.*, 1.18.*, 1.19.*, 1.20.*, 1.21.*, and 1.22.*.

To create a cluster with an older version of Aurora MySQL, specify the engine version through the RDS Console, the AWS CLI, or the Amazon RDS API.

Note

This version is designated as a long-term support (LTS) release. For more information, see [Aurora MySQL long-term support \(LTS\) releases \(p. 1085\)](#).

If you have any questions or concerns, AWS Support is available on the community forums and through [AWS Premium Support](#). For more information, see [Maintaining an Amazon Aurora DB cluster \(p. 443\)](#).

Improvements

Availability improvements:

- Resolved an issue that could cause the database to stall, and subsequently restart or fail over due to a concurrency conflict between internal cleanup threads.
- Resolved an issue that could cause the cluster to become unavailable if the database restarted while holding XA transactions in prepared state, and then restarted again before those transactions were committed or rolled back. Prior to this fix, you can address the issue by restoring the cluster to a point in time before the first restart.
- Resolved an issue that could cause the InnoDB purge to become blocked if the database restarts while processing a DDL statement. As a result, the InnoDB history list length would grow and the cluster storage volume would keep growing until it fills up, making the database unavailable. Prior to this fix, you can mitigate the issue by restarting the database again to unblock purge.

Aurora MySQL database engine updates 2021-03-04 (version 1.22.4)

Version: 1.22.4

Aurora MySQL 1.22.4 is generally available. Aurora MySQL 1.* versions are compatible with MySQL 5.6 and Aurora MySQL 2.* versions are compatible with MySQL 5.7.

Currently supported Aurora MySQL releases for upgrade to 1.22.4 are 1.14.*, 1.15.*, 1.16.*, 1.17.*, 1.18.*, 1.19.*, 1.20.*, 1.21.*, and 1.22.*.

To create a cluster with an older version of Aurora MySQL, specify the engine version through the RDS Console, the AWS CLI, or the Amazon RDS API.

Note

This version is designated as a long-term support (LTS) release. For more information, see [Aurora MySQL long-term support \(LTS\) releases \(p. 1085\)](#).

If you have any questions or concerns, AWS Support is available on the community forums and through [AWS Premium Support](#). For more information, see [Maintaining an Amazon Aurora DB cluster \(p. 443\)](#).

Improvements

Security fixes:

Fixes and other enhancements to fine-tune handling in a managed environment. Additional CVE fixes below:

- [CVE-2020-14867](#)
- [CVE-2020-14812](#)
- [CVE-2020-14793](#)
- [CVE-2020-14769](#)
- [CVE-2020-14765](#)
- [CVE-2020-14672](#)
- [CVE-2020-1971](#)

Availability improvements:

- Fixed an issue that could trigger a database restart or failover during a `kill session` command. If you encounter this issue, contact AWS support to enable this fix on your instance.
- Improved binary logging to reduce crash recovery time and commit latency when large transactions are involved.
- Fixed an issue that caused a binlog replica to stop with an `HA_ERR_KEY_NOT_FOUND` error.

Aurora MySQL database engine updates 2020-11-09 (version 1.22.3)

Version: 1.22.3

Aurora MySQL 1.22.3 is generally available. Aurora MySQL 1.* versions are compatible with MySQL 5.6 and Aurora MySQL 2.* versions are compatible with MySQL 5.7.

Currently supported Aurora MySQL releases for upgrade to 1.22.3 are 1.14.*, 1.15.*, 1.16.*, 1.17.*, 1.18.*, 1.19.*, 1.20.*, 1.21.*, and 1.22.*.

To create a cluster with an older version of Aurora MySQL, specify the engine version through the RDS Console, the AWS CLI, or the Amazon RDS API.

Note

This version is designated as a long-term support (LTS) release. For more information, see [Aurora MySQL long-term support \(LTS\) releases \(p. 1085\)](#).

If you have any questions or concerns, AWS Support is available on the community forums and through [AWS Premium Support](#). For more information, see [Maintaining an Amazon Aurora DB cluster \(p. 443\)](#).

Improvements

Security fixes:

Fixes and other enhancements to fine-tune handling in a managed environment. Additional CVE fixes below:

- [CVE-2020-14559](#)
- [CVE-2020-14539](#)

- [CVE-2020-2579](#)
- [CVE-2020-2812](#)
- [CVE-2020-2780](#)
- [CVE-2020-2763](#)

Incompatible changes:

This version introduces a permission change that affects the behavior of the `mysqldump` command. Users must have the `PROCESS` privilege to access the `INFORMATION_SCHEMA.FILES` table. To run the `mysqldump` command without any changes, grant the `PROCESS` privilege to the database user that the `mysqldump` command connects to. You can also run the `mysqldump` command with the `--no-tablespaces` option. With that option, the `mysqldump` output doesn't include any `CREATE LOGFILE GROUP` or `CREATE TABLESPACE` statements. In that case, the `mysqldump` command doesn't access the `INFORMATION_SCHEMA.FILES` table, and you don't need to grant the `PROCESS` permission.

Availability improvements:

- Fixed issues that might cause server restarts during recovery of a DDL statement that was not committed.
- Fixed race conditions in the lock manager that can cause a server restart.
- Fixed an issue that might cause the monitoring agent to restart the server during recovery of a large transaction

General improvements:

- Changed the behavior to map `MIXED binlog_format` to `ROW` instead of `STATEMENT` when executing `LOAD DATA FROM INFILE | S3`.
- Fixed an issue where a binlog replica connected to an Aurora MySQL binlog primary might show incomplete data when the primary executed `LOAD DATA FROM S3` and `binlog_format` is set to `STATEMENT`.

Integration of MySQL community edition bug fixes

- Bug #26654685: A corrupt index ID encountered during a foreign key check raised an assertion
- Bug #15831300: By default, when promoting integers from a smaller type on the master to a larger type on the slave (for example, from a `SMALLINT` column on the master to a `BIGINT` column on the slave), the promoted values are treated as though they are signed. Now in such cases it is possible to modify or override this behavior using one or both of `ALL_SIGNED`, `ALL_UNSIGNED` in the set of values specified for the `slave_type_conversions` server system variable. For more information, see [Row-based replication: attribute promotion and demotion](#), as well as the description of the variable.
- Bug #17449901: With `foreign_key_checks=0`, InnoDB permitted an index required by a foreign key constraint to be dropped, placing the table into an inconsistent state and causing the foreign key check that occurs at table load to fail. InnoDB now prevents dropping an index required by a foreign key constraint, even with `foreign_key_checks=0`. The foreign key constraint must be removed before dropping the foreign key index.
- BUG #20768847: An `ALTER TABLE ... DROP INDEX` operation on a table with foreign key dependencies raised an assertion.

Aurora MySQL database engine updates 2020-03-05 (version 1.22.2)

Version: 1.22.2

Aurora MySQL 1.22.2 is generally available. Aurora MySQL 1.* versions are compatible with MySQL 5.6 and Aurora MySQL 2.* versions are compatible with MySQL 5.7.

Currently supported Aurora MySQL releases are 1.14.*, 1.15.*, 1.16.*, 1.17.*, 1.18.*, 1.19.*, 1.20.*, 1.21.*, 1.22.*, 2.01.*, 2.02.*, 2.03.*, 2.04.*, 2.05.*, 2.06.* and 2.07.*. You can restore the snapshot of an Aurora MySQL 1.* database into Aurora MySQL 1.22.2.

To create a cluster with an older version of Aurora MySQL, please specify the engine version through the RDS Console, the AWS CLI, or the Amazon RDS API.

Note

This version is currently not available in the following regions: AWS GovCloud (US-East) [us-gov-east-1], AWS GovCloud (US-West) [us-gov-west-1]. There will be a separate announcement once it is made available.

This version is designated as a long-term support (LTS) release. For more information, see [Aurora MySQL long-term support \(LTS\) releases \(p. 1085\)](#).

If you have any questions or concerns, AWS Support is available on the community forums and through [AWS Premium Support](#). For more information, see [Maintaining an Amazon Aurora DB cluster \(p. 443\)](#).

Improvements

High priority fixes:

- Fixed an issue of intermittent connection failures after certificate rotation.
- Fixed an issue that caused cloning to take longer on some database clusters with high write loads.
- Fixed an issue that broke logical replication when the `binlog_checksum` parameter is set to different values on the master and the replica.
- Fixed an issue where slow log and general log may not properly rotate on read replicas.
- Fixed an issue with ANSI Read Committed Isolation Level behavior.

Aurora MySQL database engine updates 2019-12-23 (version 1.22.1)

Version: 1.22.1

Aurora MySQL 1.22.1 is generally available. Aurora MySQL 1.* versions are compatible with MySQL 5.6 and Aurora MySQL 2.* versions are compatible with MySQL 5.7.

Currently supported Aurora MySQL releases are 1.14.*, 1.15.*, 1.16.*, 1.17.*, 1.18.*, 1.19.*, 1.20.*, 1.21.*, 1.22.*, 2.01.*, 2.02.*, 2.03.*, 2.04.*, 2.05.*, 2.06.* and 2.07.*. To create a cluster with an older version of Aurora MySQL, please specify the engine version through the AWS Management Console, the AWS CLI or the RDS API. You have the option to upgrade existing Aurora MySQL 1.* database clusters to Aurora MySQL 1.22.1.

Note

This version is currently not available in the following AWS Regions: AWS GovCloud (US-East) [us-gov-east-1], AWS GovCloud (US-West) [us-gov-west-1], China (Ningxia) [cn-northwest-1], Asia Pacific (Hong Kong) [ap-east-1], and Middle East (Bahrain) [me-south-1]. There will be a separate announcement once it is made available.

If you have any questions or concerns, AWS Support is available on the community forums and through [AWS Premium Support](#). For more information, see [Maintaining an Amazon Aurora DB cluster \(p. 443\)](#).

Note

The procedure to upgrade your DB cluster has changed. For more information, see [Upgrading the minor version or patch level of an Aurora MySQL DB cluster \(p. 1088\)](#).

Improvements

Critical fixes:

- Fixed issues that prevented engine recovery involving table locks and temporary tables.
- Improved the stability of binary log when temporary tables are used.

High priority fixes:

- Fixed a slow memory leak in Aurora specific database tracing and logging sub-system that lowers the freeable memory.

Aurora MySQL database engine updates 2019-11-25 (version 1.22.0)

Version: 1.22.0

Aurora MySQL 1.22.0 is generally available. Aurora MySQL 1.* versions are compatible with MySQL 5.6 and Aurora MySQL 2.* versions are compatible with MySQL 5.7.

Currently supported Aurora MySQL releases are 1.14.*, 1.15.*, 1.16.*, 1.17.*, 1.18.*, 1.19.*, 1.20.*, 1.21.*, 1.22.*, 2.01.*, 2.02.*, 2.03.*, 2.04.*, 2.05.*, 2.06.*, and 2.07.*. To create a cluster with an older version of Aurora MySQL, please specify the engine version through the AWS Management Console, the AWS CLI or the RDS API. You have the option to upgrade existing Aurora MySQL 1.* database clusters to Aurora MySQL 1.22.0.

Note

This version is currently not available in the following AWS Regions: AWS GovCloud (US-East) [us-gov-east-1], AWS GovCloud (US-West) [us-gov-west-1], China (Ningxia) [cn-northwest-1], Asia Pacific (Hong Kong) [ap-east-1], Middle East (Bahrain) [me-south-1], and South America (São Paulo) [sa-east-1]. There will be a separate announcement once it is made available.

If you have any questions or concerns, AWS Support is available on the community forums and through [AWS Premium Support](#). For more information, see [Maintaining an Amazon Aurora DB cluster \(p. 443\)](#).

Note

The procedure to upgrade your DB cluster has changed. For more information, see [Upgrading the minor version or patch level of an Aurora MySQL DB cluster \(p. 1088\)](#).

Improvements

New features:

- Aurora MySQL clusters now support the instance types r5.8xlarge, r5.16xlarge and r5.24xlarge.
- Binlog has new enhancements for improved commit time latency when very large transactions are involved.
- Aurora MySQL now has a mechanism to minimize the time window during which events of a large transaction are written to binlog on commit. This effectively prevents lengthy offline recovery incurred when database crashes occur during that time window. This feature also fixes the issue where a large transaction blocks small transactions on binlog commit. This feature is off by default and can be enabled by the service team if needed for your workload. When enabled, it will be triggered when a transaction size is > 500MB.
- Added support for the ANSI READ COMMITTED isolation level on the read replicas. This isolation level enables long-running queries on the read replica to execute without impacting the high throughput of writes on the writer node. For more information, see [Aurora MySQL isolation levels](#).

- Global Databases now allow adding secondary read-only replica regions for database clusters deployed in these AWS Regions: regions: US East (N. Virginia) [us-east-1], US East (Ohio) [us-east-2], US West (N. California) [us-west-1], US West (Oregon) [us-west-2], Europe (Ireland) [eu-west-1], Europe (London) [eu-west-2], Europe (Paris) [eu-west-3], Asia Pacific (Tokyo) [ap-northeast-1], Asia Pacific (Seoul) [ap-northeast-2], Asia Pacific (Singapore) [ap-southeast-1], Asia Pacific (Sydney) [ap-southeast-2], Canada (Central) [ca-central-1], Europe (Frankfurt) [eu-central-1], and Asia Pacific (Mumbai) [ap-south-1].
- The hot row contention feature is now generally available and does not require the Aurora lab mode setting to be ON. This feature substantially improves throughput for workloads with many transactions contending for rows on the same page.
- This version has updated timezone files to support the latest Brazil timezone update for new clusters.

Critical fixes:

- [CVE-2019-2922](#)
- [CVE-2019-2923](#)
- [CVE-2019-2924](#)
- [CVE-2019-2910](#)

High priority fixes:

- [CVE-2019-2805](#)
- [CVE-2019-2730](#)
- [CVE-2019-2740](#)
- [CVE-2018-3064](#)
- [CVE-2018-3058](#)
- [CVE-2017-3653](#)
- [CVE-2017-3464](#)
- [CVE-2017-3244](#)
- [CVE-2016-5612](#)
- [CVE-2016-5439](#)
- [CVE-2016-0606](#)
- [CVE-2015-4904](#)
- [CVE-2015-4879](#)
- [CVE-2015-4864](#)
- [CVE-2015-4830](#)
- [CVE-2015-4826](#)
- [CVE-2015-2620](#)
- [CVE-2015-0382](#)
- [CVE-2015-0381](#)
- [CVE-2014-6555](#)
- [CVE-2014-4258](#)
- [CVE-2014-4260](#)
- [CVE-2014-2444](#)
- [CVE-2014-2436](#)
- [CVE-2013-5881](#)
- [CVE-2014-0393](#)

- [CVE-2013-5908](#)
- [CVE-2013-5807](#)
- [CVE-2013-3806](#)
- [CVE-2013-3811](#)
- [CVE-2013-3804](#)
- [CVE-2013-3807](#)
- [CVE-2013-2378](#)
- [CVE-2013-2375](#)
- [CVE-2013-1523](#)
- [CVE-2013-2381](#)
- [CVE-2012-5615](#)
- [CVE-2014-6489](#)
- Fixed an issue in the DDL recovery component that resulted in prolonged database downtime. Clusters that become unavailable after executing `TRUNCATE TABLE` query on a table with an `AUTO_INCREMENT` column should be updated.
- Fixed an issue in the DDL recovery component that resulted in prolonged database downtime. Clusters that become unavailable after executing `DROP TABLE` query on multiple tables in parallel should be updated.

General stability fixes:

- Fixed an issue that caused read replicas to restart during a long-running transaction. Customers who encounter replica restarts that coincide with an accelerated drop in freeable memory should consider upgrading to this version.
- Fixed an issue that incorrectly reported `ERROR 1836` when a nested query is executed against a temporary table on the read replica.
- Fixed a parallel query abort error on an Aurora reader instance while a heavy write workload is running on the Aurora writer instance.
- Fixed an issue that causes a database configured as a Binlog Master to restart while a heavy write workload is running.
- Fixed an issue of prolonged unavailability while restarting the engine. This addresses an issue in the buffer pool initialization. This issue occurs rarely but can potentially impact any supported release.
- Fixed an issue that generated inconsistent data in the `information_schema.replica_host_status` table.
- Fixed a race condition between the parallel query and the standard execution paths that caused the Reader nodes to restart intermittently.
- Improved stability of the database when the number of client connections exceeds the `max_connections` parameter value.
- Improved stability of the reader instances by blocking unsupported DDL and `LOAD FROM S3` queries.

Integration of MySQL community edition bug fixes

- Bug#16346241 - SERVER CRASH IN ITEM_PARAM::QUERY_VAL_STR
- Bug#17733850 - NAME_CONST() CRASH IN ITEM_NAME_CONST::ITEM_NAME_CONST()
- Bug #20989615 - INNODB AUTO_INCREMENT PRODUCES SAME VALUE TWICE
- Bug #20181776 - ACCESS CONTROL DOESN'T MATCH MOST SPECIFIC HOST WHEN IT CONTAINS WILDCARD
- Bug #27326796 - MYSQL CRASH WITH INNODB ASSERTION FAILURE IN FILE PARSONS.CC

- Bug #20590013 - IF YOU HAVE A FULLTEXT INDEX AND DROP IT YOU CAN NO LONGER PERFORM ONLINE DDL

Aurora MySQL database engine updates 2019-11-25 (version 1.21.0)

Version: 1.21.0

Aurora MySQL 1.21.0 is generally available. Aurora MySQL 1.* versions are compatible with MySQL 5.6 and Aurora MySQL 2.* versions are compatible with MySQL 5.7.

Currently supported Aurora MySQL releases are 1.14.*, 1.15.*, 1.16.*, 1.17.*, 1.18.*, 1.19.*, 1.20.*, 1.21.*, 2.01.*, 2.02.*, 2.03.* and 2.04.*. To create a cluster with an older version of Aurora MySQL, please specify the engine version through the AWS Management Console, the AWS CLI or the RDS API. You have the option to upgrade existing Aurora MySQL 1.* database clusters to Aurora MySQL 1.21.0.

Note

This version is currently not available in the following AWS Regions: AWS GovCloud (US-East) [us-gov-east-1], AWS GovCloud (US-West) [us-gov-west-1], China (Ningxia) [cn-northwest-1], Asia Pacific (Hong Kong) [ap-east-1], Europe (Stockholm) [eu-north-1], and Middle East (Bahrain) [me-south-1]. There will be a separate announcement once it is made available.

If you have any questions or concerns, AWS Support is available on the community forums and through [AWS Premium Support](#). For more information, see [Maintaining an Amazon Aurora DB cluster \(p. 443\)](#).

Note

The procedure to upgrade your DB cluster has changed. For more information, see [Upgrading the minor version or patch level of an Aurora MySQL DB cluster \(p. 1088\)](#).

Improvements

Critical fixes:

- [CVE-2018-0734](#)
- [CVE-2019-2534](#)
- [CVE-2018-2612](#)
- [CVE-2017-3599](#)
- [CVE-2018-2562](#)
- [CVE-2017-3329](#)
- [CVE-2018-2696](#)
- [CVE-2015-4737](#)

High priority fixes:

- Customers with database size close to 64 tebibytes (TiB) are strongly advised to upgrade to this version to avoid downtime due to stability bugs affecting volumes close to the Aurora storage limit.

General stability fixes:

- Fixed a parallel query abort error on Aurora reader instances while a heavy write workload is running on the Aurora writer instance.
- Fixed an issue on Aurora reader instances that reduced free memory during long-running transactions while there is a heavy transaction commit traffic on the writer instance.
- The value of the parameter `aurora_disable_hash_join` is now persisted after database restart or host replacement.

- Fixed an issue related to the Full Text Search cache that caused the Aurora instance to run out of memory. Customers using Full Text Search should upgrade.
- Improved stability of the database when the hash join feature is enabled and the instance is low on memory. Customers using hash join should upgrade.
- Fixed an issue in the query cache where the "Too many connections" error could cause a reboot.
- Fixed the free memory calculation on T2 instances to include swap memory space to prevent unnecessary reboots.

Integration of MySQL community edition bug fixes

- Bug #19929406: HANDLE_FATAL_SIGNAL (SIG=11) IN __MEMMOVE_SSSE3_BACK FROM STRING::COPY
- Bug #17059925: For `UNION` statements, the rows-examined value was calculated incorrectly. This was manifested as too-large values for the `ROWS_EXAMINED` column of Performance Schema statement tables (such as `events_statements_current`).
- Bug #11827369: Some queries with `SELECT ... FROM DUAL` nested subqueries raised an assertion.
- Bug #16311231: Incorrect results were returned if a query contained a subquery in an `IN` clause that contained an `XOR` operation in the `WHERE` clause.

Aurora MySQL database engine updates 2020-03-05 (version 1.20.1)

Version: 1.20.1

Aurora MySQL 1.20.1 is generally available. Aurora MySQL 1.* versions are compatible with MySQL 5.6 and Aurora MySQL 2.* versions are compatible with MySQL 5.7.

Currently supported Aurora MySQL releases are 1.14.*, 1.15.*, 1.16.*, 1.17.*, 1.18.*, 1.19.*, 1.20.*, 1.21.*, 1.22.*, 2.01.*, 2.02.*, 2.03.*, 2.04.*, 2.05.*, 2.06.* and 2.07.*. You can restore the snapshot of an Aurora MySQL 1.* database into Aurora MySQL 1.20.1.

To create a cluster with an older version of Aurora MySQL, please specify the engine version through the RDS Console, the AWS CLI, or the Amazon RDS API.

Note

This version is currently not available in the following regions: AWS GovCloud (US-East) [us-gov-east-1], AWS GovCloud (US-West) [us-gov-west-1]. There will be a separate announcement once it is made available.

If you have any questions or concerns, AWS Support is available on the community forums and through [AWS Premium Support](#). For more information, see [Maintaining an Amazon Aurora DB cluster \(p. 443\)](#).

Improvements

High priority fixes:

- Fixed an issue of intermittent connection failures after certificate rotation.
- Fixed an issue related to connection close concurrency that would result in a failover under heavy workload.

General stability fixes:

- Fixed a crash during execution of a complex query involving multi-table joins and aggregation that uses intermediate tables internally.

Aurora MySQL database engine updates 2019-11-11 (version 1.20.0)

Version: 1.20.0

Aurora MySQL 1.20.0 is generally available. Aurora MySQL 1.* versions are compatible with MySQL 5.6 and Aurora MySQL 2.* versions are compatible with MySQL 5.7.

Currently supported Aurora MySQL releases are 1.14.*, 1.15.*, 1.16.*, 1.17.*, 1.18.*, 1.19.*, 1.20.*, 2.01.*, 2.02.*, 2.03.* and 2.04.*. To create a cluster with an older version of Aurora MySQL, please specify the engine version through the AWS Management Console, the AWS CLI or the RDS API. You have the option to upgrade existing Aurora MySQL 1.* database clusters, up to 1.19.5, to Aurora MySQL 1.20.0.

Note

This version is currently not available in the following AWS Regions: AWS GovCloud (US-East) [us-gov-east-1], AWS GovCloud (US-West) [us-gov-west-1], China (Ningxia) [cn-northwest-1], Asia Pacific (Hong Kong) [ap-east-1], Europe (Stockholm) [eu-north-1], and Middle East (Bahrain) [me-south-1]. There will be a separate announcement once it is made available.

If you have any questions or concerns, AWS Support is available on the community forums and through [AWS Premium Support](#). For more information, see [Maintaining an Amazon Aurora DB cluster \(p. 443\)](#).

Note

The procedure to upgrade your DB cluster has changed. For more information, see [Upgrading the minor version or patch level of an Aurora MySQL DB cluster \(p. 1088\)](#).

Improvements

Critical fixes:

- [CVE-2018-0734](#)
- [CVE-2019-2534](#)
- [CVE-2018-2612](#)
- [CVE-2017-3599](#)
- [CVE-2018-2562](#)
- [CVE-2017-3329](#)
- [CVE-2018-2696](#)
- [CVE-2015-4737](#)

High priority fixes:

- Customers with database size close to 64 tebibytes (TiB) are strongly advised to upgrade to this version to avoid downtime due to stability bugs affecting volumes close to the Aurora storage limit.

General stability fixes:

- Fixed a parallel query abort error on Aurora reader instances while a heavy write workload is running on the Aurora writer instance.
- Fixed an issue on Aurora reader instances that reduced free memory during long-running transactions while there is a heavy transaction commit traffic on the writer instance.
- The value of the parameter `aurora_disable_hash_join` is now persisted after database restart or host replacement.
- Fixed an issue related to the Full Text Search cache that caused the Aurora instance to run out of memory. Customers using Full Text Search should upgrade.

- Improved stability of the database when the hash join feature is enabled and the instance is low on memory. Customers using hash join should upgrade.
- Fixed an issue in the query cache where the "Too many connections" error could cause a reboot.
- Fixed the free memory calculation on T2 instances to include swap memory space to prevent unnecessary reboots.

Integration of MySQL community edition bug fixes

- Bug #19929406: HANDLE_FATAL_SIGNAL (SIG=11) IN __MEMMOVE_SSSE3_BACK FROM STRING::COPY
- Bug #17059925: For [UNION](#) statements, the rows_examined value was calculated incorrectly. This was manifested as too-large values for the ROWS_EXAMINED column of Performance Schema statement tables (such as [events_statements_current](#)).
- Bug #11827369: Some queries with `SELECT ... FROM DUAL` nested subqueries raised an assertion.
- Bug #16311231: Incorrect results were returned if a query contained a subquery in an `IN` clause that contained an [XOR](#) operation in the `WHERE` clause.

Aurora MySQL database engine updates 2020-03-05 (version 1.19.6)

Version: 1.19.6

Aurora MySQL 1.19.6 is generally available. Aurora MySQL 1.* versions are compatible with MySQL 5.6 and Aurora MySQL 2.* versions are compatible with MySQL 5.7.

Currently supported Aurora MySQL releases are 1.14.*, 1.15.*, 1.16.*, 1.17.*, 1.18.*, 1.19.*, 1.20.*, 1.21.*, 1.22.*, 2.01.*, 2.02.*, 2.03.*, 2.04.*, 2.05.*, 2.06.* and 2.07.*. You can restore the snapshot of an Aurora MySQL 1.* database into Aurora MySQL 1.19.6.

To create a cluster with an older version of Aurora MySQL, please specify the engine version through the RDS Console, the AWS CLI, or the Amazon RDS API.

Note

This version is currently not available in the following regions: AWS GovCloud (US-East) [us-gov-east-1], AWS GovCloud (US-West) [us-gov-west-1]. There will be a separate announcement once it is made available.

If you have any questions or concerns, AWS Support is available on the community forums and through [AWS Premium Support](#). For more information, see [Maintaining an Amazon Aurora DB cluster \(p. 443\)](#).

Improvements

High priority fixes:

- Fixed an issue of intermittent connection failures after certificate rotation.

Aurora MySQL database engine updates 2019-09-19 (version 1.19.5)

Version: 1.19.5

Aurora MySQL 1.19.5 is generally available. Aurora MySQL 1.* versions are compatible with MySQL 5.6 and Aurora MySQL 2.* versions are compatible with MySQL 5.7.

You have the option to upgrade existing database clusters to Aurora MySQL 1.19.5. You can restore snapshots of Aurora MySQL 1.14.* , 1.15.* , 1.16.* , 1.17.* , 1.18.* , 1.19.1, and 1.19.2 into Aurora MySQL 1.19.5.

To use an older version of Aurora MySQL, you can create new database clusters by specifying the engine version through the AWS Management Console, the AWS CLI, or the RDS API.

If you have any questions or concerns, AWS Support is available on the community forums and through [AWS Premium Support](#). For more information, see [Maintaining an Amazon Aurora DB cluster \(p. 443\)](#).

Note

This version is currently not available in the following AWS Regions: Europe (London) [eu-west-2], AWS GovCloud (US-East) [us-gov-east-1], AWS GovCloud (US-West) [us-gov-west-1], China (Ningxia) [cn-northwest-1], and Asia Pacific (Hong Kong) [ap-east-1]. There will be a separate announcement once it is made available.

Note

The procedure to upgrade your DB cluster has changed. For more information, see [Upgrading the minor version or patch level of an Aurora MySQL DB cluster \(p. 1088\)](#).

Improvements

- Fixed an issue on Aurora reader instances that reduced free memory during long-running transactions while there is a heavy transaction commit traffic on the writer instance.
- Fixed a parallel query abort error on Aurora reader instances while a heavy write workload is running on the Aurora writer instance.
- The value of the parameter `aurora_disable_hash_join` is now persisted after database restart or host replacement.
- Fixed an issue related to the Full Text Search cache that caused the Aurora instance to run out of memory.
- Improved stability of the database when the volume size is close to the 64 tebibyte (TiB) volume limit by reserving 160 GB of space for the recovery workflow to complete without a failover.
- Improved stability of the database when the hash join feature is enabled and the instance is low on memory.
- Fixed the free memory calculation to include swap memory space on T2 instances that caused them to reboot prematurely.
- Fixed an issue in the query cache where the "Too many connections" error could cause a reboot.

Integration of MySQL community edition bug fixes

- [CVE-2018-2696](#)
- [CVE-2015-4737](#)
- Bug #19929406: `HANDLE_FATAL_SIGNAL (SIG=11) IN __MEMMOVE_SSSE3_BACK FROM STRING::COPY`
- Bug #17059925: For `UNION` statements, the rows-examined value was calculated incorrectly. This was manifested as too-large values for the `ROWS_EXAMINED` column of Performance Schema statement tables (such as `events_statements_current`).
- Bug #11827369: Some queries with `SELECT ... FROM DUAL` nested subqueries raised an assertion.
- Bug #16311231: Incorrect results were returned if a query contained a subquery in an `IN` clause that contained an `XOR` operation in the `WHERE` clause.

Aurora MySQL database engine updates 2019-06-05 (version 1.19.2)

Version: 1.19.2

Aurora MySQL 1.19.2 is generally available. All new Aurora MySQL database clusters with MySQL 5.6 compatibility, including those restored from snapshots, can be created with 1.17.8, 1.19.0, 1.19.1, or 1.19.2. You have the option, but are not required, to upgrade existing database clusters to Aurora MySQL 1.19.2. To use an older version, you can create new database clusters in Aurora MySQL 1.14.4, Aurora MySQL 1.15.1, Aurora MySQL 1.16, Aurora MySQL 1.17.8, or Aurora MySQL 1.18. You can do so using the AWS CLI or the Amazon RDS API and specifying the engine version.

If you have any questions or concerns, AWS Support is available on the community forums and through [AWS Premium Support](#). For more information, see [Maintaining an Amazon Aurora DB cluster \(p. 443\)](#).

Note

This version is currently not available in the AWS GovCloud (US-West) [us-gov-west-1], Europe (Stockholm) [eu-north-1], China (Ningxia) [cn-northwest-1], and Asia Pacific (Hong Kong) [ap-east-1] AWS Regions. There will be a separate announcement once it is made available.

Note

The procedure to upgrade your DB cluster has changed. For more information, see [Upgrading the minor version or patch level of an Aurora MySQL DB cluster \(p. 1088\)](#).

Improvements

- Fixed an issue that could cause failures when loading data into Aurora from Amazon S3.
- Fixed an issue that could cause failures when uploading data from Aurora to Amazon S3.
- Fixed an issue that created zombie sessions left in a killed state.
- Fixed an issue that caused aborted connections when handling an error in network protocol management.
- Fixed an issue that could cause a crash when dealing with partitioned tables.
- Fixed an issue related to binlog replication of trigger creation.

Aurora MySQL database engine updates 2019-05-09 (version 1.19.1)

Version: 1.19.1

Aurora MySQL 1.19.1 is generally available. All new Aurora MySQL database clusters with MySQL 5.6 compatibility, including those restored from snapshots, can be created with 1.17.8, 1.19.0, or 1.19.1. You have the option, but are not required, to upgrade existing database clusters to Aurora MySQL 1.19.1. To use an older version, you can create new database clusters in Aurora MySQL 1.14.4, Aurora MySQL 1.15.1, Aurora MySQL 1.16, Aurora MySQL 1.17.8, or Aurora MySQL 1.18. You can do so using the AWS CLI or the Amazon RDS API and specifying the engine version.

If you have any questions or concerns, AWS Support is available on the community forums and through [AWS Premium Support](#). For more information, see [Maintaining an Amazon Aurora DB cluster \(p. 443\)](#).

Note

This version is currently not available in the AWS GovCloud (US-West) [us-gov-west-1] and China (Beijing) [cn-north-1] regions. There will be a separate announcement once it is made available.

Note

The procedure to upgrade your DB cluster has changed. For more information, see [Upgrading the minor version or patch level of an Aurora MySQL DB cluster \(p. 1088\)](#).

Improvements

- Fixed a bug in binlog replication that can cause an issue on Aurora instances configured as binlog worker.

- Fixed an error in handling certain kinds of `ALTER TABLE` commands.
- Fixed an issue with aborted connections because of an error in network protocol management.

Aurora MySQL database engine updates 2019-02-07 (version 1.19.0)

Version: 1.19.0

Aurora MySQL 1.19.0 is generally available. All new Aurora MySQL database clusters with MySQL 5.6 compatibility, including those restored from snapshots, can be created with 1.17.8 or 1.19.0. You have the option, but are not required, to upgrade existing database clusters to Aurora MySQL 1.19.0. To use an older version, you can create new database clusters in Aurora MySQL 1.14.4, Aurora MySQL 1.15.1, Aurora MySQL 1.16, Aurora MySQL 1.17.8, or Aurora MySQL 1.18.0. You can do so using the AWS CLI or the Amazon RDS API and specifying the engine version.

If you have any questions or concerns, AWS Support is available on the community forums and through [AWS Premium Support](#). For more information, see [Maintaining an Amazon Aurora DB cluster \(p. 443\)](#).

Note

This version is currently not available in the AWS GovCloud (US-West) [us-gov-west-1] and China (Beijing) [cn-north-1] regions. There will be a separate announcement once it is made available.

Note

The procedure to upgrade your DB cluster has changed. For more information, see [Upgrading the minor version or patch level of an Aurora MySQL DB cluster \(p. 1088\)](#).

Features

- **Aurora Version Selector** - Starting with Aurora MySQL 1.19.0, you can choose from among multiple versions of MySQL 5.6 compatible Aurora on the Amazon RDS console. For more information, see [Checking or specifying Aurora MySQL engine versions through AWS \(p. 1083\)](#).

Improvements

- Fixed a stability issue related to the `CHECK TABLE` query on an Aurora Replica.
- Introduced a new global user variable `aurora_disable_hash_join` to disable Hash Join.
- Fixed a stability issue when generating the output row during multiple table hash join.
- Fixed an issue that returned a wrong result because of a plan change during Hash Join applicability check.
- Zero Downtime Patching is supported with long running transactions. This enhancement will come into effect when upgrading from version 1.19 to a higher one.
- Zero Downtime Patching is now supported when binlog is enabled. This enhancement will come into effect when upgrading from version 1.19 to a higher one.
- Fixed an issue that caused a spike in CPU utilization on the Aurora Replica unrelated to the workload.
- Fixed a race condition in the lock manager that resulted in a database restart.
- Fixed a race condition in the lock manager component to improve stability of Aurora instances.
- Improved stability of the deadlock detector inside the lock manager component.
- `INSERT` operation on a table is prohibited if InnoDB detects that the index is corrupted.
- Fixed a stability issue in Fast DDL.
- Improved Aurora stability by reducing the memory consumption in scan batching for single-row subquery.

- Fixed a stability issue that occurred after a foreign key was dropped while the system variable `foreign_key_checks` is set to 0.
- Fixed an issue in the Out Of Memory Avoidance feature that erroneously overrode changes to the `table_definition_cache` value made by the user.
- Fixed stability issues in the Out Of Memory Avoidance feature.
- Fixed an issue that set `query_time` and `lock_time` in `slow_query_log` to garbage values.
- Fixed a parallel query stability issue triggered by improper handling of string collation internally.
- Fixed a parallel query stability issue triggered by a secondary index search.
- Fixed a parallel query stability issue triggered by a multi-table update.

Integration of MySQL community edition bug fixes

- BUG #32917: DETECT ORPHAN TEMP-POOL FILES, AND HANDLE GRACEFULLY
- BUG #63144 CREATE TABLE IF NOT EXISTS METADATA LOCK IS TOO RESTRICTIVE

Aurora MySQL database engine updates 2018-09-20

Version: 1.18.0

Aurora MySQL 1.18.0 is generally available. All new Aurora MySQL parallel query clusters with MySQL 5.6 compatibility, including those restored from snapshots, will be created in Aurora MySQL 1.18.0. You have the option, but are not required, to upgrade existing parallel query clusters to Aurora MySQL 1.18.0. You can create new DB clusters in Aurora MySQL 1.14.4, Aurora MySQL 1.15.1, Aurora MySQL 1.16, or Aurora MySQL 1.17.6. You can do so using the AWS CLI or the Amazon RDS API and specifying the engine version.

With version 1.18.0 of Aurora MySQL, we are using a cluster patching model where all nodes in an Aurora DB cluster are patched at the same time.

Important

Aurora MySQL 1.18.0 only applies to Aurora parallel query clusters. If you upgrade a provisioned 5.6.10a cluster, the resulting version is 1.17.8. If you upgrade a parallel query 5.6.10a cluster, the resulting version is 1.18.0.

If you have any questions or concerns, AWS Support is available on the community forums and through [AWS Premium Support](#). For more information, see [Maintaining an Amazon Aurora DB cluster \(p. 443\)](#).

Features

- **Parallel Query** is available with this release, for new clusters and restored snapshots. Aurora MySQL parallel query is an optimization that parallelizes some of the I/O and computation involved in processing data-intensive queries. The work that is parallelized includes retrieving rows from storage, extracting column values, and determining which rows match the conditions in the `WHERE` clause and `join` clauses. This data-intensive work is delegated (in database optimization terms, pushed down) to multiple nodes in the Aurora distributed storage layer. Without parallel query, each query brings all the scanned data to a single node within the Aurora MySQL cluster (the head node) and performs all the query processing there.
 - When the parallel query feature is enabled, the Aurora MySQL engine automatically determines when queries can benefit, without requiring SQL changes such as hints or table attributes.

For more information, see [Working with parallel query for Amazon Aurora MySQL \(p. 881\)](#).

- **OOM Avoidance:** This feature monitors the system memory and tracks memory consumed by various components of the database. Once the system runs low on memory, it performs a list of actions to

release memory from various tracked components in an attempt to save the database from running into Out of Memory (OOM) and thereby avoiding a database restart. This best-effort feature is enabled by default for t2 instances and can be enabled on other instance classes via a new instance parameter named `aurora_oom_response`. The instance parameter takes a string of comma separated actions that an instance should take when its memory is low. Valid actions include "print", "tune", "decline", "kill_query" or any combination of these. Any empty string means there should be no actions taken and effectively renders the feature to be disabled. Note that the default actions for the feature is "print, tune". Usage examples:

- "print" – Only prints the queries taking high amount of memory.
- "tune" – Tunes the internal table caches to release some memory back to the system.
- "decline" – Declines new queries once the instance is low on memory.
- "kill_query" – Kills the queries in descending order of memory consumption until the instance memory surfaces above the low threshold. Data definition language (DDL) statements are not killed.
- "print, tune" – Performs actions described for both "print" and "tune".
- "tune, decline, kill_query" – Performs the actions described for "tune", "decline", and "kill_query".

For information about handling out-of-memory conditions and other troubleshooting advice, see [Amazon Aurora MySQL out of memory issues \(p. 1817\)](#).

Aurora MySQL database engine updates 2020-03-05

Version: 1.17.9

Aurora MySQL 1.17.9 is generally available. Aurora MySQL 1.* versions are compatible with MySQL 5.6 and Aurora MySQL 2.* versions are compatible with MySQL 5.7.

Currently supported Aurora MySQL releases are 1.14.*, 1.15.*, 1.16.*, 1.17.*, 1.18.*, 1.19.*, 1.20.*, 1.21.*, 1.22.*, 2.01.*, 2.02.*, 2.03.*, 2.04.*, 2.05.*, 2.06.* and 2.07.*. You can restore the snapshot of an Aurora MySQL 1.* database into Aurora MySQL 1.17.9.

To create a cluster with an older version of Aurora MySQL, please specify the engine version through the RDS Console, the AWS CLI, or the Amazon RDS API.

Note

This version is currently not available in the following regions: AWS GovCloud (US-East) [us-gov-east-1], AWS GovCloud (US-West) [us-gov-west-1]. There will be a separate announcement once it is made available.

If you have any questions or concerns, AWS Support is available on the community forums and through [AWS Premium Support](#). For more information, see [Maintaining an Amazon Aurora DB cluster \(p. 443\)](#).

Improvements

High priority fixes:

- Fixed an issue of intermittent connection failures after certificate rotation.

Aurora MySQL database engine updates 2019-01-17

Version: 1.17.8

Aurora MySQL 1.17.8 is generally available. All new Aurora MySQL database clusters with MySQL 5.6 compatibility, including those restored from snapshots, will be created in Aurora MySQL 1.17.8. You have the option, but are not required, to upgrade existing database clusters to Aurora MySQL 1.17.8. To use

an older version, you can create new database clusters in Aurora MySQL 1.14.4, 1.15.1, 1.16, or 1.17.7. You can do so using the AWS CLI or the Amazon RDS API and specifying the engine version.

With version 1.17.8 of Aurora MySQL, we are using a cluster patching model where all nodes in an Aurora DB cluster are patched at the same time.

Note

This version is currently not available in the AWS GovCloud (US-West) [us-gov-west-1] and China (Beijing) [cn-north-1] regions. There will be a separate announcement once it is made available.

If you have any questions or concerns, AWS Support is available on the community forums and through [AWS Premium Support](#). For more information, see [Maintaining an Amazon Aurora DB cluster \(p. 443\)](#).

Improvements

- Fixed a performance issue that increased the CPU utilization on an Aurora Replica after a restart.
- Fixed a stability issue for `SELECT` queries that used hash join.

Integration of MySQL community edition bug fixes

- BUG #13418638: CREATE TABLE IF NOT EXISTS METADATA LOCK IS TOO RESTRICTIVE

Aurora MySQL database engine updates 2018-10-08

Version: 1.17.7

Aurora MySQL 1.17.7 is generally available. All new Aurora MySQL database clusters with MySQL 5.6 compatibility, including those restored from snapshots, will be created in Aurora MySQL 1.17.7. You have the option, but are not required, to upgrade existing database clusters to Aurora MySQL 1.17.7. To use an older version, you can create new database clusters in Aurora MySQL 1.14.4, 1.15.1, 1.16, or 1.17.6. You can do so using the AWS CLI or the Amazon RDS API and specifying the engine version.

With version 1.17.7 of Aurora MySQL, we are using a cluster patching model where all nodes in an Aurora DB cluster are patched at the same time.

Note

This version is currently not available in the AWS GovCloud (US-West) [us-gov-west-1] and China (Beijing) [cn-north-1] regions. There will be a separate announcement once it is made available.

If you have any questions or concerns, AWS Support is available on the community forums and through [AWS Premium Support](#). For more information, see [Maintaining an Amazon Aurora DB cluster \(p. 443\)](#).

Improvements

- The InnoDB status variable `innodb_buffer_pool_size` has been made publicly visible for the customers to modify.
- Fixed a stability issue on the Aurora cluster that occurred during failovers.
- Improved cluster availability by fixing a DDL recovery issue that occurred after an unsuccessful `TRUNCATE` operation.
- Fixed a stability issue related to the `mysql.innodb_table_stats` table update, triggered by DDL operations.
- Fixed Aurora Replica stability issues triggered during query cache invalidation after a DDL operation.
- Fixed a stability issue triggered by invalid memory access during periodic dictionary cache eviction in the background.

Integration of MySQL community edition bug fixes

- Bug #16208542: Drop index on a foreign key column leads to missing table.
- Bug #76349: memory leak in add_derived_key().
- Bug #16862316: For partitioned tables, queries could return different results depending on whether Index Merge was used.
- Bug #17588348: Queries using the index_merge optimization (see [Index merge optimization](#)) could return invalid results when run against tables that were partitioned by HASH.

Aurora MySQL database engine updates 2018-09-06

Version: 1.17.6

Aurora MySQL 1.17.6 is generally available. All new Aurora MySQL database clusters with MySQL 5.6 compatibility, including those restored from snapshots, will be created in Aurora MySQL 1.17.6. You have the option, but are not required, to upgrade existing database clusters to Aurora MySQL 1.17.6. To use an older version, you can create new database clusters in Aurora MySQL 1.14.4, 1.15.1, 1.16, or 1.17.5. You can do so using the AWS CLI or the Amazon RDS API and specifying the engine version.

With version 1.17.6 of Aurora MySQL, we are using a cluster patching model where all nodes in an Aurora DB cluster are patched at the same time.

Note

This version is currently not available in the AWS GovCloud (US-West) [us-gov-west-1] and China (Beijing) [cn-north-1] regions. There will be a separate announcement once it is made available.

If you have any questions or concerns, AWS Support is available on the community forums and through [AWS Premium Support](#). For more information, see [Maintaining an Amazon Aurora DB cluster \(p. 443\)](#).

Improvements

- Fixed a stability issue on the Aurora Reader for `SELECT` queries while the Aurora Writer is performing DDL operations on the same table.
- Fixed a stability issue caused by the creation and deletion of DDL logs for temporary tables that use Heap/Memory engine.
- Fixed a stability issue on the binlog worker when DDL statements are being replicated while the connection to the Binlog Master is unstable.
- Fixed a stability issue encountered while writing to the slow query log.
- Fixed an issue with the replica status table that exposed incorrect Aurora Reader lag information.

Integration of MySQL community edition bug fixes

- For an `ALTER TABLE` statement that renamed or changed the default value of a `BINARY` column, the alteration was done using a table copy and not in place. (Bug #67141, Bug #14735373, Bug #69580, Bug #17024290)
- An outer join between a regular table and a derived table that is implicitly groups could cause a server exit. (Bug #16177639)

Aurora MySQL database engine updates 2018-08-14

Version: 1.17.5

Aurora MySQL 1.17.5 is generally available. All new Aurora MySQL database clusters with MySQL 5.6 compatibility, including those restored from snapshots, will be created in Aurora MySQL 1.17.5. You have the option, but are not required, to upgrade existing database clusters to Aurora MySQL 1.17.5. To use an older version, you can create new database clusters in Aurora MySQL 1.14.4, 1.15.1, 1.16, or 1.17.4. You can do so using the AWS CLI or the Amazon RDS API and specifying the engine version.

With version 1.17.5 of Aurora MySQL, we are using a cluster patching model where all nodes in an Aurora DB cluster are patched at the same time.

Note

This version is currently not available in the AWS GovCloud (US-West) [us-gov-west-1] and China (Beijing) [cn-north-1] regions. There will be a separate announcement once it is made available.

If you have any questions or concerns, AWS Support is available on the community forums and through [AWS Premium Support](#). For more information, see [Maintaining an Amazon Aurora DB cluster \(p. 443\)](#).

Improvements

- Fixed an issue where an Aurora Writer might experience a restart after an Aurora cluster is patched using the Zero-Downtime Patching feature.

Aurora MySQL database engine updates 2018-08-07

Version: 1.17.4

Aurora MySQL 1.17.4 is generally available. All new Aurora MySQL database clusters with MySQL 5.6 compatibility, including those restored from snapshots, will be created in Aurora MySQL 1.17.4. You have the option, but are not required, to upgrade existing database clusters to Aurora MySQL 1.17.4. To use an older version, you can create new database clusters in Aurora MySQL 1.14.4, 1.15.1, 1.16, or 1.17.3. You can do so using the AWS CLI or the Amazon RDS API and specifying the engine version.

With version 1.17.4 of Aurora MySQL, we are using a cluster patching model where all nodes in an Aurora DB cluster are patched at the same time.

Note

This version is currently not available in the AWS GovCloud (US-West) [us-gov-west-1] and China (Beijing) [cn-north-1] regions. There will be a separate announcement once it is made available.

If you have any questions or concerns, AWS Support is available on the community forums and through [AWS Premium Support](#). For more information, see [Maintaining an Amazon Aurora DB cluster \(p. 443\)](#).

Improvements

- Replication improvements:
 - Reduced network traffic by not transmitting binlog records to cluster replicas. This improvement is enabled by default.
 - Reduced network traffic by compressing replication messages. This improvement is enabled by default for 8xlarge and 16xlarge instance classes. Such large instances can sustain a heavy volume of write traffic that results in substantial network traffic for replication messages.
- Fixes to the replica query cache.
- Fixed an issue where `ORDER BY LOWER(col_name)` could produce incorrect ordering while using the `utf8_bin` collation.
- Fixed an issue where DDL statements (especially `TRUNCATE TABLE`) could cause problems on Aurora replicas, including instability or missing tables.

- Fixed an issue where sockets are left in a half-open state when storage nodes are restarted.
- The following new DB cluster parameters are available:
 - `aurora_enable_zdr` – Allow connections opened on an Aurora Replica to stay active on replica restart.
 - `aurora_enable_replica_log_compression` – Enable compression of replication payloads to improve network bandwidth utilization between the master and Aurora Replicas.
 - `aurora_enable_repl_bin_log_filtering` – Enable filtering of replication records that are unusable by Aurora Replicas on the master.

Aurora MySQL database engine updates 2018-06-05

Version: 1.17.3

Aurora MySQL 1.17.3 is generally available. All new Aurora MySQL database clusters with MySQL 5.6 compatibility, including those restored from snapshots, will be created in Aurora MySQL 1.17.3. You have the option, but are not required, to upgrade existing database clusters to Aurora MySQL 1.17.3. You can create new database clusters in Aurora MySQL 1.14.4, Aurora MySQL 1.15.1, or Aurora MySQL 1.16. You can do so using the AWS CLI or the Amazon RDS API and specifying the engine version.

With version 1.17.3 of Aurora MySQL, we are using a cluster patching model where all nodes in an Aurora DB cluster are patched at the same time.

Note

This version is currently not available in the AWS GovCloud (US-West) [us-gov-west-1] and China (Beijing) [cn-north-1] regions. There will be a separate announcement once it is made available.

If you have any questions or concerns, AWS Support is available on the community forums and through [AWS Premium Support](#). For more information, see [Maintaining an Amazon Aurora DB cluster \(p. 443\)](#).

Improvements

- Fixed an issue where an Aurora Replica can restart when using optimistic cursor restores while reading records.
- Fixed an issue where an Aurora Writer restarts when trying to kill a MySQL session (`kill "<session id>"`) with performance schema enabled.
- Fixed an issue where an Aurora Writer restarts when computing a threshold for garbage collection.
- Fixed an issue where an Aurora Writer can occasionally restart when tracking Aurora Replica progress in log application.
- Fixed an issue with the Query Cache when auto-commit is off and that could potentially cause stale reads.

Aurora MySQL database engine updates 2018-04-27

Version: 1.17.2

Aurora MySQL 1.17.2 is generally available. All new Aurora MySQL database clusters with MySQL 5.6 compatibility, including those restored from snapshots, will be created in Aurora MySQL 1.17.2. You have the option, but are not required, to upgrade existing database clusters to Aurora MySQL 1.17.2. You can create new database clusters in Aurora MySQL 1.14.4, Aurora MySQL 1.15.1, or Aurora MySQL 1.16. You can do so using the AWS CLI or the Amazon RDS API and specifying the engine version.

With version 1.17.2 of Aurora MySQL, we are using a cluster patching model where all nodes in an Aurora DB cluster are patched at the same time.

If you have any questions or concerns, AWS Support is available on the community forums and through [AWS Premium Support](#). For more information, see [Maintaining an Amazon Aurora DB cluster \(p. 443\)](#).

Improvements

- Fixed an issue which was causing restarts during certain DDL partition operations.
- Fixed an issue which was causing support for invocation of AWS Lambda functions via native Aurora MySQL functions to be disabled.
- Fixed an issue with cache invalidation which was causing restarts on Aurora Replicas.
- Fixed an issue in lock manager which was causing restarts.

Aurora MySQL database engine updates 2018-03-23

Version: 1.17.1

Aurora MySQL 1.17.1 is generally available. All new database clusters, including those restored from snapshots, will be created in Aurora MySQL 1.17.1. You have the option, but are not required, to upgrade existing database clusters to Aurora MySQL 1.17.1. You can create new DB clusters in Aurora MySQL 1.15.1, Aurora MySQL 1.16, or Aurora MySQL 1.17. You can do so using the AWS CLI or the Amazon RDS API and specifying the engine version.

With version 1.17.1 of Aurora MySQL, we are using a cluster patching model where all nodes in an Aurora DB cluster are patched at the same time. This release fixes some known engine issues as well as regressions.

If you have any questions or concerns, AWS Support is available on the community forums and through [AWS Premium Support](#). For more information, see [Maintaining an Amazon Aurora DB cluster \(p. 443\)](#).

Note

There is an issue in the latest version of the Aurora MySQL engine. After upgrading to 1.17.1, the engine version is reported incorrectly as 1.17. If you upgraded to 1.17.1, you can confirm the upgrade by checking the **Maintenance** column for the DB cluster in the AWS Management Console. If it displays none, then the engine is upgraded to 1.17.1.

Improvements

- Fixed an issue in binary log recovery that resulted in longer recovery times for situations with large binary log index files which can happen if binary logs rotate very often.
- Fixed an issue in the query optimizer that generated an inefficient query plan for partitioned tables.
- Fixed an issue in the query optimizer due to which a range query resulted in a restart of the database engine.

Aurora MySQL database engine updates 2018-03-13

Version: 1.17

Aurora MySQL 1.17 is generally available. Aurora MySQL 1.x versions are only compatible with MySQL 5.6, and not MySQL 5.7. All new 5.6-compatible database clusters, including those restored from snapshots, will be created in Aurora 1.17. You have the option, but are not required, to upgrade existing database clusters to Aurora 1.17. You can create new DB clusters in Aurora 1.14.1, Aurora 1.15.1, or Aurora 1.16. You can do so using the AWS CLI or the Amazon RDS API and specifying the engine version.

With version 1.17 of Aurora, we are using a cluster patching model where all nodes in an Aurora DB cluster are patched at the same time. We support zero-downtime patching, which works on a best-effort

basis to preserve client connections through the patching process. For more information, see [Maintaining an Amazon Aurora DB cluster \(p. 443\)](#).

If you have any questions or concerns, AWS Support is available on the community forums and through [AWS Premium Support](#).

Zero-downtime patching

The zero-downtime patching (ZDP) feature attempts, on a best-effort basis, to preserve client connections through an engine patch. For more information about ZDP, see [Using zero-downtime patching \(p. 1091\)](#).

New features

- Aurora MySQL now supports lock compression, which optimizes the lock manager's memory usage. Starting in version 1.17, you can use this feature without enabling lab mode.

Improvements

- Fixed an issue predominantly seen on instances with fewer cores where a single core might have 100% CPU utilization even when the database is idle.
- Improved the performance of fetching binary logs from Aurora clusters.
- Fixed an issue where Aurora Replicas attempt to write table statistics to persistent storage, and crash.
- Fixed an issue where query cache did not work as expected on Aurora Replicas.
- Fixed a race condition in lock manager that resulted in an engine restart.
- Fixed an issue where locks taken by read-only, auto-commit transactions resulted in an engine restart.
- Fixed an issue where some queries are not written to the audit logs.
- Fixed an issue with recovery of certain partition maintenance operations on failover.

Integration of MySQL bug fixes

- LAST_INSERT_ID is replicated incorrectly if replication filters are used (Bug #69861)
- Query returns different results depending on whether INDEX_MERGE setting (Bug #16862316)
- Query proc re-execute of stored routine, inefficient query plan (Bug #16346367)
- INNODB FTS : Assert in FTS_CACHE_APPEND_DELETED_DOC_IDS (BUG #18079671)
- Assert RBT_EMPTY(INDEX_CACHE->WORDS) in ALTER TABLE CHANGE COLUMN (BUG #17536995)
- INNODB fulltext search doesn't find records when savepoints are involved (BUG #70333, BUG #17458835)

Aurora MySQL database engine updates 2017-12-11

Version: 1.16

Aurora MySQL 1.16 is generally available. All new database clusters, including those restored from snapshots, will be created in Aurora 1.16. You have the option, but are not required, to upgrade existing database clusters to Aurora 1.16. You can create new DB clusters in Aurora 1.14.1 or Aurora 1.15.1. You can do so using the AWS CLI or the Amazon RDS API and specifying the engine version.

With version 1.16 of Aurora, we are using a cluster patching model where all nodes in an Aurora DB cluster are patched at the same time. We are enabling zero-downtime patching, which works on a best-

effort basis to preserve client connections through the patching process. For more information, see [Maintaining an Amazon Aurora DB cluster \(p. 443\)](#).

If you have any questions or concerns, AWS Support is available on the community forums and through [AWS Premium Support](#).

Zero-downtime patching

The zero-downtime patching (ZDP) feature attempts, on a [best-effort](#) basis, to preserve client connections through an engine patch. For more information about ZDP, see [Using zero-downtime patching \(p. 1091\)](#).

New features

- Aurora MySQL now supports synchronous AWS Lambda invocations via the native function `lambda_sync()`. Also available is native function `lambda_async()`, which can be used as an alternative to the existing stored procedure for asynchronous Lambda invocation. For more information, see [Invoking a Lambda function from an Amazon Aurora MySQL DB cluster \(p. 1010\)](#).
- Aurora MySQL now supports hash joins to speed up equijoin queries. Aurora's cost-based optimizer can automatically decide when to use hash joins; you can also force their use in a query plan. For more information, see [Optimizing large Aurora MySQL join queries with hash joins \(p. 1038\)](#).
- Aurora MySQL now supports scan batching to speed up in-memory scan-oriented queries significantly. The feature boosts the performance of table full scans, index full scans, and index range scans by batch processing.

Improvements

- Fixed an issue where read replicas crashed when running queries on tables that have just been dropped on the master.
- Fixed an issue where restarting the writer on a database cluster with a very large number of `FULLTEXT` indexes results in longer than expected recovery.
- Fixed an issue where flushing binary logs causes `LOST_EVENTS` incidents in binlog events.
- Fixed stability issues with the scheduler when performance schema is enabled.
- Fixed an issue where a subquery that uses temporary tables could return partial results.

Integration of MySQL bug fixes

None

Aurora MySQL database engine updates 2017-11-20

Version: 1.15.1

Aurora MySQL 1.15.1 is generally available. All new database clusters, including those restored from snapshots, will be created in Aurora 1.15.1. You have the option, but are not required, to upgrade existing DB clusters to Aurora 1.15.1. You can create new DB clusters in Aurora 1.14.1. You can do so using the AWS CLI or the Amazon RDS API and specifying the engine version.

With version 1.15.1 of Aurora, we are using a cluster patching model where all nodes in an Aurora DB cluster are patched at the same time. We are enabling zero-downtime patching, which works on a best-effort basis to preserve client connections through the patching process. For more information, see [Maintaining an Amazon Aurora DB cluster \(p. 443\)](#).

If you have any questions or concerns, AWS Support is available on the community forums and through [AWS Premium Support](#). For more information, see [Maintaining an Amazon Aurora DB cluster \(p. 443\)](#).

Zero-downtime patching

The zero-downtime patching (ZDP) feature attempts, on a best-effort basis, to preserve client connections through an engine patch. For more information about ZDP, see [Using zero-downtime patching \(p. 1091\)](#).

Improvements

- Fixed an issue in the adaptive segment selector for a read request that would cause it to choose the same segment twice causing a spike in read latency under certain conditions.
- Fixed an issue that stems from an optimization in Aurora MySQL for the thread scheduler. This problem manifests itself into what are spurious errors while writing to the slow log, while the associated queries themselves perform fine.
- Fixed an issue with stability of read replicas on large (> 5 TB) volumes.
- Fixed an issue where worker thread count increases continuously due to a bogus outstanding connection count.
- Fixed an issue with table locks that caused long semaphore waits during insert workloads.
- Reverted the following MySQL bug fixes included in Aurora MySQL 1.15:
 - MySQL instance stalling "doing SYNC index" (Bug #73816)
 - Assert RBT_EMPTY(INDEX_CACHE->WORDS) in ALTER TABLE CHANGE COLUMN (Bug #17536995)
 - InnoDB Fulltext search doesn't find records when savepoints are involved (Bug #70333)

Integration of MySQL bug fixes

None

Aurora MySQL database engine updates 2017-10-24

Version: 1.15

Aurora MySQL 1.15 is generally available. All new database clusters, including those restored from snapshots, will be created in Aurora 1.15. You have the option, but are not required, to upgrade existing DB clusters to Aurora 1.15. You can create new DB clusters in Aurora 1.14.1. You can do so using the AWS CLI or the Amazon RDS API and specifying the engine version.

With version 1.15 of Aurora, we are using a cluster patching model where all nodes in an Aurora DB cluster are patched at the same time. Updates require a database restart, so you will experience 20 to 30 seconds of downtime, after which you can resume using your DB cluster or clusters. If your DB clusters are currently running Aurora 1.14 or Aurora 1.14.1, the zero-downtime patching feature in Aurora MySQL might allow client connections to your Aurora MySQL primary instance to persist through the upgrade, depending on your workload.

If you have any questions or concerns, AWS Support is available on the community forums and through [AWS Premium Support](#). For more information, see [Maintaining an Amazon Aurora DB cluster \(p. 443\)](#).

Zero-downtime patching

The zero-downtime patching (ZDP) feature attempts, on a best-effort basis, to preserve client connections through an engine patch. For more information about ZDP, see [Using zero-downtime patching \(p. 1091\)](#).

New features

- **Asynchronous Key Prefetch** – Asynchronous key prefetch (AKP) is a feature targeted to improve the performance of non-cached index joins, by prefetching keys in memory ahead of when they

are needed. The primary use case targeted by AKP is an index join between a small outer and large inner table, where the index is highly selective on the larger table. Also, when the Multi-Range Read (MRR) interface is enabled, AKP will be leveraged for a secondary to primary index lookup. Smaller instances which have memory constraints might in some cases be able to leverage AKP, given the right key cardinality. For more information, see [Optimizing Amazon Aurora indexed join queries with asynchronous key prefetch \(p. 1036\)](#).

- **Fast DDL** – We have extended the feature that was released in [Aurora 1.13 \(p. 1231\)](#) to operations that include default values. With this extension, Fast DDL is applicable for operations that add a nullable column at the end of a table, with or without default values. The feature remains under Aurora lab mode. For more information, see [Altering tables in Amazon Aurora using fast DDL \(p. 832\)](#).

Improvements

- Fixed a calculation error during optimization of WITHIN/CONTAINS spatial queries which previously resulted in an empty result set.
- Fixed SHOW VARIABLE command to show the updated `innodb_buffer_pool_size` parameter value whenever it is changed in the parameter group.
- Improved stability of primary instance during bulk insert into a table altered using Fast DDL when adaptive hash indexing is disabled and the record to be inserted is the first record of a page.
- Improved stability of Aurora when the user attempts to set `server_audit_events` DB cluster parameter value to `default`.
- Fixed an issue in which a database character set change for an ALTER TABLE statement that ran on the Aurora primary instance was not being replicated on the Aurora Replicas until they were restarted.
- Improved stability by fixing a race condition on the primary instance which previously allowed it to register an Aurora Replica even if the primary instance had closed its own volume.
- Improved performance of the primary instance during index creation on a large table by changing the locking protocol to enable concurrent data manipulation language (DML) statements during index build.
- Fixed InnoDB metadata inconsistency during ALTER TABLE RENAME query which improved stability. Example: When columns of table t1(c1, c2) are renamed cyclically to t1(c2,c3) within the same ALTER statement.
- Improved stability of Aurora Replicas for the scenario where an Aurora Replica has no active workload and the primary instance is unresponsive.
- Improved availability of Aurora Replicas for a scenario in which the Aurora Replica holds an explicit lock on a table and blocks the replication thread from applying any DDL changes received from the primary instance.
- Improved stability of the primary instance when a foreign key and a column are being added to a table from two separate sessions at the same time and Fast DDL has been enabled.
- Improved stability of the purge thread on the primary instance during a heavy write workload by blocking truncate of undo records until they have been purged.
- Improved stability by fixing the lock release order during commit process of transactions which drop tables.
- Fixed a defect for Aurora Replicas in which the DB instance could not complete startup and complained that port 3306 was already in use.
- Fixed a race condition in which a SELECT query run on certain information_schema tables (`innodb trx`, `innodb lock`, `innodb lock waits`) increased cluster instability.

Integration of MySQL bug fixes

- CREATE USER accepts plugin and password hash, but ignores the password hash (Bug #78033)

- The partitioning engine adds fields to the read bit set to be able to return entries sorted from a partitioned index. This leads to the join buffer will try to read unneeded fields. Fixed by not adding all partitioning fields to the read_set, but instead only sort on the already set prefix fields in the read_set. Added a DBUG_ASSERT that if doing key_cmp, at least the first field must be read (Bug #16367691).
- MySQL instance stalling "doing SYNC index" (Bug #73816)
- Assert RBT_EMPTY(INDEX_CACHE->WORDS) in ALTER TABLE CHANGE COLUMN (Bug #17536995)
- InnoDB Fulltext search doesn't find records when savepoints are involved (Bug #70333)

Aurora MySQL database engine updates: 2018-03-13

Version: 1.14.4

Aurora MySQL 1.14.4 is generally available. You can create new DB clusters in Aurora 1.14.4, using the AWS CLI or the Amazon RDS API and specifying the engine version. You have the option, but are not required, to upgrade existing 1.14.x DB clusters to Aurora 1.14.4.

With version 1.14.4 of Aurora, we are using a cluster-patching model where all nodes in an Aurora DB cluster are patched at the same time. We support zero-downtime patching, which works on a best-effort basis to preserve client connections through the patching process. For more information, see [Maintaining an Amazon Aurora DB cluster \(p. 443\)](#).

If you have any questions or concerns, AWS Support is available on the community forums and through [AWS Premium Support](#). For more information, see [Maintaining an Amazon Aurora DB cluster \(p. 443\)](#).

Zero-downtime patching

The zero-downtime patching (ZDP) feature attempts, on a [best-effort](#) basis, to preserve client connections through an engine patch. For more information about ZDP, see [Using zero-downtime patching \(p. 1091\)](#).

New features

- Aurora MySQL now supports db.r4 instance classes.

Improvements

- Fixed an issue where `LOST_EVENTS` were generated when writing large binlog events.

Integration of MySQL bug fixes

- Ignorable events don't work and are not tested (Bug #74683)
- NEW->OLD ASSERT FAILURE 'GTID_MODE > 0' (Bug #20436436)

Aurora MySQL database engine updates: 2017-09-22

Version: 1.14.1

Aurora MySQL 1.14.1 is generally available. All new database clusters, including those restored from snapshots, will be created in Aurora MySQL 1.14.1. Aurora MySQL 1.14.1 is also a mandatory upgrade for existing Aurora MySQL DB clusters. For more information, see [Announcement: Extension to mandatory upgrade schedule for Amazon Aurora](#) on the AWS Developer Forums website.

With version 1.14.1 of Aurora MySQL, we are using a cluster patching model where all nodes in an Aurora MySQL DB cluster are patched at the same time. Updates require a database restart, so you will

experience 20 to 30 seconds of downtime, after which you can resume using your DB cluster or clusters. If your DB clusters are currently running version 1.13 or greater, the zero-downtime patching feature in Aurora MySQL might allow client connections to your Aurora MySQL primary instance to persist through the upgrade, depending on your workload.

If you have any questions or concerns, AWS Support is available on the community forums and through [AWS Premium Support](#).

Improvements

- Fixed race conditions associated with inserts and purge to improve the stability of the Fast DDL feature, which remains in Aurora MySQL lab mode.

Aurora MySQL database engine updates: 2017-08-07

Version: 1.14

Aurora MySQL 1.14 is generally available. All new database clusters, including those restored from snapshots, will be created in Aurora MySQL 1.14. Aurora MySQL 1.14 is also a mandatory upgrade for existing Aurora MySQL DB clusters. We will send a separate announcement with the timeline for deprecating earlier versions of Aurora MySQL.

With version 1.14 of Aurora MySQL, we are using a cluster patching model where all nodes in an Aurora DB cluster are patched at the same time. Updates require a database restart, so you will experience 20 to 30 seconds of downtime, after which you can resume using your DB cluster or clusters. If your DB clusters are currently running version 1.13, Aurora's zero-downtime patching feature may allow client connections to your Aurora primary instance to persist through the upgrade, depending on your workload.

If you have any questions or concerns, AWS Support is available on the community forums and through [AWS Premium Support](#).

Zero-downtime patching

The zero-downtime patching (ZDP) feature attempts, on a best-effort basis, to preserve client connections through an engine patch. For more information about ZDP, see [Using zero-downtime patching \(p. 1091\)](#).

Improvements

- Fixed an incorrect "record not found" error when a record is found in the secondary index but not in the primary index.
- Fixed a stability issue that can occur due to a defensive assertion (added in 1.12) that was too strong in the case when an individual write spans over 32 pages. Such a situation can occur, for instance, with large BLOB values.
- Fixed a stability issue because of inconsistencies between the tablespace cache and the dictionary cache.
- Fixed an issue in which an Aurora Replica becomes unresponsive after it has exceeded the maximum number of attempts to connect to the primary instance. An Aurora Replica now restarts if the period of inactivity is more than the heartbeat time period used for health check by the primary instance.
- Fixed a livelock that can occur under very high concurrency when one connection tries to acquire an exclusive meta data lock (MDL) while issuing a command, such as `ALTER TABLE`.
- Fixed a stability issue in an Aurora Read Replica in the presence of logical/parallel read ahead.
- Improved `LOAD FROM S3` in two ways:

- 1. Better handling of Amazon S3 timeout errors by using the SDK retry in addition to the existing retry.
- 2. Performance optimization when loading very big files or large numbers of files by caching and reusing client state.
- Fixed the following stability issues with Fast DDL for `ALTER TABLE` operations:
 1. When the `ALTER TABLE` statement has multiple `ADD COLUMN` commands and the column names are not in ascending order.
 2. When the name string of the column to be updated and its corresponding name string, fetched from the internal system table, differs by a null terminating character (/0).
 3. Under certain B-tree split operations.
 4. When the table has a variable length primary key.
- Fixed a stability issue with Aurora Replicas when it takes too long to make its Full Text Search (FTS) index cache consistent with that of the primary instance. This can happen if a large portion of the newly created FTS index entries on the primary instance have not yet been flushed to disk.
- Fixed a stability issue that can happen during index creation.
- New infrastructure that tracks memory consumption per connection and associated telemetry that will be used for building out Out-Of-Memory (OOM) avoidance strategies.
- Fixed an issue where `ANALYZE TABLE` was incorrectly allowed on Aurora Replicas. This has now been blocked.
- Fixed a stability issue caused by a rare deadlock as a result of a race condition between logical read-ahead and purge.

Integration of MySQL bug fixes

- A full-text search combined with derived tables (subqueries in the `FROM` clause) caused a server exit. Now, if a full-text operation depends on a derived table, the server produces an error indicating that a full-text search cannot be done on a materialized table. (Bug #68751, Bug #16539903)

Aurora MySQL database engine updates: 2017-05-15

Version: 1.13

Note

We enabled a new feature - `SELECT INTO OUTFILE S3` - in Aurora MySQL version 1.13 after the initial release, and have updated the release notes to reflect that change.

Aurora MySQL 1.13 is generally available. All new database clusters, including those restored from snapshots, will be created in Aurora MySQL 1.13. You have the option, but are not required, to upgrade existing database clusters to Aurora MySQL 1.13. With version 1.13 of Aurora, we are using a cluster patching model where all nodes in an Aurora DB cluster are patched at the same time. We are enabling zero-downtime patching, which works on a best-effort basis to preserve client connections through the patching process. For more information, see [Maintaining an Amazon Aurora DB cluster \(p. 443\)](#).

Zero-downtime patching

The zero-downtime patching (ZDP) feature attempts, on a best-effort basis, to preserve client connections through an engine patch. For more information about ZDP, see [Using zero-downtime patching \(p. 1091\)](#).

New features:

- **`SELECT INTO OUTFILE S3`** – Aurora MySQL now allows you to upload the results of a query to one or more files in an Amazon S3 bucket. For more information, see [Saving data from an Amazon Aurora MySQL DB cluster into text files in an Amazon S3 bucket \(p. 1004\)](#).

Improvements:

- Implemented truncation of CSV format log files at engine startup to avoid long recovery time. The `general_log_backup`, `general_log`, `slow_log_backup`, and `slow_log` tables now don't survive a database restart.
- Fixed an issue where migration of a database named `test` would fail.
- Improved stability in the lock manager's garbage collector by reusing the correct lock segments.
- Improved stability of the lock manager by removing invalid assertions during deadlock detection algorithm.
- Re-enabled asynchronous replication, and fixed an associated issue which caused incorrect replica lag to be reported under no-load or read-only workload. The replication pipeline improvements that were introduced in version 1.10. These improvements were introduced in order to apply log stream updates to the buffer cache of an Aurora Replica, which helps to improve read performance and stability on Aurora Replicas.
- Fixed an issue where `autocommit=OFF` leads to scheduled events being blocked and long transactions being held open until the server reboots.
- Fixed an issue where general, audit, and slow query logs could not log queries handled by asynchronous commit.
- Improved the performance of the logical read ahead (LRA) feature by up to 2.5 times. This was done by allowing pre-fetches to continue across intermediate pages in a B-tree.
- Added parameter validation for audit variables to trim unnecessary spaces.
- Fixed a regression, introduced in Aurora MySQL version 1.11, in which queries can return incorrect results when using the `SQL_CALC_FOUND_ROWS` option and invoking the `FOUND_ROWS()` function.
- Fixed a stability issue when the Metadata Lock list was incorrectly formed.
- Improved stability when `sql_mode` is set to `PAD_CHAR_TO_FULL_LENGTH` and the command `SHOW FUNCTION STATUS WHERE Db='string'` is executed.
- Fixed a rare case when instances would not come up after Aurora version upgrade because of a false volume consistency check.
- Fixed the performance issue, introduced in Aurora MySQL version 1.12, where the performance of the Aurora writer was reduced when users have a large number of tables.
- Improved stability issue when the Aurora writer is configured as a binlog worker and the number of connections approaches 16,000.
- Fixed a rare issue where an Aurora Replica could restart when a connection gets blocked waiting for Metadata Lock when running DDL on the Aurora master.

Integration of MySQL bug fixes

- With an empty InnoDB table, it's not possible to decrease the `auto_increment` value using an `ALTER TABLE` statement, even when the table is empty. (Bug #69882)
- `MATCH() ... AGAINST` queries that use a long string as an argument for `AGAINST()` could result in an error when run on an InnoDB table with a full-text search index. (Bug #17640261)
- Handling of `SQL_CALC_FOUND_ROWS` in combination with `ORDER BY` and `LIMIT` could lead to incorrect results for `FOUND_ROWS()`. (Bug #68458, Bug # 16383173)
- `ALTER TABLE` does not allow to change nullability of the column if foreign key exists. (Bug #77591)

Aurora MySQL database engine updates: 2017-04-05

Version: 1.12

Aurora MySQL 1.12 is now the preferred version for the creation of new DB clusters, including restores from snapshots.

This is not a mandatory upgrade for existing clusters. You will have the option to upgrade existing clusters to version 1.12 after we complete the fleet-wide patch to 1.11 (see [Aurora 1.11 release notes \(p. 1234\)](#) and corresponding [forum announcement](#)). With version 1.12 of Aurora, we are using a cluster patching model where all nodes in an Aurora DB cluster are patched at the same time. For more information, see [Maintaining an Amazon Aurora DB cluster \(p. 443\)](#).

New features

- **Fast DDL** – Aurora MySQL now allows you to execute an ALTER TABLE *tbl_name* ADD COLUMN *col_name column_definition* operation nearly instantaneously. The operation completes without requiring the table to be copied and without materially impacting other DML statements. Since it does not consume temporary storage for a table copy, it makes DDL statements practical even for large tables on small instance classes. Fast DDL is currently only supported for adding a nullable column, without a default value, at the end of a table. This feature is currently available in Aurora lab mode. For more information, see [Altering tables in Amazon Aurora using fast DDL \(p. 832\)](#).
- **Show volume status** – We have added a new monitoring command, SHOW VOLUME STATUS, to display the number of nodes and disks in a volume. For more information, see [Displaying volume status for an Aurora MySQL DB cluster \(p. 837\)](#).

Improvements

- Implemented changes to lock compression to further reduce memory allocated per lock object. This improvement is available in lab mode.
- Fixed an issue where the `trx_active_transactions` metric decrements rapidly even when the database is idle.
- Fixed an invalid error message regarding fault injection query syntax when simulating failure in disks and nodes.
- Fixed multiple issues related to race conditions and dead latches in the lock manager.
- Fixed an issue causing a buffer overflow in the query optimizer.
- Fixed a stability issue in Aurora read replicas when the underlying storage nodes experience low available memory.
- Fixed an issue where idle connections persisted past the `wait_timeout` parameter setting.
- Fixed an issue where `query_cache_size` returns an unexpected value after reboot of the instance.
- Fixed a performance issue that is the result of a diagnostic thread probing the network too often in the event that writes are not progressing to storage.

Integration of MySQL bug fixes

- Reloading a table that was evicted while empty caused an AUTO_INCREMENT value to be reset. (Bug #21454472, Bug #77743)
- An index record was not found on rollback due to inconsistencies in the `purge_node_t` structure. The inconsistency resulted in warnings and error messages such as "error in sec index entry update", "unable to purge a record", and "tried to purge sec index entry not marked for deletion". (Bug #19138298, Bug #70214, Bug #21126772, Bug #21065746)
- Wrong stack size calculation for `qsort` operation leads to stack overflow. (Bug #73979)
- Record not found in an index upon rollback. (Bug #70214, Bug #72419)
- ALTER TABLE add column TIMESTAMP on update CURRENT_TIMESTAMP inserts ZERO-datas (Bug #17392)

Aurora MySQL database engine updates: 2017-02-23

Version: 1.11

We will patch all Aurora MySQL DB clusters with the latest version over a short period following the release. DB clusters are patched using the legacy procedure with a downtime of about 5-30 seconds.

Patching occurs during the system maintenance window that you have specified for each of your database instances. You can view or change this window using the AWS Management Console. For more information, see [Maintaining an Amazon Aurora DB cluster \(p. 443\)](#).

Alternatively, you can apply the patch immediately in the AWS Management Console by choosing a DB cluster, choosing **Cluster Actions**, and then choosing **Upgrade Now**.

With version 1.11 of Aurora MySQL, we are using a cluster patching model where all nodes in an Aurora DB cluster are patched at the same time.

New features

- **MANIFEST option for LOAD DATA FROM S3** – LOAD DATA FROM S3 was released in version 1.8. The options for this command have been expanded, and you can now specify a list of files to be loaded into an Aurora DB cluster from Amazon S3 by using a manifest file. This makes it easy to load data from specific files in one or more locations, as opposed to loading data from a single file by using the FILE option or loading data from multiple files that have the same location and prefix by using the PREFIX option. The manifest file format is the same as that used by Amazon Redshift. For more information about using LOAD DATA FROM S3 with the MANIFEST option, see [Using a manifest to specify data files to load \(p. 1000\)](#).
- **Spatial indexing enabled by default** – This feature was released in lab mode in version 1.10, and is now turned on by default. Spatial indexing improves query performance on large datasets for queries that use spatial data. For more information about using spatial indexing, see [Amazon Aurora MySQL and spatial data \(p. 747\)](#).
- **Advanced Auditing timing change** – This feature was released in version 1.10.1 to provide a high-performance facility for auditing database activity. In this release, the precision of audit log timestamps has been changed from one second to one microsecond. The more accurate timestamps allow you to better understand when an audit event happened. For more information about audit, see [Using Advanced Auditing with an Amazon Aurora MySQL DB cluster \(p. 914\)](#).

Improvements

- Modified the `thread_handling` parameter to prevent you from setting it to anything other than `multiple-connections-per-thread`, which is the only supported model with Aurora's thread pool.
- Fixed an issue caused when you set either the `buffer_pool_size` or the `query_cache_size` parameter to be larger than the DB cluster's total memory. In this circumstance, Aurora sets the modified parameter to the default value, so the DB cluster can start up and not crash.
- Fixed an issue in the query cache where a transaction gets stale read results if the table is invalidated in another transaction.
- Fixed an issue where binlog files marked for deletion are removed after a small delay rather than right away.
- Fixed an issue where a database created with the name `tmp` is treated as a system database stored on ephemeral storage and not persisted to Aurora distributed storage.
- Modified the behavior of SHOW TABLES to exclude certain internal system tables. This change helps avoid an unnecessary failover caused by mysqldump locking all files listed in SHOW TABLES, which in turn prevents writes on the internal system table, causing the failover.

- Fixed an issue where an Aurora Replica incorrectly restarts when creating a temporary table from a query that invokes a function whose argument is a column of an InnoDB table.
- Fixed an issue related to a metadata lock conflict in an Aurora Replica node that causes the Aurora Replica to fall behind the primary DB cluster and eventually get restarted.
- Fixed a dead latch in the replication pipeline in reader nodes, which causes an Aurora Replica to fall behind and eventually get restarted.
- Fixed an issue where an Aurora Replica lags too much with encrypted volumes larger than 1 terabyte (TB).
- Improved Aurora Replica dead latch detection by using an improved way to read the system clock time.
- Fixed an issue where an Aurora Replica can restart twice instead of once following de-registration by the writer.
- Fixed a slow query performance issue on Aurora Replicas that occurs when transient statistics cause statistics discrepancy on non-unique index columns.
- Fixed an issue where an Aurora Replica can crash when a DDL statement is replicated on the Aurora Replica at the same time that the Aurora Replica is processing a related query.
- Changed the replication pipeline improvements that were introduced in version 1.10 from enabled by default to disabled by default. These improvements were introduced in order to apply log stream updates to the buffer cache of an Aurora Replica, and although this feature helps to improve read performance and stability on Aurora Replicas, it increases replica lag in certain workloads.
- Fixed an issue where the simultaneous occurrence of an ongoing DDL statement and pending Parallel Read Ahead on the same table causes an assertion failure during the commit phase of the DDL transaction.
- Enhanced the general log and slow query log to survive DB cluster restart.
- Fixed an out-of-memory issue for certain long running queries by reducing memory consumption in the ACL module.
- Fixed a restart issue that occurs when a table has non-spatial indexes, there are spatial predicates in the query, the planner chooses to use a non-spatial index, and the planner incorrectly pushes the spatial condition down to the index.
- Fixed an issue where the DB cluster restarts when there is a delete, update, or purge of very large geospatial objects that are stored externally (like LOBs).
- Fixed an issue where fault simulation using ALTER SYSTEM SIMULATE ... FOR INTERVAL isn't working properly.
- Fixed a stability issue caused by an invalid assertion on an incorrect invariant in the lock manager.
- Disabled the following two improvements to InnoDB Full-Text Search that were introduced in version 1.10 because they introduce stability issues for some demanding workloads:
 - Updating the cache only after a read request to an Aurora Replica in order to improve full-text search index cache replication speed.
 - Offloading the cache sync task to a separate thread as soon as the cache size crosses 10% of the total size, in order to avoid MySQL queries stalling for too long during FTS cache sync to disk. (Bugs #22516559, #73816).

Integration of MySQL bug fixes

- Running ALTER table DROP foreign key simultaneously with another DROP operation causes the table to disappear. (Bug #16095573)
- Some INFORMATION_SCHEMA queries that used ORDER BY did not use a filesort optimization as they did previously. (Bug #16423536)
- FOUND_ROWS () returns the wrong count of rows on a table. (Bug #68458)
- The server fails instead of giving an error when too many temp tables are open. (Bug #18948649)

Aurora MySQL database engine updates: 2017-01-12

Version: 1.10.1

Version 1.10.1 of Aurora MySQL is an opt-in version and is not used to patch your database instances. It is available for creating new Aurora instances and for upgrading existing instances. You can apply the patch by choosing a cluster in the [Amazon RDS console](#), choosing **Cluster Actions**, and then choosing **Upgrade Now**. Patching requires a database restart with downtime typically lasting 5-30 seconds, after which you can resume using your DB clusters. This patch is using a cluster patching model where all nodes in an Aurora cluster are patched at the same time.

New features

- **Advanced Auditing** – Aurora MySQL provides a high-performance Advanced Auditing feature, which you can use to audit database activity. For more information about enabling and using Advanced Auditing, see [Using Advanced Auditing with an Amazon Aurora MySQL DB cluster \(p. 914\)](#).

Improvements

- Fixed an issue with spatial indexing when creating a column and adding an index on it in the same statement.
- Fixed an issue where spatial statistics aren't persisted across DB cluster restart.

Aurora MySQL database engine updates: 2016-12-14

Version: 1.10

New features

- **Zero downtime patch** – This feature allows a DB instance to be patched without any downtime. That is, database upgrades are performed without disconnecting client applications, or rebooting the database. This approach increases the availability of your Aurora DB clusters during the maintenance window. Note that temporary data like that in the performance schema is reset during the upgrade process. This feature applies to service-delivered patches during a maintenance window as well as user-initiated patches.

When a patch is initiated, the service ensures there are no open locks, transactions or temporary tables, and then waits for a suitable window during which the database can be patched and restarted. Application sessions are preserved, although there is a drop in throughput while the patch is in progress (for approximately 5 seconds). If no suitable window can be found, then patching defaults to the standard patching behavior.

Zero downtime patching takes place on a best-effort basis, subject to certain limitations as described following:

- This feature is currently applicable for patching single-node DB clusters or writer instances in multi-node DB clusters.
- SSL connections are not supported in conjunction with this feature. If there are active SSL connections, Amazon Aurora MySQL won't perform a zero downtime patch, and instead will retry periodically to see if the SSL connections have terminated. If they have, zero downtime patching proceeds. If the SSL connections persist after more than a couple seconds, standard patching with downtime proceeds.
- The feature is available in Aurora release 1.10 and beyond. Going forward, we will identify any releases or patches that can't be applied by using zero downtime patching.
- This feature is not applicable if replication based on binary logging is active.

- **Spatial indexing** – Spatial indexing improves query performance on large datasets for queries that use spatial data. For more information about using spatial indexing, see [Amazon Aurora MySQL and spatial data \(p. 747\)](#).

This feature is disabled by default and can be activated by enabling Aurora lab mode. For information, see [Amazon Aurora MySQL lab mode \(p. 1032\)](#).

- **Replication pipeline improvements** – Aurora MySQL now uses an improved mechanism to apply log stream updates to the buffer cache of an Aurora Replica. This feature improves the read performance and stability on Aurora Replicas when there is a heavy write load on the master as well as a significant read load on the Replica. This feature is enabled by default.
- **Throughput improvement for workloads with cached reads** – Aurora MySQL now uses a latch-free concurrent algorithm to implement read views, which leads to better throughput for read queries served by the buffer cache. As a result of this and other improvements, Amazon Aurora MySQL can achieve throughput of up to 625K reads per second compared to 164K reads per second by MySQL 5.7 for a SysBench SELECT-only workload.
- **Throughput improvement for workloads with hot row contention** – Aurora MySQL uses a new lock release algorithm that improves performance, particularly when there is hot page contention (that is, many transactions contending for the rows on the same page). In tests with the TPC-C benchmark, this can result in up to 16x throughput improvement in transactions per minute relative to MySQL 5.7. This feature is disabled by default and can be activated by enabling Aurora lab mode. For information, see [Amazon Aurora MySQL lab mode \(p. 1032\)](#).

Improvements

- Full-text search index cache replication speed has been improved by updating the cache only after a read request to an Aurora Replica. This approach avoids any reads from disk by the replication thread.
- Fixed an issue where dictionary cache invalidation does not work on an Aurora Replica for tables that have a special character in the database name or table name.
- Fixed a `STUCK IO` issue during data migration for distributed storage nodes when storage heat management is enabled.
- Fixed an issue in the lock manager where an assertion check fails for the transaction lock wait thread when preparing to rollback or commit a transaction.
- Fixed an issue when opening a corrupted dictionary table by correctly updating the reference count to the dictionary table entries.
- Fixed a bug where the DB cluster minimum read point can be held by slow Aurora Replicas.
- Fixed a potential memory leak in the query cache.
- Fixed a bug where an Aurora Replica places a row-level lock on a table when a query is used in an `IF` statement of a stored procedure.

Integration of MySQL bug fixes

- UNION of derived tables returns wrong results with '1=0/false'-clauses. (Bug #69471)
- Server crashes in `ITEM_FUNC_GROUP_CONCAT::FIX_FIELDS` on 2nd execution of stored procedure. (Bug #20755389)
- Avoid MySQL queries from stalling for too long during FTS cache sync to disk by offloading the cache sync task to a separate thread, as soon as the cache size crosses 10% of the total size. (Bug #22516559, #73816)

Aurora MySQL database engine updates: 2016-11-10

Version: 1.9.0, 1.9.1

New features

- **Improved index build** – The implementation for building secondary indexes now operates by building the index in a bottom-up fashion, which eliminates unnecessary page splits. This can reduce the time needed to create an index or rebuild a table by up to 75% (based on an db.r3.8xlarge DB instance class). This feature was in lab mode in Aurora MySQL version 1.7 and is enabled by default in Aurora version 1.9 and later. For information, see [Amazon Aurora MySQL lab mode \(p. 1032\)](#).
- **Lock compression (lab mode)** – This implementation significantly reduces the amount of memory that lock manager consumes by up to 66%. Lock manager can acquire more row locks without encountering an out-of-memory exception. This feature is disabled by default and can be activated by enabling Aurora lab mode. For information, see [Amazon Aurora MySQL lab mode \(p. 1032\)](#).
- **Performance schema** – Aurora MySQL now includes support for performance schema with minimal impact on performance. In our testing using SysBench, enabling performance schema could degrade MySQL performance by up to 60%.

SysBench testing of an Aurora DB cluster showed an impact on performance that is 4x less than MySQL. Running the db.r3.8xlarge DB instance class resulted in 100K SQL writes/sec and over 550K SQL reads/sec, even with performance schema enabled.

- **Hot row contention improvement** – This feature reduces CPU utilization and increases throughput when a small number of hot rows are accessed by a large number of connections. This feature also eliminates `error 188` when there is hot row contention.
- **Improved out-of-memory handling** – When non-essential, locking SQL statements are executed and the reserved memory pool is breached, Aurora forces rollback of those SQL statements. This feature frees memory and prevents engine crashes due to out-of-memory exceptions.
- **Smart read selector** – This implementation improves read latency by choosing the optimal storage segment among different segments for every read, resulting in improved read throughput. SysBench testing has shown up to a 27% performance increase for write workloads .

Improvements

- Fixed an issue where an Aurora Replica encounters a shared lock during engine start up.
- Fixed a potential crash on an Aurora Replica when the read view pointer in the purge system is NULL.

Aurora MySQL database engine updates: 2016-10-26

Version: 1.8.1

Improvements

- Fixed an issue where bulk inserts that use triggers that invoke AWS Lambda procedures fail.
- Fixed an issue where catalog migration fails when autocommit is off globally.
- Resolved a connection failure to Aurora when using SSL and improved Diffie-Hellman group to deal with LogJam attacks.

Integration of MySQL bug fixes

- OpenSSL changed the Diffie-Hellman key length parameters due to the LogJam issue. (Bug #18367167)

Aurora MySQL database engine updates: 2016-10-18

Version: 1.8

New features

- **AWS Lambda integration** – You can now asynchronously invoke an AWS Lambda function from an Aurora DB cluster using the `mysql.lambda_async` procedure. For more information, see [Invoking a Lambda function from an Amazon Aurora MySQL DB cluster \(p. 1010\)](#).
- **Load data from Amazon S3** – You can now load text or XML files from an Amazon S3 bucket into your Aurora DB cluster using the `LOAD DATA FROM S3` or `LOAD XML FROM S3` commands. For more information, see [Loading data into an Amazon Aurora MySQL DB cluster from text files in an Amazon S3 bucket \(p. 997\)](#).
- **Catalog migration** – Aurora now persists catalog metadata in the cluster volume to support versioning. This enables seamless catalog migration across versions and restores.
- **Cluster-level maintenance and patching** – Aurora now manages maintenance updates for an entire DB cluster. For more information, see [Maintaining an Amazon Aurora DB cluster \(p. 443\)](#).

Improvements

- Fixed an issue where an Aurora Replica crashes when not granting a metadata lock to an inflight DDL table.
- Allowed Aurora Replicas to modify non-InnoDB tables to facilitate rotation of the slow and general log CSV files where `log_output=TABLE`.
- Fixed a lag when updating statistics from the primary instance to an Aurora Replica. Without this fix, the statistics of the Aurora Replica can get out of sync with the statistics of the primary instance and result in a different (and possibly under-performing) query plan on an Aurora Replica.
- Fixed a race condition that ensures that an Aurora Replica does not acquire locks.
- Fixed a rare scenario where an Aurora Replica that registers or de-registers with the primary instance could fail.
- Fixed a race condition that could lead to a deadlock on `db.r3.large` instances when opening or closing a volume.
- Fixed an out-of-memory issue that can occur due to a combination of a large write workload and failures in the Aurora Distributed Storage service.
- Fixed an issue with high CPU consumption because of the purge thread spinning in the presence of a long-running transaction.
- Fixed an issue when running information schema queries to get information about locks under heavy load.
- Fixed an issue with a diagnostics process that could in rare cases cause Aurora writes to storage nodes to stall and restart/fail-over.
- Fixed a condition where a successfully created table might be deleted during crash recovery if the crash occurred while a `CREATE TABLE [if not exists]` statement was being handled.
- Fixed a case where the log rotation procedure is broken when the general log and slow log are not stored on disk using catalog mitigation.
- Fixed a crash when a user creates a temporary table within a user defined function, and then uses the user defined function in the select list of the query.
- Fixed a crash that occurred when replaying GTID events. GTID is not supported by Aurora MySQL.

Integration of MySQL bug fixes:

- When dropping all indexes on a column with multiple indexes, InnoDB failed to block a `DROP INDEX` operation when a foreign key constraint requires an index. (Bug #16896810)
- Solve add foreign key constraint crash. (Bug #16413976)
- Fixed a crash when fetching a cursor in a stored procedure, and analyzing or flushing the table at the same time. (Bug # 18158639)

- Fixed an auto-increment bug when a user alters a table to change the AUTO_INCREMENT value to less than the maximum auto-increment column value. (Bug # 16310273)

Aurora MySQL database engine updates: 2016-09-20

Version: 1.7.1

Improvements

- Fixes an issue where an Aurora Replica crashes if the InnoDB full-text search cache is full.
- Fixes an issue where the database engine crashes if a worker thread in the thread pool waits for itself.
- Fixes an issue where an Aurora Replica crashes if a metadata lock on a table causes a deadlock.
- Fixes an issue where the database engine crashes due to a race condition between two worker threads in the thread pool.
- Fixes an issue where an unnecessary failover occurs under heavy load if the monitoring agent doesn't detect the advancement of write operations to the distributed storage subsystem.

Aurora MySQL database engine updates: 2016-08-30

Version: 1.7.0

New features

- **NUMA aware scheduler** – The task scheduler for the Aurora MySQL engine is now Non-Uniform Memory Access (NUMA) aware. This minimizes cross-CPU socket contention resulting in improved performance throughput for the db.r3.8xlarge DB instance class.
- **Parallel read-ahead operates asynchronously in the background** – Parallel read-ahead has been revised to improve performance by using a dedicated thread to reduce thread contention.
- **Improved index build (lab mode)** – The implementation for building secondary indexes now operates by building the index in a bottom-up fashion, which eliminates unnecessary page splits. This can reduce the time needed to create an index or rebuild a table. This feature is disabled by default and can be activated by enabling Aurora lab mode. For information, see [Amazon Aurora MySQL lab mode \(p. 1032\)](#).

Improvements

- Fixed an issue where establishing a connection was taking a long time if there was a surge in the number of connections requested for an instance.
- Fixed an issue where a crash occurred if ALTER TABLE was run on a partitioned table that did not use InnoDB.
- Fixed an issue where heavy write workload can cause a failover.
- Fixed an erroneous assertion that caused a failure if RENAME TABLE was run on a partitioned table.
- Improved stability when rolling back a transaction during insert-heavy workload.
- Fixed an issue where full-text search indexes were not viable on an Aurora Replica.

Integration of MySQL bug fixes

- Improve scalability by partitioning LOCK_grant lock. (Port WL #8355)
- Opening cursor on SELECT in stored procedure causes segfault. (Port Bug #16499751)
- MySQL gives the wrong result with some special usage. (Bug #11751794)

- Crash in GET_SEL_ARG_FOR_KEYPART – caused by patch for bug #11751794. (Bug #16208709)
- Wrong results for a simple query with GROUP BY. (Bug #17909656)
- Extra rows on semijoin query with range predicates. (Bug #16221623)
- Adding an ORDER BY clause following an IN subquery could cause duplicate rows to be returned. (Bug #16308085)
- Crash with explain for a query with loose scan for GROUP BY, MyISAM. (Bug #16222245)
- Loose index scan with quoted int predicate returns random data. (Bug #16394084)
- If the optimizer was using a loose index scan, the server could exit while attempting to create a temporary table. (Bug #16436567)
- COUNT(DISTINCT) should not count NULL values, but they were counted when the optimizer used loose index scan. (Bug #17222452)
- If a query had both MIN() MAX() and aggregate_function(DISTINCT) (for example, SUM(DISTINCT)) and was executed using loose index scan, the result values of MIN() MAX() were set improperly. (Bug #17217128)

Aurora MySQL database engine updates: 2016-06-01

Version: 1.6.5

New features

- **Efficient storage of Binary Logs** – Efficient storage of binary logs is now enabled by default for all Aurora MySQL DB clusters, and is not configurable. Efficient storage of binary logs was introduced in the April 2016 update. For more information, see [Aurora MySQL database engine updates: 2016-04-06 \(p. 1241\)](#).

Improvements

- Improved stability for Aurora Replicas when the primary instance is encountering a heavy workload.
- Improved stability for Aurora Replicas when running queries on partitioned tables and tables with special characters in the table name.
- Fixed connection issues when using secure connections.

Integration of MySQL bug fixes

- SLAVE CAN'T CONTINUE REPLICATION AFTER MASTER'S CRASH RECOVERY (Port Bug #17632285)

Aurora MySQL database engine updates: 2016-04-06

Version: 1.6

This update includes the following improvements:

New features

- **Parallel read-ahead** – Parallel read-ahead is now enabled by default for all Aurora MySQL DB clusters, and is not configurable. Parallel read-ahead was introduced in the December 2015 update. For more information, see [Aurora MySQL database engine updates: 2015-12-03 \(p. 1243\)](#).

In addition to enabling parallel read-ahead by default, this release includes the following improvements to parallel read-ahead:

- Improved logic so that parallel read-ahead is less aggressive, which is beneficial when your DB cluster encounters many parallel workloads.
- Improved stability on smaller tables.
- **Efficient storage of Binary Logs (lab mode)** – MySQL binary log files are now stored more efficiently in Aurora MySQL. The new storage implementation enables binary log files to be deleted much earlier and improves system performance for an instance in an Aurora MySQL DB cluster that is a binary log replication master.

To enable efficient storage of binary logs, set the `aurora_lab_mode` parameter to 1 in the parameter group for your primary instance or Aurora Replica. The `aurora_lab_mode` parameter is an instance-level parameter that is in the `default.aurora5.6` parameter group by default. For information on modifying a DB parameter group, see [Modifying parameters in a DB parameter group \(p. 347\)](#). For information on parameter groups and Aurora MySQL, see [Aurora MySQL configuration parameters \(p. 1042\)](#).

Only turn on efficient storage of binary logs for instances in an Aurora MySQL DB cluster that are MySQL binary log replication master instances.

- **AURORA_VERSION system variable** – You can now get the Aurora version of your Aurora MySQL DB cluster by querying for the `AURORA_VERSION` system variable.

To get the Aurora version, use one of the following queries:

```
select AURORA_VERSION();
select @@aurora_version;
show variables like '%version';
```

You can also see the Aurora version in the AWS Management Console when you modify a DB cluster, or by calling the [describe-db-engine-versions](#) AWS CLI command or the [DescribeDBEngineVersions](#) API operation.

- **Lock manager memory usage metric** – Information about lock manager memory usage is now available as a metric.

To get the lock manager memory usage metric, use one of the following queries:

```
show global status where variable_name in ('aurora_lockmgr_memory_used');
select * from INFORMATION_SCHEMA.GLOBAL_STATUS where variable_name in
('aurora_lockmgr_memory_used');
```

Improvements

- Improved stability during binlog and XA transaction recovery.
- Fixed a memory issue resulting from a large number of connections.
- Improved accuracy in the following metrics: Read Throughput, Read IOPS, Read Latency, Write Throughput, Write IOPS, Write Latency, and Disk Queue Depth.
- Fixed a stability issue causing slow startup for large instances after a crash.
- Improved concurrency in the data dictionary regarding synchronization mechanisms and cache eviction.
- Stability and performance improvements for Aurora Replicas:
 - Fixed a stability issue for Aurora Replicas during heavy or burst write workloads for the primary instance.
 - Improved replica lag for db.r3.4xlarge and db.r3.8xlarge instances.

- Improved performance by reducing contention between application of log records and concurrent reads on an Aurora Replica.
- Fixed an issue for refreshing statistics on Aurora Replicas for newly created or updated statistics.
- Improved stability for Aurora Replicas when there are many transactions on the primary instance and concurrent reads on the Aurora Replicas across the same data.
- Improved stability for Aurora Replicas when running `UPDATE` and `DELETE` statements with `JOIN` statements.
- Improved stability for Aurora Replicas when running `INSERT ... SELECT` statements.

Integration of MySQL bug fixes

- BACKPORT Bug #18694052 FIX FOR ASSERTION `!M_ORDERED_REC_BUFFER' FAILED TO 5.6 (Port Bug #18305270)
- SEGV IN MEMCPY(), HA_PARTITION::POSITION (Port Bug # 18383840)
- WRONG RESULTS WITH PARTITIONING,INDEX_MERGE AND NO PK (Port Bug # 18167648)
- FLUSH TABLES FOR EXPORT: ASSERTION IN HA_PARTITION::EXTRA (Port Bug # 16943907)
- SERVER CRASH IN VIRTUAL HA_ROWS HANDLER::MULTI_RANGE_READ_INFO_CONST (Port Bug # 16164031)
- RANGE OPTIMIZER CRASHES IN SEL_ARG::RB_INSERT() (Port Bug # 16241773)

Aurora MySQL database engine updates: 2016-01-11

Version: 1.5

This update includes the following improvements:

Improvements

- Fixed a 10 second pause of write operations for idle instances during Aurora storage deployments.
- Logical read-ahead now works when `innodb_file_per_table` is set to No. For more information on logical read-ahead, see [Aurora MySQL database engine updates: 2015-12-03 \(p. 1243\)](#).
- Fixed issues with Aurora Replicas reconnecting with the primary instance. This improvement also fixes an issue when you specify a large value for the `quantity` parameter when testing Aurora Replica failures using fault-injection queries. For more information, see [Testing an Aurora replica failure \(p. 830\)](#).
- Improved monitoring of Aurora Replicas falling behind and restarting.
- Fixed an issue that caused an Aurora Replica to lag, become deregistered, and then restart.
- Fixed an issue when you run the `show innodb status` command during a deadlock.
- Fixed an issue with failovers for larger instances during high write throughput.

Integration of MySQL bug fixes

- Addressed incomplete fix in MySQL full text search affecting tables where the database name begins with a digit. (Port Bug #17607956)

Aurora MySQL database engine updates: 2015-12-03

Version: 1.4

This update includes the following improvements:

New features

- **Fast Insert** – Accelerates parallel inserts sorted by primary key. For more information, see [Amazon Aurora MySQL performance enhancements \(p. 746\)](#).
- **Large dataset read performance** – Aurora MySQL automatically detects an IO heavy workload and launches more threads in order to boost the performance of the DB cluster. The Aurora scheduler looks into IO activity and decides to dynamically adjust the optimal number of threads in the system, quickly adjusting between IO heavy and CPU heavy workloads with low overhead.
- **Parallel read-ahead** – Improves the performance of B-Tree scans that are too large for the memory available on your primary instance or Aurora Replica (including range queries). Parallel read-ahead automatically detects page read patterns and pre-fetches pages into the buffer cache in advance of when they are needed. Parallel read-ahead works multiple tables at the same time within the same transaction.

Improvements:

- Fixed brief Aurora database availability issues during Aurora storage deployments.
- Correctly enforce the `max_connection` limit.
- Improve binlog purging where Aurora is the binlog master and the database is restarting after a heavy data load.
- Fixed memory management issues with the table cache.
- Add support for huge pages in shared memory buffer cache for faster recovery.
- Fixed an issue with thread-local storage not being initialized.
- Allow 16K connections by default.
- Dynamic thread pool for IO heavy workloads.
- Fixed an issue with properly invalidating views involving UNION in the query cache.
- Fixed a stability issue with the dictionary stats thread.
- Fixed a memory leak in the dictionary subsystem related to cache eviction.
- Fixed high read latency issue on Aurora Replicas when there is very low write load on the master.
- Fixed stability issues on Aurora Replicas when performing operations on DDL partitioned tables such as `ALTER TABLE ... REORGANIZE PARTITION` on the master.
- Fixed stability issues on Aurora Replicas during volume growth.
- Fixed performance issue for scans on non-clustered indexes in Aurora Replicas.
- Fix stability issue that makes Aurora Replicas lag and eventually get deregistered and re-started.

Integration of MySQL bug fixes

- SEGV in FTSPARSE(). (Bug #16446108)
- InnoDB data dictionary is not updated while renaming the column. (Bug #19465984)
- FTS crash after renaming table to different database. (Bug #16834860)
- Failed preparing of trigger on truncated tables cause error 1054. (Bug #18596756)
- Metadata changes might cause problems with trigger execution. (Bug #18684393)
- Materialization is not chosen for long UTF8 VARCHAR field. (Bug #17566396)
- Poor execution plan when ORDER BY with limit X. (Bug #16697792)
- Backport bug #11765744 TO 5.1, 5.5 AND 5.6. (Bug #17083851)

- Mutex issue in SQL/SQL_SHOW.CC resulting in SIG6. Source likely FILL_VARIABLES. (Bug #20788853)
- Backport bug #18008907 to 5.5+ versions. (Bug #18903155)
- Adapt fix for a stack overflow error in MySQL 5.7. (Bug #19678930)

Aurora MySQL database engine updates: 2015-10-16

Versions: 1.2, 1.3

This update includes the following improvements:

Fixes

- Resolved out-of-memory issue in the new lock manager with long-running transactions
- Resolved security vulnerability when replicating with non-RDS for MySQL databases
- Updated to ensure that quorum writes retry correctly with storage failures
- Updated to report replica lag more accurately
- Improved performance by reducing contention when many concurrent transactions are trying to modify the same row
- Resolved query cache invalidation for views that are created by joining two tables
- Disabled query cache for transactions with UNCOMMITTED_READ isolation

Improvements

- Better performance for slow catalog queries on warm caches
- Improved concurrency in dictionary statistics
- Better stability for the new query cache resource manager, extent management, files stored in Amazon Aurora smart storage, and batch writes of log records

Integration of MySQL bug fixes

- Killing a query inside innodb causes it to eventually crash with an assertion. (Bug #1608883)
- For failure to create a new thread for the event scheduler, event execution, or new connection, no message was written to the error log. (Bug #16865959)
- If one connection changed its default database and simultaneously another connection executed SHOW PROCESSLIST, the second connection could access invalid memory when attempting to display the first connection's default database memory. (Bug #11765252)
- PURGE BINARY LOGS by design does not remove binary log files that are in use or active, but did not provide any notice when this occurred. (Bug #13727933)
- For some statements, memory leaks could result when the optimizer removed unneeded subquery clauses. (Bug #15875919)
- During shutdown, the server could attempt to lock an uninitialized mutex. (Bug #16016493)
- A prepared statement that used GROUP_CONCAT() and an ORDER BY clause that named multiple columns could cause the server to exit. (Bug #16075310)
- Performance Schema instrumentation was missing for replica worker threads. (Bug #16083949)
- STOP SLAVE could cause a deadlock when issued concurrently with a statement such as SHOW STATUS that retrieved the values for one or more of the status variables Slave_retried_transactions, Slave_heartbeat_period, Slave_received_heartbeats, Slave_last_heartbeat, or Slave_running. (Bug #16088188)

- A full-text query using Boolean mode could return zero results in some cases where the search term was a quoted phrase. (Bug #16206253)
- The optimizer's attempt to remove redundant subquery clauses raised an assertion when executing a prepared statement with a subquery in the ON clause of a join in a subquery. (Bug #16318585)
- GROUP_CONCAT unstable, crash in ITEM_SUM::CLEAN_UP_AFTER_REMOVAL. (Bug #16347450)
- Attempting to replace the default InnoDB full-text search (FTS) stopword list by creating an InnoDB table with the same structure as INFORMATION_SCHEMA.INNODB_FT_DEFAULT_STOPWORD would result in an error. (Bug #16373868)
- After the client thread on a worker performed a FLUSH TABLES WITH READ LOCK and was followed by some updates on the master, the worker hung when executing SHOW SLAVE STATUS. (Bug #16387720)
- When parsing a delimited search string such as "abc-def" in a full-text search, InnoDB now uses the same word delimiters as MyISAM. (Bug #16419661)
- Crash in FTS_AST_TERM_SET_WILDCARD. (Bug #16429306)
- SEGFAULT in FTS_AST_VISIT() for FTS RQG test. (Bug # 16435855)
- For debug builds, when the optimizer removed an Item_ref pointing to a subquery, it caused a server exit. (Bug #16509874)
- Full-text search on InnoDB tables failed on searches for literal phrases combined with + or - operators. (Bug #16516193)
- START SLAVE failed when the server was started with the options --master-info-repository=TABLE relay-log-info-repository=TABLE and with autocommit set to 0, together with --skip-slave-start. (Bug #16533802)
- Very large InnoDB full-text search (FTS) results could consume an excessive amount of memory. (Bug #16625973)
- In debug builds, an assertion could occur in OPT_CHECK_ORDER_BY when using binary directly in a search string, as binary might include NULL bytes and other non-meaningful characters. (Bug #16766016)
- For some statements, memory leaks could result when the optimizer removed unneeded subquery clauses. (Bug #16807641)
- It was possible to cause a deadlock after issuing FLUSH TABLES WITH READ LOCK by issuing STOP SLAVE in a new connection to the worker, then issuing SHOW SLAVE STATUS using the original connection. (Bug #16856735)
- GROUP_CONCAT() with an invalid separator could cause a server exit. (Bug #16870783)
- The server did excessive locking on the LOCK_active_mi and active_mi->rli->data_lock mutexes for any SHOW STATUS LIKE 'pattern' statement, even when the pattern did not match status variables that use those mutexes (Slave_heartbeat_period, Slave_last_heartbeat, Slave_received_heartbeats, Slave_retried_transactions, Slave_running). (Bug #16904035)
- A full-text search using the IN BOOLEAN MODE modifier would result in an assertion failure. (Bug #16927092)
- Full-text search on InnoDB tables failed on searches that used the + boolean operator. (Bug #17280122)
- 4-way deadlock: zombies, purging binlogs, show processlist, show binlogs. (Bug #17283409)
- When an SQL thread which was waiting for a commit lock was killed and restarted it caused a transaction to be skipped on worker. (Bug #17450876)
- An InnoDB full-text search failure would occur due to an "unended" token. The string and string length should be passed for string comparison. (Bug #17659310)
- Large numbers of partitioned InnoDB tables could consume much more memory when used in MySQL 5.6 or 5.7 than the memory used by the same tables used in previous releases of the MySQL Server. (Bug #17780517)

- For full-text queries, a failure to check that num_token is less than max_proximity_item could result in an assertion. (Bug #18233051)
- Certain queries for the INFORMATION_SCHEMA TABLES and COLUMNS tables could lead to excessive memory use when there were large numbers of empty InnoDB tables. (Bug #18592390)
- When committing a transaction, a flag is now used to check whether a thread has been created, rather than checking the thread itself, which uses more resources, particularly when running the server with master_info_repository=TABLE. (Bug #18684222)
- If a client thread on a worker executed FLUSH TABLES WITH READ LOCK while the master executed a DML, executing SHOW SLAVE STATUS in the same client became blocked, causing a deadlock. (Bug #19843808)
- Ordering by a GROUP_CONCAT() result could cause a server exit. (Bug #19880368)

Aurora MySQL database engine updates: 2015-08-24

Version: 1.1

This update includes the following improvements:

- Replication stability improvements when replicating with a MySQL database (binlog replication). For information on Aurora MySQL replication with MySQL, see [Replication with Amazon Aurora \(p. 70\)](#).
- A 1 gigabyte (GB) limit on the size of the relay logs accumulated for an Aurora MySQL DB cluster that is a replication worker. This improves the file management for the Aurora DB clusters.
- Stability improvements in the areas of read ahead, recursive foreign-key relationships, and Aurora replication.
- Integration of MySQL bug fixes.
 - InnoDB databases with names beginning with a digit cause a full-text search (FTS) parser error. (Bug #17607956)
 - InnoDB full-text searches fail in databases whose names began with a digit. (Bug #17161372)
 - For InnoDB databases on Windows, the full-text search (FTS) object ID is not in the expected hexadecimal format. (Bug #16559254)
 - A code regression introduced in MySQL 5.6 negatively impacted DROP TABLE and ALTER TABLE performance. This could cause a performance drop between MySQL Server 5.5.x and 5.6.x. (Bug #16864741)
- Simplified logging to reduce the size of log files and the amount of storage that they require.

Database engine updates for Aurora MySQL Serverless clusters

The following are Aurora MySQL database engine updates for Aurora MySQL Serverless DB clusters. These release notes provide information about bug fixes, updates to the database engine, and other critical information. Aurora Serverless v1 doesn't have its own version number. It shares the version number (and name) of the Aurora database engine that supports Aurora Serverless v1. For more information, see [Aurora Serverless v1 and Aurora database engine versions \(p. 177\)](#).

- [Aurora MySQL Serverless 5.7 engine updates 2021-07-16 \(version 2.08.3\) \(p. 1248\)](#)
- [Aurora MySQL Serverless 5.7 engine updates 2020-06-18 \(version 2.07.1\) \(p. 1248\)](#)
- [Aurora MySQL Serverless 5.6 engine updates 2021-07-16 \(version 1.22.3\) \(p. 1249\)](#)
- [Aurora MySQL Serverless 5.6 engine updates 2020-08-14 \(version 1.21.0\) \(p. 1249\)](#)

Aurora MySQL Serverless 5.7 engine updates 2021-07-16 (version 2.08.3)

Aurora Serverless 5.7 is generally available. It has the same features and bug fixes as Aurora MySQL 2.08.3.

You can restore a snapshot from a currently supported Aurora MySQL release into Aurora MySQL Serverless 5.7. By using a snapshot, you can also upgrade an Aurora MySQL-5.6 compatible Aurora Serverless v1 DB cluster to an Aurora MySQL-5.7 compatible Aurora Serverless v1 DB cluster. To do so:

- Create a snapshot from the Aurora MySQL 5.6 Serverless DB cluster. To learn how, see [Creating a DB cluster snapshot \(p. 495\)](#).
- Restore the snapshot to a new Aurora MySQL 5.7 Serverless cluster. For more information, see [Restoring an Aurora Serverless v1 DB cluster \(p. 166\)](#).

Aurora Serverless v1 doesn't have its own version number. It uses the number of the Aurora MySQL version that supports it to distinguish between Aurora MySQL 5.7 and Aurora MySQL 5.6 updates. For more information, see [Aurora Serverless v1 and Aurora database engine versions \(p. 177\)](#).

For general information about Aurora Serverless, see [Using Amazon Aurora Serverless v1 \(p. 147\)](#).

If you have any questions or concerns, AWS Support is available on the community forums and through [AWS Premium Support](#). For more information, see [Maintaining an Aurora DB cluster](#).

Bug fixes:

This Aurora Serverless release includes all bug fixes up to Aurora MySQL version 2.08.3. For details, see [Aurora MySQL database engine updates 2020-11-12 \(version 2.08.3\) \(p. 1133\)](#) and the release notes for previous Aurora MySQL versions.

Features:

This Aurora Serverless release includes all new features up to Aurora MySQL version 2.08.3. For details, see [Aurora MySQL database engine updates 2020-11-12 \(version 2.08.3\) \(p. 1133\)](#) and the release notes for previous Aurora MySQL versions.

Aurora MySQL Serverless 5.7 engine updates 2020-06-18 (version 2.07.1)

Aurora Serverless 5.7 is generally available. It has the same features and bug fixes as Aurora MySQL 2.07.1.

You can restore a snapshot from a currently supported Aurora MySQL release into Aurora MySQL Serverless 5.7. By using a snapshot, you can also upgrade an Aurora MySQL-5.6 compatible Aurora Serverless v1 DB cluster to an Aurora MySQL-5.7 compatible Aurora Serverless v1 DB cluster. To do so:

- Create a snapshot from the Aurora MySQL 5.6 Serverless DB cluster. To learn how, see [Creating a DB cluster snapshot \(p. 495\)](#).
- Restore the snapshot to a new Aurora MySQL 5.7 Serverless cluster. For more information, see [Restoring an Aurora Serverless v1 DB cluster \(p. 166\)](#).

Aurora Serverless v1 doesn't have its own version number. It uses the number of the Aurora MySQL version that supports it to distinguish between Aurora MySQL 5.7 and Aurora MySQL 5.6 updates. For more information, see [Aurora Serverless v1 and Aurora database engine versions \(p. 177\)](#).

For general information about Aurora Serverless, see [Using Amazon Aurora Serverless v1 \(p. 147\)](#).

If you have any questions or concerns, AWS Support is available on the community forums and through [AWS Premium Support](#). For more information, see [Maintaining an Aurora DB cluster](#).

Bug fixes:

This Aurora Serverless release includes all bug fixes up to Aurora MySQL version 2.07.1. For details, see [Aurora MySQL database engine updates 2019-12-23 \(version 2.07.1\) \(p. 1151\)](#) and the release notes for previous Aurora MySQL versions.

Features:

This Aurora Serverless release includes all new features up to Aurora MySQL version 2.07.1. For details, see [Aurora MySQL database engine updates 2019-12-23 \(version 2.07.1\) \(p. 1151\)](#) and the release notes for previous Aurora MySQL versions. The following features are of particular interest for users of Aurora Serverless or Aurora MySQL with MySQL 5.7 compatibility:

- Aurora MySQL Serverless now supports the hot row contention feature. For more information, see [Aurora MySQL database engine updates: 2016-12-14 \(p. 1236\)](#).
- Aurora MySQL Serverless now supports the hash join feature. To use this feature, you must specify the configuration setting `optimizer_switch='hash_join=on'`. For more information, see [Optimizing large Aurora MySQL join queries with hash joins \(p. 1038\)](#).
- Aurora Serverless 5.7 can use the Data API. For more information, see [Using the Data API for Aurora Serverless \(p. 178\)](#).
- Aurora Serverless 5.7 can use the query editor. For more information, see [Using the query editor for Aurora Serverless \(p. 204\)](#).
- Aurora Serverless 5.7 supports the same JSON features as other Aurora MySQL versions that are compatible with MySQL 5.7.

Aurora MySQL Serverless 5.6 engine updates 2021-07-16 (version 1.22.3)

Aurora Serverless 5.6 is generally available. It has the same features and bug fixes as Aurora MySQL 1.22.3.

Aurora Serverless v1 doesn't have its own version number. It uses the number of the Aurora MySQL version that supports it to distinguish between Aurora MySQL 5.6 and Aurora MySQL 5.7 updates. For more information, see [Aurora Serverless v1 and Aurora database engine versions \(p. 177\)](#). For general information about Aurora Serverless, see [Using Amazon Aurora Serverless v1 \(p. 147\)](#).

If you have any questions or concerns, AWS Support is available on the community forums and through [AWS Premium Support](#). For more information, see [Maintaining an Aurora DB cluster](#).

Bug fixes:

This Aurora Serverless release includes all bug fixes up to Aurora MySQL version 1.22.3. For details, see [Aurora MySQL database engine updates 2020-11-09 \(version 1.22.3\) \(p. 1205\)](#) and the release notes for previous Aurora MySQL versions.

Aurora MySQL Serverless 5.6 engine updates 2020-08-14 (version 1.21.0)

Aurora Serverless 5.6 is generally available. It has the same features and bug fixes as Aurora MySQL 1.21.0.

Aurora Serverless v1 doesn't have its own version number. It uses the number of the Aurora MySQL version that supports it to distinguish between Aurora MySQL 5.6 and Aurora MySQL 5.7 updates. For

more information, see [Aurora Serverless v1 and Aurora database engine versions \(p. 177\)](#). For general information about Aurora Serverless, see [Using Amazon Aurora Serverless v1 \(p. 147\)](#).

If you have any questions or concerns, AWS Support is available on the community forums and through [AWS Premium Support](#). For more information, see [Maintaining an Aurora DB cluster](#).

Bug fixes:

This Aurora Serverless release includes all bug fixes up to Aurora MySQL version 1.21.0. For details, see [Aurora MySQL database engine updates 2019-11-25 \(version 1.21.0\) \(p. 1211\)](#) and the release notes for previous Aurora MySQL versions.

Features:

This Aurora Serverless release improves CPU utilization across the Serverless fleet, especially benefiting clusters with one and two Aurora capacity units (ACUs). See [Aurora Serverless v1 architecture \(p. 152\)](#) for more information about ACUs.

MySQL bugs fixed by Aurora MySQL database engine updates

The following sections identify MySQL bugs that have been fixed by Aurora MySQL database engine updates.

Topics

- [MySQL bugs fixed by Aurora MySQL 2.x database engine updates \(p. 1250\)](#)
- [MySQL bugs fixed by Aurora MySQL 1.x database engine updates \(p. 1261\)](#)

MySQL bugs fixed by Aurora MySQL 2.x database engine updates

MySQL 5.7-compatible version Aurora contains all MySQL bug fixes through MySQL 5.7.12. The following table identifies additional MySQL bugs that have been fixed by Aurora MySQL database engine updates, and which update they were fixed in.

Database engine update	Version	MySQL bugs fixed
Aurora MySQL database engine updates 2022-01-26 (version 2.10.2) (p. 1110)	2.10.2	<ul style="list-style-type: none">• Fixed an issue in InnoDB where an error in code related to table statistics raised an assertion in the dict0stats.cc (http://dict0stats.cc/) source file. (Bug #24585978)• A secondary index over a virtual column became corrupted when the index was built online. For UPDATE (https://dev.mysql.com/doc/refman/5.7/en/update.html) statements, we fix this as follows: If the virtual column value of the index record is set to NULL, then we generate this value from the cluster index record. (Bug #30556595)• ASSERTION "NOT OTHER_LOCK" IN LOCK_REC_ADD_TO_QUEUE (Bug #29195848)• HANDLE_FATAL_SIGNAL (SIG=11) IN __STRCHR_SSE2 (Bug #28653104)• Fixed an issue which a query interruption during a lock wait can cause an error in InnoDB. (Bug #28068293)

Database engine update	Version	MySQL bugs fixed
		<ul style="list-style-type: none">Interleaved transactions could sometimes deadlock the replica applier when the transaction isolation level was set to REPEATABLE READ. (Bug #25040331)Fixed an issue which can cause binlog replicas to stall due to lock wait timeout.(Bug #27189701)
Aurora MySQL database engine updates 2021-10-21 (version 2.10.1) (p. 1113)	2.10.1	CURRENT_TIMESTAMP PRODUCES ZEROS IN TRIGGER. (Bug #25209512)

Database engine update	Version	MySQL bugs fixed
Aurora MySQL database engine updates 2021-05-25 (version 2.10.0) (p. 1115)	2.10.0	<ul style="list-style-type: none"> Interleaved transactions could sometimes deadlock the replica applier when the transaction isolation level was set to REPEATABLE READ. (Bug #25040331) When a stored procedure contained a statement referring to a view which in turn referred to another view, the procedure could not be invoked successfully more than once. (Bug #87858, Bug #26864199) For queries with many OR conditions, the optimizer now is more memory-efficient and less likely to exceed the memory limit imposed by the range_optimizer_max_mem_size system variable. In addition, the default value for that variable has been raised from 1,536,000 to 8,388,608. (Bug #79450, Bug #22283790) <i>Replication:</i> In the <code>next_event()</code> function, which is called by a replica's SQL thread to read the next event from the relay log, the SQL thread did not release the <code>relaylog.log_lock</code> it acquired when it ran into an error (for example, due to a closed relay log), causing all other threads waiting to acquire a lock on the relay log to hang. With this fix, the lock is released before the SQL thread leaves the function under the situation. (Bug #21697821) Fixing a memory corruption for <code>ALTER TABLE</code> with virtual column. (Bug #24961167; Bug #24960450) <i>Replication:</i> Multithreaded replicas could not be configured with small queue sizes using <code>slave_pending_jobs_size_max</code> if they ever needed to process transactions larger than that size. Any packet larger than <code>slave_pending_jobs_size_max</code> was rejected with the error <code>ER_MTS_EVENT_BIGGER_PENDING_JOBS_SIZE_MAX</code>, even if the packet was smaller than the limit set by <code>slave_max_allowed_packet</code>. With this fix, <code>slave_pending_jobs_size_max</code> becomes a soft limit rather than a hard limit. If the size of a packet exceeds <code>slave_pending_jobs_size_max</code> but is less than <code>slave_max_allowed_packet</code>, the transaction is held until all the replica workers have empty queues, and then processed. All subsequent transactions are held until the large transaction has been completed. The queue size for replica workers can therefore be limited while still allowing occasional larger transactions. (Bug #21280753, Bug #77406) <i>Replication:</i> When using a multithreaded replica, applier errors displayed worker ID data that was inconsistent with data externalized in Performance Schema replication tables. (Bug #25231367) <i>Replication:</i> On a GTID-based replication replica running with <code>-gtid-mode=ON</code>, <code>-log-bin=OFF</code>, and using <code>-slave-skip-errors</code>, when an error was encountered that should be ignored <code>Exec_Master_Log_Pos</code> was not being correctly updated, causing <code>Exec_Master_Log_Pos</code> to loose synchrony with <code>Read_master_log_pos</code>. If a <code>GTID_NEXT</code> was not specified, the replica would never update its GTID state when rolling back from a single statement transaction. The <code>Exec_Master_Log_Pos</code> would not be updated because even though the transaction was finished, its GTID state would show otherwise. The fix removes

Database engine update	Version	MySQL bugs fixed
		<p>the restraint of updating the GTID state when a transaction is rolled back only if <code>GTID_NEXT</code> is specified. (Bug #22268777)</p> <ul style="list-style-type: none"> <i>Replication:</i> A partially failed statement was not correctly consuming an auto-generated or specified GTID when binary logging was disabled. The fix ensures that a partially failed <code>DROP TABLE</code>, a partially failed <code>DROP USER</code>, or a partially failed <code>DROP VIEW</code> consume respectively the relevant GTID and save it into <code>@@GLOBAL.GTID_EXECUTED</code> and <code>mysql.gtid_executed</code> table when binary logging is disabled. (Bug #21686749) <i>Replication:</i> Replicas running MySQL 5.7 could not connect to a MySQL 5.5 source due to an error retrieving the <code>server_uuid</code>, which is not part of MySQL 5.5. This was caused by changes in the method of retrieving the <code>server_uuid</code>. (Bug #22748612) Binlog replication: GTID transaction skipping mechanism was not working properly for XA transaction before this fix. Server has a mechanism to skip (silently) a GTID transaction if it is already executed that particular transaction in the past. (BUG#25041920) <code>">XA ROLLBACK</code> statements that failed because an incorrect transaction ID was given, could be recorded in the binary log with the correct transaction ID, and could therefore be actioned by replication replicas. A check is now made for the error situation before binary logging takes place, and failed XA ROLLBACK statements are not logged. (Bug #26618925) <i>Replication:</i> If a replica was set up using a <code>CHANGE MASTER TO</code> statement that did not specify the source log file name and source log position, then shut down before <code>START SLAVE</code> was issued, then restarted with the option <code>-relay-log-recovery</code> set, replication did not start. This happened because the receiver thread had not been started before relay log recovery was attempted, so no log rotation event was available in the relay log to provide the source log file name and source log position. In this situation, the replica now skips relay log recovery and logs a warning, then proceeds to start replication. (Bug #28996606, Bug #93397) <i>Replication:</i> In row-based replication, a message that incorrectly displayed field lengths was returned when replicating from a table with a <code>utf8mb3</code> column to a table of the same definition where the column was defined with a <code>utf8mb4</code> character set. (Bug #25135304, Bug #83918) <i>Replication:</i> When a <code>RESET SLAVE</code> statement was issued on a replication replica with GTIDs in use, the existing relay log files were purged, but the replacement new relay log file was generated before the set of received GTIDs for the channel had been cleared. The former GTID set was therefore written to the new relay log file as the <code>PREVIOUS_GTIDS</code> event, causing a fatal error in replication stating that the replica had more GTIDs than the source, even though the <code>gtid_executed</code> set for both servers was empty. Now, when <code>RESET SLAVE</code> is issued, the set of received GTIDs is cleared before the new relay log file is generated, so that this situation does not occur. (Bug #27411175) <i>Replication:</i> With GTIDs in use for replication, transactions including statements that caused a parsing error

Database engine update	Version	MySQL bugs fixed
		<p>(ER_PARSE_ERROR) could not be skipped manually by the recommended method of injecting an empty or replacement transaction with the same GTID. This action should result in the replica identifying the GTID as already used, and therefore skipping the unwanted transaction that shared its GTID. However, in the case of a parsing error, because the statement was parsed before the GTID was checked to see if it needed to be skipped, the replication applier thread stopped due to the parsing error, even though the intention was for the transaction to be skipped anyway. With this fix, the replication applier thread now ignores parsing errors if the transaction concerned needs to be skipped because the GTID was already used. Note that this behavior change does not apply in the case of workloads consisting of binary log output produced by <code>mysqlbinlog</code>. In that situation, there would be a risk that a transaction with a parsing error that immediately follows a skipped transaction would also be silently skipped, when it ought to raise an error. (Bug #27638268)</p> <ul style="list-style-type: none"> • <i>Replication:</i> Enable the SQL thread to GTID skip a partial transaction. (Bug #25800025) • <i>Replication:</i> When a negative or fractional timeout parameter was supplied to <code>WAIT_UNTIL_SQL_THREAD_AFTER_GTIDS()</code>, the server behaved in unexpected ways. With this fix: <ul style="list-style-type: none"> • A fractional timeout value is read as-is, with no round-off. • A negative timeout value is rejected with an error if the server is on a strict SQL mode; if the server is not on a strict SQL mode, the value makes the function return <code>NULL</code> immediately without any waiting and then issue a warning. (Bug #24976304, Bug #83537) • <i>Replication:</i> If the <code>WAIT_FOR_EXECUTED_GTID_SET()</code> function was used with a timeout value including a fractional part (for example, 1.5), an error in the casting logic meant that the timeout was rounded down to the nearest whole second, and to zero for values less than 1 second (for example, 0.1). The casting logic has now been corrected so that the timeout value is applied as originally specified with no rounding. Thanks to Dirkjan Bussink for the contribution. (Bug #29324564, Bug #94247) • With GTIDs enabled, <code>XA COMMIT</code> on a disconnected XA transaction within a multiple-statement transaction raised an assertion. (Bug #22173903) • <i>Replication:</i> An assertion was raised in debug builds if an <code>XA ROLLBACK</code> statement was issued for an unknown transaction identifier when the <code>gtid_next</code> value had been set manually. The server now does not attempt to update the GTID state if an XA ROLLBACK statement fails with an error. (Bug #27928837, Bug #90640) • Fix wrong sorting order issue when multiple <code>CASE</code> functions are used in <code>ORDER BY</code> clause (Bug#22810883).

Database engine update	Version	MySQL bugs fixed
Aurora MySQL database engine updates 2021-11-12 (version 2.09.3) (p. 1119)	2.09.3	<ul style="list-style-type: none"> • ASSERTION !M_PREBUILT->TRX->CHECK_FOREIGNS. (Bug #23533396) • Replication: A locking issue in the WAIT_FOR_EXECUTED_GTID_SET() function could cause the server to hang in certain circumstances. The issue has now been corrected. (Bug #29550513)
Aurora MySQL database engine updates 2020-12-11 (version 2.09.1) (p. 1124)	2.09.1	<ul style="list-style-type: none"> • Replication: Interleaved transactions could sometimes deadlock the slave applier when the transaction isolation level was set to REPEATABLE READ. (Bug #25040331) • For a table having a TIMESTAMP or DATETIME column having a default of CURRENT_TIMESTAMP, the column could be initialized to 0000-00-00 00:00:00 if the table had a BEFORE INSERT trigger. (Bug #25209512, Bug #84077) • For an INSERT statement for which the VALUES list produced values for the second or later row using a subquery containing a join, the server could exit after failing to resolve the required privileges. (Bug #23762382)
Aurora MySQL database engine updates 2020-11-12 (version 2.08.3) (p. 1133)	2.08.3	<ul style="list-style-type: none"> • Bug #23762382 - INSERT VALUES QUERY WITH JOIN IN A SELECT CAUSES INCORRECT BEHAVIOR. • Bug #25209512 - CURRENT_TIMESTAMP PRODUCES ZEROS IN TRIGGER.
Aurora MySQL database engine updates 2020-06-02 (version 2.08.0) (p. 1137)	2.08.0	<ul style="list-style-type: none"> • Bug #25289359: A full-text cache lock taken when data is synchronized was not released if the full-text cache size exceeded the full-text cache size limit. • Bug #29138644: Manually changing the system time while the MySQL server was running caused page cleaner thread delays. • Bug #25222337: A NULL virtual column field name in a virtual index caused a server exit during a field name comparison that occurs while populating virtual columns affected by a foreign key constraint. • Bug #25053286: Executing a stored procedure containing a query that accessed a view could allocate memory that was not freed until the session ended. • Bug #25586773: Executing a stored procedure containing a statement that created a table from the contents of certain SELECT statements could result in a memory leak. • Bug #28834208: During log application, after an OPTIMIZE TABLE operation, InnoDB did not populate virtual columns before checking for virtual column index updates. • Bug #26666274: Infinite loop in performance schema buffer container due to 32-bit unsigned integer overflow.
Aurora MySQL database engine updates 2021-11-24 (version 2.07.7) (p. 1140)	2.07.7	INSERTING 64K SIZE RECORDS TAKE TOO MUCH TIME. (Bug#23031146)

Database engine update	Version	MySQL bugs fixed
Aurora MySQL database engine updates 2021-09-02 (version 2.07.6) (p. 1142)	2.07.6	<ul style="list-style-type: none">• INSERTING 64K SIZE RECORDS TAKE TOO MUCH TIME. (Bug#23031146)
Aurora MySQL database engine updates 2021-03-04 (version 2.07.4) (p. 1145)	2.07.4	<ul style="list-style-type: none">• Fixed an issue in the Full-text ngram parser when dealing with tokens containing '' (space), '%', or '. Customers should rebuild their FTS indexes if using ngram parser. (Bug #25873310)• Fixed an issue that could cause engine restart during query execution with nested SQL views. (Bug #27214153, Bug #26864199)

Database engine update	Version	MySQL bugs fixed
Aurora MySQL database engine updates 2020-11-10 (version 2.07.3) (p. 1147)	2.07.3	<ul style="list-style-type: none"> • <i>InnoDB</i>: Concurrent XA transactions that ran successfully to the XA prepare stage on the master conflicted when replayed on the slave, resulting in a lock wait timeout in the applier thread. The conflict was due to the GAP lock range which differed when the transactions were replayed serially on the slave. To prevent this type of conflict, GAP locks taken by XA transactions in READ COMMITTED isolation level are now released (and no longer inherited) when XA transactions reach the prepare stage. (Bug #27189701, Bug #25866046) • <i>InnoDB</i>: A gap lock was taken unnecessarily during foreign key validation while using the READ COMMITTED isolation level. (Bug #25082593) • <i>Replication</i>: When using XA transactions, if a lock wait timeout or deadlock occurred for the applier (SQL) thread on a replication slave, the automatic retry did not work. The cause was that while the SQL thread would do a rollback, it would not roll the XA transaction back. This meant that when the transaction was retried, the first event was XA START which was invalid as the XA transaction was already in progress, leading to an XAER_RMFAIL error. (Bug #24764800) • <i>Replication</i>: Interleaved transactions could sometimes deadlock the slave applier when the transaction isolation level was set to REPEATABLE READ. (Bug #25040331) • <i>Replication</i>: The value returned by a SHOW SLAVE STATUS statement for the total combined size of all existing relay log files (Relay_Log_Space) could become much larger than the actual disk space used by the relay log files. The I/O thread did not lock the variable while it updated the value, so the SQL thread could automatically delete a relay log file and write a reduced value before the I/O thread finished updating the value. The I/O thread then wrote its original size calculation, ignoring the SQL thread's update and so adding back the space for the deleted file. The Relay_Log_Space value is now locked during updates to prevent concurrent updates and ensure an accurate calculation. (Bug #26997096, Bug #87832) • For an INSERT statement for which the VALUES list produced values for the second or later row using a subquery containing a join, the server could exit after failing to resolve the required privileges. (Bug #23762382) • For a table having a TIMESTAMP or DATETIME column having a default of CURRENT_TIMESTAMP, the column could be initialized to 0000-00-00 00:00:00 if the table had a BEFORE INSERT trigger. (Bug #25209512, Bug #84077) • A server exit could result from simultaneous attempts by multiple threads to register and deregister metadata Performance Schema objects. (Bug #26502135) • Executing a stored procedure containing a statement that created a table from the contents of certain SELECT statements could result in a memory leak. (Bug #25586773)

Database engine update	Version	MySQL bugs fixed
		<ul style="list-style-type: none"> Executing a stored procedure containing a query that accessed a view could allocate memory that was not freed until the session ended. (Bug #25053286) Certain cases of subquery materialization could cause a server exit. These queries now produce an error suggesting that materialization be disabled. (Bug #26402045) Queries with many left joins were slow if join buffering was used (for example, using the block nested loop algorithm). (Bug #18898433, Bug #72854) The optimizer skipped the second column in a composite index when executing an inner join with a <code>LIKE</code> clause against the second column. (Bug #28086754)
Aurora MySQL database engine updates 2020-04-17 (version 2.07.2) (p. 1149)	2.07.2	<ul style="list-style-type: none"> Bug #23104498: Fixed an issue in Performance Schema in reporting total memory used. (https://github.com/mysql/mysql-server/commit/20b6840df5452f47313c6f9a6ca075bfbc00a96b) Bug #22551677: Fixed an issue in Performance Schema that could lead to the database engine crashing when attempting to take it offline. (https://github.com/mysql/mysql-server/commit/05e2386eccd32b6b444b900c9f8a87a1d8d531e9) Bug #23550835, Bug #23298025, Bug #81464: Fixed an issue in Performance Schema that causes a database engine crash due to exceeding the capacity of an internal buffer. (https://github.com/mysql/mysql-server/commit/b4287f93857bf2f99b18fd06f555bbe5b12debfc, https://github.com/mysql/mysql-server/commit/b4287f93857bf2f99b18fd06f555bbe5b12debfc)
Aurora MySQL database engine updates 2019-11-25 (version 2.07.0) (p. 1153)	2.07.0	<ul style="list-style-type: none"> Bug #26251621: INCORRECT BEHAVIOR WITH TRIGGER AND GCOL Bug #22574695: ASSERTION `!TABLE (!TABLE->READ_SET BITMAP_IS_SET(TABLE->READ_SET, FIEL Bug #25966845: INSERT ON DUPLICATE KEY GENERATE A DEADLOCK Bug #23070734: CONCURRENT TRUNCATE TABLES CAUSE STALL Bug #26191879: FOREIGN KEY CASCADES USE EXCESSIVE MEMORY Bug #20989615: INNODB AUTO_INCREMENT PRODUCES SAME VALUE TWICE
Aurora MySQL database engine updates 2019-11-11 (version 2.05.0) (deprecated) (p. 1158)	2.05.0	<ul style="list-style-type: none"> Bug#23054591: PURGE BINARY LOGS TO is reading the whole binlog file and causing MySql to stall

Database engine update	Version	MySQL bugs fixed
Aurora MySQL database engine updates 2020-08-14 (version 2.04.9) (p. 1159)	2.04.9	<ul style="list-style-type: none"> Bug #23070734, Bug #80060: Concurrent TRUNCATE TABLEs cause stalls Bug #23103937: PS_TRUNCATE_ALL_TABLES() DOES NOT WORK IN SUPER_READ_ONLY MODE Bug#22551677: When taking the server offline, a race condition within the Performance Schema could lead to a server exit. Bug #27082268: Invalid FTS sync synchronization. BUG #12589870: Fixed an issues which causes a restart with multi-query statement when query cache is enabled. Bug #26402045: Certain cases of subquery materialization could cause a server exit. These queries now produce an error suggesting that materialization be disabled. Bug #18898433: Queries with many left joins were slow if join buffering was used (for example, using the block nested loop algorithm). Bug #25222337: A NULL virtual column field name in a virtual index caused a server exit during a field name comparison that occurs while populating virtual columns affected by a foreign key constraint. (https://github.com/mysql/mysql-server/commit/273d5c9d7072c63b6c47dbef6963d7dc491d5131) Bug #25053286: Executing a stored procedure containing a query that accessed a view could allocate memory that was not freed until the session ended. (https://github.com/mysql/mysql-server/commit/d7b37d4d141a95f577916448650c429f0d6e193d) Bug #25586773: Executing a stored procedure containing a statement that created a table from the contents of certain SELECT (https://dev.mysql.com/doc/refman/5.7/en/select.html) statements could result in a memory leak. (https://github.com/mysql/mysql-server/commit/88301e5adab65f6750f66af284be410c4369d0c1) Bug #26666274: INFINITE LOOP IN PERFORMANCE SCHEMA BUFFER CONTAINER. Bug #23550835, Bug #23298025, Bug #81464: A SELECT Performance Schema tables when an internal buffer was full could cause a server exit.
Aurora MySQL database engine updates 2019-09-19 (version 2.04.6) (p. 1166)	2.04.6	<ul style="list-style-type: none"> Bug#23054591: PURGE BINARY LOGS TO is reading the whole binlog file and causing MySql to stall
Aurora MySQL database engine updates 2019-05-02 (version 2.04.2) (p. 1172)	2.04.2	Bug #24829050 - INDEX_MERGE_INTERSECTION OPTIMIZATION CAUSES WRONG QUERY RESULTS

Database engine update	Version	MySQL bugs fixed
Aurora MySQL database engine updates 2018-10-11 (version 2.03) (deprecated) (p. 1182)	2.03	<ul style="list-style-type: none"> REVERSE SCAN ON A PARTITIONED TABLE DOES ICP - ORDER BY DESC (Bug #24929748). JSON_OBJECT CREATES INVALID JSON CODE (Bug#26867509). INSERTING LARGE JSON DATA TAKES AN INORDINATE AMOUNT OF TIME (Bug #22843444). PARTITIONED TABLES USE MORE MEMORY IN 5.7 THAN 5.6 (Bug #25080442).
Aurora MySQL database engine updates 2018-09-21 (version 2.02.4) (deprecated) (p. 1185)	2.02.4	<ul style="list-style-type: none"> BUG#13651665 INNODB MAY BE UNABLE TO LOAD TABLE DEFINITION AFTER RENAME BUG#21371070 INNODB: CANNOT ALLOCATE 0 BYTES. BUG#21378944 FTS ASSERT ENC.SRC_IILIST_PTR != NULL, FTS_OPTIMIZE_WORD(), OPTIMIZE TABLE BUG#21508537 ASSERTION FAILURE UT_A(!VICTIM_TRX->READ_ONLY) BUG#21983865 UNEXPECTED DEADLOCK WITH INNODB_AUTOINC_LOCK_MODE=0 BUG#22679185 INVALID INNODB FTS DOC ID DURING INSERT BUG#22899305 GCOLS: ASSERTION: !(COL->PRTYPE & 256). BUG#22956469 MEMORY LEAK INTRODUCED IN 5.7.8 IN MEMORY/INNODB/OSOFILe BUG#22996488 CRASH IN FTS_SYNC_INDEX WHEN DOING DDL IN A LOOP BUG#23014521 GCOL:INNODB: ASSERTION: !IS_V BUG#23021168 REPLICATION STOPS AFTER TRX IS ROLLED BACK ASYNC BUG#23052231 ASSERTION: ADD_AUTOINC < DICT_TABLE_GET_N_USER_COLS BUG#23149683 ROTATE INNODB MASTER KEY WITH KEYRING_OKV_CONF_DIR MISSING: SIGSEGV; SIGNAL 11 BUG#23762382 INSERT VALUES QUERY WITH JOIN IN A SELECT CAUSES INCORRECT BEHAVIOR BUG#25209512 CURRENT_TIMESTAMP PRODUCES ZEROS IN TRIGGER BUG#26626277 BUG IN "INSERT... ON DUPLICATE KEY UPDATE" QUERY BUG#26734162 INCORRECT BEHAVIOR WITH INSERT OF BLOB + ON DUPLICATE KEY UPDATE BUG#27460607 INCORRECT WHEN INSERT SELECT's SOURCE TABLE IS EMPTY
Aurora MySQL database engine updates 2018-05-03 (version 2.02) (deprecated) (p. 1191)	2.02.0	Left join returns incorrect results on the outer side (Bug #22833364)

MySQL bugs fixed by Aurora MySQL 1.x database engine updates

MySQL 5.6-compatible version Aurora contains all MySQL bug fixes through MySQL 5.6.10. The following table identifies additional MySQL bugs that have been fixed by Aurora MySQL database engine updates, and which update they were fixed in.

Database engine update	Version	MySQL bugs fixed
Aurora MySQL database engine updates 2021-03-18 (version 1.23.2) (p. 1198)	1.23.2	<ul style="list-style-type: none"> • <i>Replication:</i> While a SHOW BINLOG EVENTS statement was executing, any parallel transaction was blocked. The fix ensures that the SHOW BINLOG EVENTS process now only acquires a lock for the duration of calculating the file's end position, therefore parallel transactions are not blocked for long durations. (Bug #76618, Bug #20928790)
Aurora MySQL database engine updates 2020-09-02 (version 1.23.0) (p. 1200)	1.23.0	<ul style="list-style-type: none"> • Binlog events with ALTER TABLE ADD COLUMN ALGORITHM=QUICK will be rewritten as ALGORITHM=DEFAULT to be compatible with the community edition. • BUG #22350047: IF CLIENT KILLED AFTER ROLLBACK TO SAVEPOINT PREVIOUS STMTS COMMITTED • Bug #29915479: RUNNING COM_REGISTER_SLAVE WITHOUT COM_BINLOG_DUMP CAN RESULTS IN SERVER EXIT • Bug #30441969: BUG #29723340: MYSQL SERVER CRASH AFTER SQL QUERY WITH DATA ?AST • Bug #30628268: OUT OF MEMORY CRASH • Bug #27081349: UNEXPECTED BEHAVIOUR WHEN DELETE WITH SPATIAL FUNCTION • Bug #27230859: UNEXPECTED BEHAVIOUR WHILE HANDLING INVALID POLYGON" • Bug #27081349: UNEXPECTED BEHAVIOUR WHEN DELETE WITH SPATIAL" • Bug #26935001: ALTER TABLE AUTO_INCREMENT TRIES TO READ INDEX FROM DISCARDED TABLESPACE • Bug #29770705: SERVER CRASHED WHILE EXECUTING SELECT WITH SPECIFIC WHERE CLAUSE • Bug #27659490: SELECT USING DYNAMIC RANGE AND INDEX MERGE USE TOO MUCH MEMORY(OOM) • Bug #24786290: REPLICATION BREAKS AFTER BUG #74145 HAPPENS IN MASTER • Bug #27703912: EXCESSIVE MEMORY USAGE WITH MANY PREPARE • Bug #20527363: TRUNCATE TEMPORARY TABLE CRASH: ! DICT_TF2_FLAG_IS_SET(TABLE, DICT_TF2_TEMPORARY) • Bug#23103937 PS_TRUNCATE_ALL_TABLES() DOES NOT WORK IN SUPER_READ_ONLY MODE • Bug #25053286: USE VIEW WITH CONDITION IN PROCEDURE CAUSES INCORRECT BEHAVIOR (fixed in 5.6.36) • Bug #25586773: INCORRECT BEHAVIOR FOR CREATE TABLE SELECT IN A LOOP IN SP (fixed in 5.6.39)

Database engine update	Version	MySQL bugs fixed
		<ul style="list-style-type: none"> Bug #27407480: AUTOMATIC_SP_PRIVILEGES REQUIRES NEED THE INSERT PRIVILEGES FOR MYSQL.USER TABLE Bug #26997096: <code>relay_log_space</code> value is not updated in a synchronized manner so that its value is sometimes much higher than the actual disk space used by relay logs. Bug#15831300 <code>SLAVE_TYPE_CONVERSIONS=ALL_NON_LOSSY</code> NOT WORKING AS EXPECTED SSL Bug backport Bug #17087862, Bug #20551271 Bug #16894092: PERFORMANCE REGRESSION IN 5.6.6+ FOR <code>INSERT INTO ... SELECT ... FROM</code> (fixed in 5.6.15). Port a bug fix related to <code>SLAVE_TYPE_CONVERSIONS</code>.
Aurora MySQL database engine updates 2020-11-09 (version 1.22.3) (p. 1205)	1.22.3	<ul style="list-style-type: none"> Bug #26654685: A corrupt index ID encountered during a foreign key check raised an assertion Bug #15831300: By default, when promoting integers from a smaller type on the master to a larger type on the slave (for example, from a <code>SMALLINT</code> column on the master to a <code>BIGINT</code> column on the slave), the promoted values are treated as though they are signed. Now in such cases it is possible to modify or override this behavior using one or both of <code>ALL_SIGNED</code>, <code>ALL_UNSIGNED</code> in the set of values specified for the <code>slave_type_conversions</code> server system variable. For more information, see Row-based replication: attribute promotion and demotion, as well as the description of the variable. Bug #17449901: With <code>foreign_key_checks=0</code>, InnoDB permitted an index required by a foreign key constraint to be dropped, placing the table into an inconsistent and causing the foreign key check that occurs at table load to fail. InnoDB now prevents dropping an index required by a foreign key constraint, even with <code>foreign_key_checks=0</code>. The foreign key constraint must be removed before dropping the foreign key index. BUG #20768847: An <code>ALTER TABLE ... DROP INDEX</code> operation on a table with foreign key dependencies raised an assertion.
Aurora MySQL database engine updates 2019-11-25 (version 1.22.0) (p. 1208)	1.22.0	<ul style="list-style-type: none"> Bug#16346241 - SERVER CRASH IN <code>ITEM_PARAM::QUERY_VAL_STR</code> Bug#17733850 - <code>NAME_CONST()</code> CRASH IN <code>ITEM_NAME_CONST::ITEM_NAME_CONST()</code> Bug #20989615 - INNODB AUTO_INCREMENT PRODUCES SAME VALUE TWICE Bug #20181776 - ACCESS CONTROL DOESN'T MATCH MOST SPECIFIC HOST WHEN IT CONTAINS WILDCARD Bug #27326796 - MYSQL CRASH WITH INNODB ASSERTION FAILURE IN FILE PARSONS.CC Bug #20590013 - IF YOU HAVE A FULLTEXT INDEX AND DROP IT YOU CAN NO LONGER PERFORM ONLINE DDL

Database engine update	Version	MySQL bugs fixed
Aurora MySQL database engine updates 2019-11-25 (version 1.21.0) (p. 1211)	1.21.0	<ul style="list-style-type: none"> Bug #19929406: HANDLE_FATAL_SIGNAL (SIG=11) IN __MEMMOVE_SSSE3_BACK FROM STRING::COPY Bug #17059925: For UNION statements, the rows-examined value was calculated incorrectly. This was manifested as too-large values for the ROWS_EXAMINED column of Performance Schema statement tables (such as <code>events_statements_current</code>). Bug #11827369: Some queries with <code>SELECT ... FROM DUAL</code> nested subqueries raised an assertion. Bug #16311231: Incorrect results were returned if a query contained a subquery in an IN clause that contained an XOR operation in the WHERE clause.
Aurora MySQL database engine updates 2019-11-11 (version 1.20.0) (p. 1213)	1.20.0	<ul style="list-style-type: none"> Bug #19929406: HANDLE_FATAL_SIGNAL (SIG=11) IN __MEMMOVE_SSSE3_BACK FROM STRING::COPY Bug #17059925: For UNION statements, the rows-examined value was calculated incorrectly. This was manifested as too-large values for the ROWS_EXAMINED column of Performance Schema statement tables (such as <code>events_statements_current</code>). Bug #11827369: Some queries with <code>SELECT ... FROM DUAL</code> nested subqueries raised an assertion. Bug #16311231: Incorrect results were returned if a query contained a subquery in an IN clause that contained an XOR operation in the WHERE clause.
Aurora MySQL database engine updates 2019-09-19 (version 1.19.5) (p. 1214)	1.19.5	<ul style="list-style-type: none"> CVE-2018-2696 CVE-2015-4737 Bug #19929406: HANDLE_FATAL_SIGNAL (SIG=11) IN __MEMMOVE_SSSE3_BACK FROM STRING::COPY Bug #17059925: For UNION statements, the rows-examined value was calculated incorrectly. This was manifest as too-large values for the ROWS_EXAMINED column of Performance Schema statement tables (such as <code>events_statements_current</code>). Bug #11827369: Some queries with <code>SELECT ... FROM DUAL</code> nested subqueries raised an assertion. Bug #16311231: Incorrect results were returned if a query contained a subquery in an IN clause which contained an XOR operation in the WHERE clause.
Aurora MySQL database engine updates 2019-02-07 (version 1.19.0) (p. 1217)	1.19.0	<ul style="list-style-type: none"> BUG #32917: DETECT ORPHAN TEMP-POOL FILES, AND HANDLE GRACEFULLY BUG #63144 CREATE TABLE IF NOT EXISTS METADATA LOCK IS TOO RESTRICTIVE
Aurora MySQL database engine updates 2019-01-17 (p. 1219)	1.17.8	<ul style="list-style-type: none"> BUG #13418638: CREATE TABLE IF NOT EXISTS METADATA LOCK IS TOO RESTRICTIVE

Database engine update	Version	MySQL bugs fixed
Aurora MySQL database engine updates 2018-10-08 (p. 1220)	1.17.7	<ul style="list-style-type: none"> Drop index on a foreign key column leads to missing table. (Bug #16208542) Memory leak in add_derived_key(). (Bug #76349) For partitioned tables, queries could return different results depending on whether Index Merge was used. (Bug #16862316) Queries using the index_merge optimization (see Index merge optimization) could return invalid results when run against tables that were partitioned by HASH. (Bug #17588348)
Aurora MySQL database engine updates 2018-09-06 (p. 1221)	1.17.6	<ul style="list-style-type: none"> For an ALTER TABLE statement that renamed or changed the default value of a BINARY column, the alteration was done using a table copy and not in place. (Bug #67141, Bug #14735373, Bug #69580, Bug #17024290) An outer join between a regular table and a derived table that is implicitly grouped could cause a server exit. (Bug #16177639)
Aurora MySQL database engine updates 2018-03-13 (p. 1224)	1.17.0	<ul style="list-style-type: none"> LAST_INSERT_ID is replicated incorrectly if replication filters are used (Bug #69861) Query returns different results depending on whether INDEX_MERGE setting (Bug #16862316) Query proc re-execute of stored routine, inefficient query plan (Bug #16346367) InnoDB FTS : Assert in FTS_CACHE_APPEND_DELETED_DOC_IDS (Bug #18079671) Assert RBT_EMPTY(INDEX_CACHE->WORDS) in ALTER TABLE CHANGE COLUMN (Bug #17536995) InnoDB fulltext search doesn't find records when savepoints are involved (Bug #70333, Bug #17458835)
Aurora MySQL database engine updates 2017-11-20 (p. 1226)	1.15.1	<ul style="list-style-type: none"> Reverted — MySQL instance stalling "doing SYNC index" (Bug #73816) Reverted — Assert RBT_EMPTY(INDEX_CACHE->WORDS) in ALTER TABLE CHANGE COLUMN (Bug #17536995) Reverted — InnoDB Fulltext search doesn't find records when savepoints are involved (Bug #70333)
Aurora MySQL database engine updates 2017-10-24 (p. 1227)	1.15.0	<ul style="list-style-type: none"> CREATE USER accepts plugin and password hash, but ignores the password hash (Bug #78033) The partitioning engine adds fields to the read bit set to be able to return entries sorted from a partitioned index. This leads to the join buffer will try to read unneeded fields. Fixed by not adding all partitioning fields to the read_set, but instead only sort on the already set prefix fields in the read_set. Added a DBUG_ASSERT that if doing key_cmp, at least the first field must be read (Bug #16367691). MySQL instance stalling "doing SYNC index" (Bug #73816) Assert RBT_EMPTY(INDEX_CACHE->WORDS) in ALTER TABLE CHANGE COLUMN (Bug #17536995) InnoDB Fulltext search doesn't find records when savepoints are involved (Bug #70333)

Database engine update	Version	MySQL bugs fixed
Aurora MySQL database engine updates: 2018-03-13 (p. 1229)	1.14.4	<ul style="list-style-type: none"> Ignorable events don't work and are not tested (Bug #74683) NEW->OLD ASSERT FAILURE 'GTID_MODE > 0' (Bug #20436436)
Aurora MySQL database engine updates: 2017-08-07 (p. 1230)	1.14.0	A full-text search combined with derived tables (subqueries in the <code>FROM</code> clause) caused a server exit. Now, if a full-text operation depends on a derived table, the server produces an error indicating that a full-text search cannot be done on a materialized table. (Bug #68751, Bug #16539903)
Aurora MySQL database engine updates: 2017-05-15 (p. 1231)	1.13.0	<ul style="list-style-type: none"> Reloading a table that was evicted while empty caused an <code>AUTO_INCREMENT</code> value to be reset. (Bug #21454472, Bug #77743) An index record was not found on rollback due to inconsistencies in the <code>purge_node_t</code> structure. The inconsistency resulted in warnings and error messages such as "error in sec index entry update", "unable to purge a record", and "tried to purge sec index entry not marked for deletion". (Bug #19138298, Bug #70214, Bug #21126772, Bug #21065746) Wrong stack size calculation for <code>qsort</code> operation leads to stack overflow. (Bug #73979) Record not found in an index upon rollback. (Bug #70214, Bug #72419) <code>ALTER TABLE add column TIMESTAMP on update CURRENT_TIMESTAMP</code> inserts ZERO-datas (Bug #17392)
Aurora MySQL database engine updates: 2017-04-05 (p. 1232)	1.12.0	<ul style="list-style-type: none"> Reloading a table that was evicted while empty caused an <code>AUTO_INCREMENT</code> value to be reset. (Bug #21454472, Bug #77743) An index record was not found on rollback due to inconsistencies in the <code>purge_node_t</code> structure. The inconsistency resulted in warnings and error messages such as "error in sec index entry update", "unable to purge a record", and "tried to purge sec index entry not marked for deletion". (Bug #19138298, Bug #70214, Bug #21126772, Bug #21065746) Wrong stack size calculation for <code>qsort</code> operation leads to stack overflow. (Bug #73979) Record not found in an index upon rollback. (Bug #70214, Bug #72419) <code>ALTER TABLE add column TIMESTAMP on update CURRENT_TIMESTAMP</code> inserts ZERO-datas (Bug #17392)

Database engine update	Version	MySQL bugs fixed
Aurora MySQL database engine updates: 2017-02-23 (p. 1234)	1.11.0	<ul style="list-style-type: none"> • Running ALTER table DROP foreign key simultaneously with another DROP operation causes the table to disappear. (Bug #16095573) • Some INFORMATION_SCHEMA queries that used ORDER BY did not use a filesort optimization as they did previously. (Bug #16423536) • FOUND_ROWS () returns the wrong count of rows on a table. (Bug #68458) • The server fails instead of giving an error when too many temp tables are open. (Bug #18948649)
Aurora MySQL database engine updates: 2016-12-14 (p. 1236)	1.10.0	<ul style="list-style-type: none"> • UNION of derived tables returns wrong results with '1=0/false'-clauses. (Bug #69471) • Server crashes in ITEM_FUNC_GROUP_CONCAT::FIX_FIELDS on 2nd execution of stored procedure. (Bug #20755389) • Avoid MySQL queries from stalling for too long during FTS cache sync to disk by offloading the cache sync task to a separate thread, as soon as the cache size crosses 10% of the total size. (Bugs #22516559, #73816)
Aurora MySQL database engine updates: 2016-10-26 (p. 1238)	1.8.1	<ul style="list-style-type: none"> • OpenSSL changed the Diffie-Hellman key length parameters due to the LogJam issue. (Bug #18367167)
Aurora MySQL database engine updates: 2016-10-18 (p. 1238)	1.8.0	<ul style="list-style-type: none"> • When dropping all indexes on a column with multiple indexes, InnoDB failed to block a DROP INDEX operation when a foreign key constraint requires an index. (Bug #16896810) • Solve add foreign key constraint crash. (Bug #16413976) • Fixed a crash when fetching a cursor in a stored procedure, and analyzing or flushing the table at the same time. (Bug #18158639) • Fixed an auto-increment bug when a user alters a table to change the AUTO_INCREMENT value to less than the maximum auto-increment column value. (Bug # 16310273)

Database engine update	Version	MySQL bugs fixed
Aurora MySQL database engine updates: 2016-08-30 (p. 1240)	1.7.0	<ul style="list-style-type: none"> • Improve scalability by partitioning LOCK_grant lock. (Port WL #8355) • Opening cursor on SELECT in stored procedure causes segfault. (Port Bug #16499751) • MySQL gives the wrong result with some special usage. (Bug #11751794) • Crash in GET_SEL_ARG_FOR_KEYPART – caused by patch for bug #11751794. (Bug #16208709) • Wrong results for a simple query with GROUP BY. (Bug #17909656) • Extra rows on semijoin query with range predicates. (Bug #16221623) • Adding an ORDER BY clause following an IN subquery could cause duplicate rows to be returned. (Bug #16308085) • Crash with explain for a query with loose scan for GROUP BY, MyISAM. (Bug #16222245) • Loose index scan with quoted int predicate returns random data. (Bug #16394084) • If the optimizer was using a loose index scan, the server could exit while attempting to create a temporary table. (Bug #16436567) • COUNT(DISTINCT) should not count NULL values, but they were counted when the optimizer used loose index scan. (Bug #17222452) • If a query had both MIN() MAX() and aggregate_function(DISTINCT) (for example, SUM(DISTINCT)) and was executed using loose index scan, the result values of MIN() MAX() were set improperly. (Bug #17217128)
Aurora MySQL database engine updates: 2016-06-01 (p. 1241)	1.6.5	<ul style="list-style-type: none"> • SLAVE CAN'T CONTINUE REPLICATION AFTER MASTER'S CRASH RECOVERY (Port Bug #17632285)
Aurora MySQL database engine updates: 2016-04-06 (p. 1241)	1.6.0	<ul style="list-style-type: none"> • BACKPORT Bug #18694052 FIX FOR ASSERTION `! M_ORDERED_REC_BUFFER' FAILED TO 5.6 (Port Bug #18305270) • SEGV IN MEMCPY(), HA_PARTITION::POSITION (Port Bug # 18383840) • WRONG RESULTS WITH PARTITIONING,INDEX_MERGE AND NO PK (Port Bug # 18167648) • FLUSH TABLES FOR EXPORT: ASSERTION IN HA_PARTITION::EXTRA (Port Bug # 16943907) • SERVER CRASH IN VIRTUAL HA_ROWS HANDLER::MULTI_RANGE_READ_INFO_CONST (Port Bug # 16164031) • RANGE OPTIMIZER CRASHES IN SEL_ARG::RB_INSERT() (Port Bug # 16241773)

Database engine update	Version	MySQL bugs fixed
Aurora MySQL database engine updates: 2016-01-11 (p. 1243)	1.5.0	<ul style="list-style-type: none"> Addressed incomplete fix in MySQL full text search affecting tables where the database name begins with a digit. (Port Bug #17607956)
Aurora MySQL database engine updates: 2015-12-03 (p. 1243)	1.4	<ul style="list-style-type: none"> SEGV in FTSPARSE(). (Bug #16446108) InnoDB data dictionary is not updated while renaming the column. (Bug #19465984) FTS crash after renaming table to different database. (Bug #16834860) Failed preparing of trigger on truncated tables cause error 1054. (Bug #18596756) Metadata changes might cause problems with trigger execution. (Bug #18684393) Materialization is not chosen for long UTF8 VARCHAR field. (Bug #17566396) Poor execution plan when ORDER BY with limit X. (Bug #16697792) Backport bug #11765744 TO 5.1, 5.5 AND 5.6. (Bug #17083851) Mutex issue in SQL/SQL_SHOW.CC resulting in SIG6. Source likely FILL_VARIABLES. (Bug #20788853) Backport bug #18008907 to 5.5+ versions. (Bug #18903155) Adapt fix for a stack overflow error in MySQL 5.7. (Bug #19678930)

Database engine update	Version	MySQL bugs fixed
Aurora MySQL database engine updates: 2015-10-16 (p. 1245)	1.2, 1.3	<ul style="list-style-type: none"> Killing a query inside innodb causes it to eventually crash with an assertion. (Bug #1608883) For failure to create a new thread for the event scheduler, event execution, or new connection, no message was written to the error log. (Bug #16865959) If one connection changed its default database and simultaneously another connection executed SHOW PROCESSLIST, the second connection could access invalid memory when attempting to display the first connection's default database memory. (Bug #11765252) PURGE BINARY LOGS by design does not remove binary log files that are in use or active, but did not provide any notice when this occurred. (Bug #13727933) For some statements, memory leaks could result when the optimizer removed unneeded subquery clauses. (Bug #15875919) During shutdown, the server could attempt to lock an uninitialized mutex. (Bug #16016493) A prepared statement that used GROUP_CONCAT() and an ORDER BY clause that named multiple columns could cause the server to exit. (Bug #16075310) Performance Schema instrumentation was missing for replica worker threads. (Bug #16083949) STOP SLAVE could cause a deadlock when issued concurrently with a statement such as SHOW STATUS that retrieved the values for one or more of the status variables Slave_retried_transactions, Slave_heartbeat_period, Slave_received_heartbeats, Slave_last_heartbeat, or Slave_running. (Bug #16088188) A full-text query using Boolean mode could return zero results in some cases where the search term was a quoted phrase. (Bug #16206253) The optimizer's attempt to remove redundant subquery clauses raised an assertion when executing a prepared statement with a subquery in the ON clause of a join in a subquery. (Bug #16318585) GROUP_CONCAT unstable, crash in ITEM_SUM::CLEAN_UP_AFTER_REMOVAL. (Bug #16347450) Attempting to replace the default InnoDB full-text search (FTS) stopword list by creating an InnoDB table with the same structure as INFORMATION_SCHEMA.INNODB_FT_DEFAULT_STOPWORD would result in an error. (Bug #16373868) After the client thread on a worker performed a FLUSH TABLES WITH READ LOCK and was followed by some updates on the master, the worker hung when executing SHOW SLAVE STATUS. (Bug #16387720) When parsing a delimited search string such as "abc-def" in a full-text search, InnoDB now uses the same word delimiters as MyISAM. (Bug #16419661) Crash in FTS_AST_TERM_SET_WILDCARD. (Bug #16429306)

Database engine update	Version	MySQL bugs fixed
		<ul style="list-style-type: none"> • SEGFAULT in FTS_AST_VISIT() for FTS RQG test. (Bug #16435855) • For debug builds, when the optimizer removed an Item_ref pointing to a subquery, it caused a server exit. (Bug #16509874) • Full-text search on InnoDB tables failed on searches for literal phrases combined with + or - operators. (Bug #16516193) • START SLAVE failed when the server was started with the options --master-info-repository=TABLE relay-log-info-repository=TABLE and with autocommit set to 0, together with --skip-slave-start. (Bug #16533802) • Very large InnoDB full-text search (FTS) results could consume an excessive amount of memory. (Bug #16625973) • In debug builds, an assertion could occur in OPT_CHECK_ORDER_BY when using binary directly in a search string, as binary might include NULL bytes and other non-meaningful characters. (Bug #16766016) • For some statements, memory leaks could result when the optimizer removed unneeded subquery clauses. (Bug #16807641) • It was possible to cause a deadlock after issuing FLUSH TABLES WITH READ LOCK by issuing STOP SLAVE in a new connection to the worker, then issuing SHOW SLAVE STATUS using the original connection. (Bug #16856735) • GROUP_CONCAT() with an invalid separator could cause a server exit. (Bug #16870783) • The server did excessive locking on the LOCK_active_mi and active_mi->rli->data_lock mutexes for any SHOW STATUS LIKE 'pattern' statement, even when the pattern did not match status variables that use those mutexes (Slave_heartbeat_period, Slave_last_heartbeat, Slave_received_heartbeats, Slave_retried_transactions, Slave_running). (Bug #16904035) • A full-text search using the IN BOOLEAN MODE modifier would result in an assertion failure. (Bug #16927092) • Full-text search on InnoDB tables failed on searches that used the + boolean operator. (Bug #17280122) • 4-way deadlock: zombies, purging binlogs, show processlist, show binlogs. (Bug #17283409) • When an SQL thread which was waiting for a commit lock was killed and restarted it caused a transaction to be skipped on worker. (Bug #17450876) • An InnoDB full-text search failure would occur due to an "unended" token. The string and string length should be passed for string comparison. (Bug #17659310) • Large numbers of partitioned InnoDB tables could consume much more memory when used in MySQL 5.6 or 5.7 than the memory used by the same tables used in previous releases of the MySQL Server. (Bug #17780517) • For full-text queries, a failure to check that num_token is less than max_proximity_item could result in an assertion. (Bug #18233051)

Database engine update	Version	MySQL bugs fixed
		<ul style="list-style-type: none"> Certain queries for the INFORMATION_SCHEMA TABLES and COLUMNS tables could lead to excessive memory use when there were large numbers of empty InnoDB tables. (Bug #18592390) When committing a transaction, a flag is now used to check whether a thread has been created, rather than checking the thread itself, which uses more resources, particularly when running the server with master_info_repository=TABLE. (Bug #18684222) If a client thread on a worker executed FLUSH TABLES WITH READ LOCK while the master executed a DML, executing SHOW SLAVE STATUS in the same client became blocked, causing a deadlock. (Bug #19843808) Ordering by a GROUP_CONCAT() result could cause a server exit. (Bug #19880368)
Aurora MySQL database engine updates: 2015-08-24 (p. 1247)	1.1	<ul style="list-style-type: none"> InnoDB databases with names beginning with a digit cause a full-text search (FTS) parser error. (Bug #17607956) InnoDB full-text searches fail in databases whose names began with a digit. (Bug #17161372) For InnoDB databases on Windows, the full-text search (FTS) object ID is not in the expected hexadecimal format. (Bug #16559254) A code regression introduced in MySQL 5.6 negatively impacted DROP TABLE and ALTER TABLE performance. This could cause a performance drop between MySQL Server 5.5.x and 5.6.x. (Bug #16864741)

Security vulnerabilities fixed in Amazon Aurora MySQL

Common Vulnerabilities and Exposures (CVE) is a list of entries for publicly known cybersecurity vulnerabilities. Each entry contains an identification number, a description, and at least one public reference.

You can find on this page a list of security vulnerabilities fixed in Aurora MySQL. For general information about security for Aurora, see [Security in Amazon Aurora \(p. 1706\)](#). For additional security information for Aurora MySQL, see [Security with Amazon Aurora MySQL \(p. 774\)](#).

We recommend that you always upgrade to the latest Aurora release to be protected against known vulnerabilities. You can use this page to verify whether a particular version of Aurora MySQL has a fix for a specific security vulnerability. If your cluster doesn't have the security fix, you can see which Aurora MySQL version you should upgrade to for that fix.

Any CVEs fixed in Aurora MySQL version 1 and 2 are also listed in the release notes for that version:

- [Database engine updates for Amazon Aurora MySQL version 1 \(p. 1196\)](#)
- [Database engine updates for Amazon Aurora MySQL version 2 \(p. 1108\)](#)

Note

The initial release of Aurora MySQL version 3 includes all CVEs fixed up to community MySQL 8.0.23. For future CVEs that are fixed, look for them listed here and in the Aurora MySQL version 3 release notes.

CVEs and minimum fixed Aurora MySQL versions

- [CVE-2021-35624: 2.10.2](#)
- [CVE-2021-35604: 2.10.2](#)
- [CVE-2021-23841: 2.10.0, 2.09.3, 1.23.3](#)
- [CVE-2021-3712: 2.09.3](#)
- [CVE-2021-3449: 2.10.0, 2.09.3, 1.23.3](#)
- [CVE-2021-2390: 2.10.2](#)
- [CVE-2021-2389: 2.10.2](#)
- [CVE-2021-2385: 2.10.2](#)
- [CVE-2021-2356: 2.10.2](#)
- [CVE-2021-2307: 2.10.1, 2.09.3, 1.23.4](#)
- [CVE-2021-2226: 2.10.1, 2.09.3, 1.23.4](#)
- [CVE-2021-2194: 2.10.1](#)
- [CVE-2021-2174: 2.10.1, 2.09.3](#)
- [CVE-2021-2171: 2.10.1, 2.09.3](#)
- [CVE-2021-2169: 2.10.1, 2.09.3](#)
- [CVE-2021-2166: 2.10.1, 2.09.3](#)
- [CVE-2021-2160: 2.10.1, 1.23.4](#)
- [CVE-2021-2154: 2.10.1, 2.09.3, 1.23.4](#)
- [CVE-2021-2060: 2.10.1, 2.09.3, 1.23.4](#)
- [CVE-2021-2032: 2.10.1, 2.09.3, 1.23.4](#)
- [CVE-2021-2001: 2.10.1, 2.09.3, 1.23.4](#)
- [CVE-2020-28196: 2.10.0, 2.09.3, 1.23.3](#)
- [CVE-2020-14867: 1.23.2, 1.22.4](#)
- [CVE-2020-14812: 2.09.2, 2.07.4, 1.23.2, 1.22.4](#)
- [CVE-2020-14793: 2.09.2, 2.07.4, 1.23.2, 1.22.4](#)
- [CVE-2020-14790: 2.10.0, 2.09.2, 2.07.4](#)
- [CVE-2020-14776: 2.10.0](#)
- [CVE-2020-14775: 2.09.2, 2.07.4](#)
- [CVE-2020-14769: 2.09.3, 2.09.2, 2.07.4, 1.23.2, 1.22.4](#)
- [CVE-2020-14765: 2.09.2, 2.07.4, 1.23.2, 1.22.4](#)
- [CVE-2020-14760: 2.09.2, 2.07.4](#)
- [CVE-2020-14672: 2.09.2, 2.07.4, 1.23.2, 1.22.4](#)
- [CVE-2020-14567: 2.10.0, 2.09.1, 2.08.3, 2.07.3](#)
- [CVE-2020-14559: 2.10.0, 2.09.1, 2.08.3, 2.07.3, 1.23.1, 1.22.3](#)
- [CVE-2020-14553: 2.10.0, 2.09.1, 2.08.3, 2.07.3](#)
- [CVE-2020-14547: 2.10.0, 2.09.1, 2.08.3, 2.07.3](#)
- [CVE-2020-14540: 2.10.0, 2.09.1, 2.08.3, 2.07.3](#)
- [CVE-2020-14539: 2.10.0, 1.23.1, 1.22.3](#)
- [CVE-2020-2812: 2.09.1, 2.08.3, 2.07.3, 1.22.3](#)
- [CVE-2020-2806: 2.09.1, 2.08.3, 2.07.3](#)

- [CVE-2020-2780: 2.09.1, 2.08.3, 2.07.3, 1.22.3](#)
- [CVE-2020-2765: 2.09.1, 2.08.3, 2.07.3](#)
- [CVE-2020-2763: 2.09.1, 2.08.3, 2.07.3, 1.22.3](#)
- [CVE-2020-2760: 2.09.1, 2.08.3, 2.07.3, 2.04.9](#)
- [CVE-2019-2740: 2.07.3](#)
- [CVE-2020-2579: 2.09.1, 2.08.3, 2.07.3, 1.22.3](#)
- [CVE-2020-1971: 2.09.2, 2.07.4, 1.23.2, 1.22.4](#)
- [CVE-2019-17543: 2.10.2, 2.09.3, 2.07.7](#)
- [CVE-2019-5443: 2.08.0, 2.04.9](#)
- [CVE-2019-3822: 2.08.0, 2.04.9](#)
- [CVE-2019-2960: 2.10.2, 2.09.3, 2.07.7](#)
- [CVE-2019-2948: 2.09.0](#)
- [CVE-2019-2924: 2.07.0, 2.04.9, 1.22.0](#)
- [CVE-2019-2923: 2.07.0, 2.04.9, 1.22.0](#)
- [CVE-2019-2922: 2.07.0, 2.04.9, 1.22.0](#)
- [CVE-2019-2911: 2.09.0, 2.04.9, 1.23.0](#)
- [CVE-2019-2910: 2.07.0, 2.04.9, 1.22.0](#)
- [CVE-2019-2805: 2.06.0, 2.04.9, 1.22.0](#)
- [CVE-2019-2778: 2.06.0, 2.04.9](#)
- [CVE-2019-2758: 2.06.0, 2.04.9](#)
- [CVE-2019-2740: 2.04.9, 1.22.0](#)
- [CVE-2019-2739: 2.06.0, 2.04.9](#)
- [CVE-2019-2731: 2.09.0](#)
- [CVE-2019-2730: 2.06.0, 2.04.9, 1.22.0](#)
- [CVE-2019-2628: 2.04.9](#)
- [CVE-2019-2581: 2.09.0](#)
- [CVE-2019-2537: 2.09.0, 1.23.0](#)
- [CVE-2019-2534: 2.05.0, 2.04.3 \(p. 1170\), 1.21.0, 1.20.0, 1.19.1](#)
- [CVE-2019-2482: 2.09.0](#)
- [CVE-2019-2434: 2.09.0](#)
- [CVE-2019-2420: 2.09.0](#)
- [CVE-2018-3284: 2.09.0](#)
- [CVE-2018-3251: 2.10.0](#)
- [CVE-2018-3156: 2.10.0](#)
- [CVE-2018-3155: 2.05.0, 2.04.3 \(p. 1170\)](#)
- [CVE-2018-3143: 2.10.0, 1.23.2](#)
- [CVE-2018-3065: 2.09.0](#)
- [CVE-2018-3064: 2.06.0, 2.04.9, 1.22.0](#)
- [CVE-2018-3058: 2.06.0, 2.04.9, 1.22.0](#)
- [CVE-2018-3056: 2.05.0, 2.04.4 \(p. 1169\)](#)
- [CVE-2018-2813: 2.04.9](#)
- [CVE-2018-2787: 2.09.0, 1.23.0](#)
- [CVE-2018-2786: 2.06.0, 2.04.9](#)
- [CVE-2018-2784: 2.09.0, 1.23.0](#)
- [CVE-2018-2696: 2.05.0, 2.04.5 \(p. 1168\), 1.21.0, 1.20.0, 1.19.5](#)
- [CVE-2018-2645: 2.09.0, 1.23.0](#)

- [CVE-2018-2640: 2.09.0, 1.23.0](#)
- [CVE-2018-2612: 2.05.0, 2.04.3 \(p. 1170\), 1.21.0, 1.20.0, 1.19.1](#)
- [CVE-2018-2562: 2.05.0, 2.04.4 \(p. 1169\), 1.21.0, 1.20.0, 1.19.2](#)
- [CVE-2018-0734: 2.05.0, 2.04.3 \(p. 1170\), 1.21.0, 1.20.0, 1.19.1](#)
- [CVE-2017-3653: 2.06.0, 2.04.9, 1.22.0](#)
- [CVE-2017-3599: 2.05.0, 2.04.3 \(p. 1170\), 1.21.0, 1.20.0, 1.19.1](#)
- [CVE-2017-3465: 2.06.0, 2.04.9](#)
- [CVE-2017-3464: 1.22.0, 2.04.9](#)
- [CVE-2017-3455: 2.06.0, 2.04.9](#)
- [CVE-2017-3329: 2.05.0, 2.04.4 \(p. 1169\), 1.21.0, 1.20.0, 1.19.2](#)
- [CVE-2017-3244: 2.06.0, 2.04.9, 1.22.0](#)
- [CVE-2016-5612: 2.06.0, 2.04.9, 1.22.0](#)
- [CVE-2016-5439: 1.22.0](#)
- [CVE-2016-0606: 1.22.0](#)
- [CVE-2015-4904: 1.22.0](#)
- [CVE-2015-4879: 1.22.0](#)
- [CVE-2015-4864: 1.22.0](#)
- [CVE-2015-4830: 1.22.0](#)
- [CVE-2015-4826: 1.22.0](#)
- [CVE-2015-4737: 1.21.0, 1.20.0, 1.19.5](#)
- [CVE-2015-2620: 1.22.0](#)
- [CVE-2015-0382: 1.22.0](#)
- [CVE-2015-0381: 1.22.0](#)
- [CVE-2014-6555: 1.22.0](#)
- [CVE-2014-6489: 1.22.0](#)
- [CVE-2014-4260: 1.22.0](#)
- [CVE-2014-4258: 1.22.0](#)
- [CVE-2014-2444: 1.22.0](#)
- [CVE-2014-2436: 1.22.0](#)
- [CVE-2014-0393: 1.22.0](#)
- [CVE-2013-5908: 1.22.0](#)
- [CVE-2013-5881: 1.22.0](#)
- [CVE-2013-5807: 1.22.0](#)
- [CVE-2013-3811: 1.22.0](#)
- [CVE-2013-3807: 1.22.0](#)
- [CVE-2013-3806: 1.22.0](#)
- [CVE-2013-3804: 1.22.0](#)
- [CVE-2013-2381: 1.22.0](#)
- [CVE-2013-2378: 1.22.0](#)
- [CVE-2013-2375: 1.22.0](#)
- [CVE-2013-1523: 1.22.0](#)
- [CVE-2012-5615: 1.22.0](#)

Working with Amazon Aurora PostgreSQL

Amazon Aurora PostgreSQL is a fully managed, PostgreSQL-compatible, and ACID-compliant relational database engine that combines the speed and reliability of high-end commercial databases with the simplicity and cost-effectiveness of open-source databases. Aurora PostgreSQL is a drop-in replacement for PostgreSQL and makes it simple and cost-effective to set up, operate, and scale your new and existing PostgreSQL deployments, thus freeing you to focus on your business and applications. Amazon RDS provides administration for Aurora by handling routine database tasks such as provisioning, patching, backup, recovery, failure detection, and repair. Amazon RDS also provides push-button migration tools to convert your existing Amazon RDS for PostgreSQL applications to Aurora PostgreSQL.

Aurora PostgreSQL can work with many industry standards. For example, you can use Aurora PostgreSQL databases to build HIPAA-compliant applications and to store healthcare related information, including protected health information (PHI), under a completed Business Associate Agreement (BAA) with AWS.

Aurora PostgreSQL is FedRAMP HIGH eligible. For details about AWS and compliance efforts, see [AWS services in scope by compliance program](#).

Topics

- [Security with Amazon Aurora PostgreSQL \(p. 1276\)](#)
- [Updating applications to connect to Aurora PostgreSQL DB clusters using new SSL/TLS certificates \(p. 1280\)](#)
- [Migrating data to Amazon Aurora with PostgreSQL compatibility \(p. 1283\)](#)
- [Working with Babelfish for Aurora PostgreSQL \(p. 1297\)](#)
- [Managing Amazon Aurora PostgreSQL \(p. 1360\)](#)
- [Tuning with wait events for Aurora PostgreSQL \(p. 1376\)](#)
- [Best practices with Amazon Aurora PostgreSQL \(p. 1423\)](#)
- [Replication with Amazon Aurora PostgreSQL \(p. 1431\)](#)
- [Integrating Amazon Aurora PostgreSQL with other AWS services \(p. 1437\)](#)
- [Importing Amazon S3 data into an Aurora PostgreSQL DB cluster \(p. 1438\)](#)
- [Exporting data from an Aurora PostgreSQL DB cluster to Amazon S3 \(p. 1450\)](#)
- [Managing query execution plans for Aurora PostgreSQL \(p. 1460\)](#)
- [Publishing Aurora PostgreSQL logs to Amazon CloudWatch Logs \(p. 1487\)](#)
- [Using machine learning \(ML\) with Aurora PostgreSQL \(p. 1494\)](#)
- [Fast recovery after failover with cluster cache management for Aurora PostgreSQL \(p. 1513\)](#)
- [Invoking an AWS Lambda function from an Aurora PostgreSQL DB cluster \(p. 1517\)](#)
- [Using the oracle_fdw extension to access foreign data in Aurora PostgreSQL \(p. 1527\)](#)
- [Managing PostgreSQL partitions with the pg_partman extension \(p. 1530\)](#)
- [Using Kerberos authentication with Aurora PostgreSQL \(p. 1534\)](#)
- [Amazon Aurora PostgreSQL reference \(p. 1547\)](#)

- [Amazon Aurora PostgreSQL updates \(p. 1597\)](#)

Security with Amazon Aurora PostgreSQL

Security for Amazon Aurora PostgreSQL is managed at three levels:

- To control who can perform Amazon RDS management actions on Aurora PostgreSQL DB clusters and DB instances, you use AWS Identity and Access Management (IAM). When you connect to AWS using IAM credentials, your AWS account must have IAM policies that grant the permissions required to perform Amazon RDS management operations. For more information, see [Identity and access management in Amazon Aurora \(p. 1724\)](#).

If you are using IAM to access the Amazon RDS console, you must first sign on to the AWS Management Console with your IAM user credentials. Then go to the Amazon RDS console at <https://console.aws.amazon.com/rds/>.

- Aurora DB clusters must be created in an Amazon Virtual Private Cloud (VPC). To control which devices and Amazon EC2 instances can open connections to the endpoint and port of the DB instance for Aurora DB clusters in a VPC, you use a VPC security group. These endpoint and port connections can be made using Secure Sockets Layer (SSL) and Transport Layer Security (TLS). In addition, firewall rules at your company can control whether devices running at your company can open connections to a DB instance. For more information on VPCs, see [Amazon Virtual Private Cloud VPCs and Amazon Aurora \(p. 1787\)](#).

The supported VPC tenancy depends on the DB instance class used by your Aurora PostgreSQL DB clusters. With default VPC tenancy, the VPC runs on shared hardware. With dedicated VPC tenancy, the VPC runs on a dedicated hardware instance. The burstable performance DB instance classes support default VPC tenancy only. The burstable performance DB instance classes include the db.t3 and db.t4g DB instance classes. All other Aurora PostgreSQL DB instance classes support both default and dedicated VPC tenancy.

For more information about instance classes, see [Aurora DB instance classes \(p. 54\)](#). For more information about default and dedicated VPC tenancy, see [Dedicated instances](#) in the [Amazon Elastic Compute Cloud User Guide](#).

- To authenticate login and permissions for an Amazon Aurora DB cluster, you can take the same approach as with a stand-alone instance of PostgreSQL.

Commands such as `CREATE ROLE`, `ALTER ROLE`, `GRANT`, and `REVOKE` work just as they do in on-premises databases, as does directly modifying database schema tables. For more information, see [Client authentication](#) in the PostgreSQL documentation.

Note

The Salted Challenge Response Authentication Mechanism (SCRAM) is not supported with Aurora PostgreSQL.

Note

For more information, see [Security in Amazon Aurora \(p. 1706\)](#).

When you create an Amazon Aurora PostgreSQL DB instance, the master user has the following default privileges:

- `LOGIN`
- `NOSUPERUSER`
- `INHERIT`
- `CREATEDB`

- `CREATEROLE`
- `NOREPLICATION`
- `VALID UNTIL 'infinity'`

To provide management services for each DB cluster, the `rdsadmin` user is created when the DB cluster is created. Attempting to drop, rename, change the password, or change privileges for the `rdsadmin` account will result in an error.

Restricting password management

You can restrict who can manage database user passwords to a special role. By doing this, you can have more control over password management on the client side.

You enable restricted password management with the static parameter `rds.restrict_password_commands` and use a role called `rds_password`. When the parameter `rds.restrict_password_commands` is set to 1, only users that are members of the `rds_password` role can run certain SQL commands. The restricted SQL commands are commands that modify database user passwords and password expiration time.

To use restricted password management, your DB cluster must be running Amazon Aurora for PostgreSQL 10.6 or higher. Because the `rds.restrict_password_commands` parameter is static, changing this parameter requires a database restart.

When a database has restricted password management enabled, if you try to run restricted SQL commands you get the following error: `ERROR: must be a member of rds_password to alter passwords.`

Following are some examples of SQL commands that are restricted when restricted password management is enabled.

```
postgres=> CREATE ROLE myrole WITH PASSWORD 'mypassword';
postgres=> CREATE ROLE myrole WITH PASSWORD 'mypassword' VALID UNTIL '2020-01-01';
postgres=> ALTER ROLE myrole WITH PASSWORD 'mypassword' VALID UNTIL '2020-01-01';
postgres=> ALTER ROLE myrole WITH PASSWORD 'mypassword';
postgres=> ALTER ROLE myrole VALID UNTIL '2020-01-01';
postgres=> ALTER ROLE myrole RENAME TO myrole2;
```

Some `ALTER ROLE` commands that include `RENAME TO` might also be restricted. They might be restricted because renaming a PostgreSQL role that has an MD5 password clears the password.

The `rds_superuser` role has membership for the `rds_password` role by default, and you can't change this. You can give other roles membership for the `rds_password` role by using the `GRANT` SQL command. We recommend that you give membership to `rds_password` to only a few roles that you use solely for password management. These roles require the `CREATEROLE` attribute to modify other roles.

Make sure that you verify password requirements such as expiration and needed complexity on the client side. We recommend that you restrict password-related changes by using your own client-side utility. This utility should have a role that is a member of `rds_password` and has the `CREATEROLE` role attribute.

Securing Aurora PostgreSQL data with SSL/TLS

Amazon RDS supports Secure Socket Layer (SSL) and Transport Layer Security (TLS) encryption for Aurora PostgreSQL DB clusters. Using SSL/TLS, you can encrypt a connection between your applications and your Aurora PostgreSQL DB clusters. You can also force all connections to your Aurora PostgreSQL

DB cluster to use SSL/TLS. Amazon Aurora PostgreSQL supports Transport Layer Security (TLS) versions 1.1 and 1.2. We recommend using TLS 1.2 for encrypted connections.

For general information about SSL/TLS support and PostgreSQL databases, see [SSL support](#) in the PostgreSQL documentation. For information about using an SSL/TLS connection over JDBC, see [Configuring the client](#) in the PostgreSQL documentation.

Topics

- [Requiring an SSL/TLS connection to an Aurora PostgreSQL DB cluster \(p. 1278\)](#)
- [Determining the SSL/TLS connection status \(p. 1279\)](#)

SSL/TLS support is available in all AWS Regions for Aurora PostgreSQL. Amazon RDS creates an SSL/TLS certificate for your Aurora PostgreSQL DB cluster when the DB cluster is created. If you enable SSL/TLS certificate verification, then the SSL/TLS certificate includes the DB cluster endpoint as the Common Name (CN) for the SSL/TLS certificate to guard against spoofing attacks.

To connect to an Aurora PostgreSQL DB cluster over SSL/TLS

1. Download the certificate.

For information about downloading certificates, see [Using SSL/TLS to encrypt a connection to a DB cluster \(p. 1714\)](#).

2. Import the certificate into your operating system.
3. Connect to your Aurora PostgreSQL DB cluster over SSL/TLS.

When you connect using SSL/TLS, your client can choose to verify the certificate chain or not. If your connection parameters specify `sslmode=verify-ca` or `sslmode=verify-full`, then your client requires the RDS CA certificates to be in their trust store or referenced in the connection URL. This requirement is to verify the certificate chain that signs your database certificate.

When a client, such as `psql` or JDBC, is configured with SSL/TLS support, the client first tries to connect to the database with SSL/TLS by default. If the client can't connect with SSL/TLS, it reverts to connecting without SSL/TLS. The default `sslmode` mode used is different between libpq-based clients (such as `psql`) and JDBC. The libpq-based clients default to `prefer`, where JDBC clients default to `verify-full`.

Use the `sslrootcert` parameter to reference the certificate, for example `sslrootcert=rds-ssl-ca-cert.pem`.

The following is an example of using `psql` to connect to an Aurora PostgreSQL DB cluster.

```
$ psql -h testpg.cdhmuqifdpib.us-east-1.rds.amazonaws.com -p 5432 \
  "dbname=testpg user=testuser sslrootcert=rds-ca-2015-root.pem sslmode=verify-full"
```

Requiring an SSL/TLS connection to an Aurora PostgreSQL DB cluster

You can require that connections to your Aurora PostgreSQL DB cluster use SSL/TLS by using the `rds.force_ssl` parameter. By default, the `rds.force_ssl` parameter is set to 0 (off). You can set the `rds.force_ssl` parameter to 1 (on) to require SSL/TLS for connections to your DB cluster. Updating the `rds.force_ssl` parameter also sets the PostgreSQL `ssl` parameter to 1 (on) and modifies your DB cluster's `pg_hba.conf` file to support the new SSL/TLS configuration.

You can set the `rds.force_ssl` parameter value by updating the DB cluster parameter group for your DB cluster. If the DB cluster parameter group isn't the default one, and the `ssl` parameter is already set

to 1 when you set `rds.force_ssl` to 1, you don't need to reboot your DB cluster. Otherwise, you must reboot your DB cluster for the change to take effect. For more information on parameter groups, see [Working with DB parameter groups and DB cluster parameter groups \(p. 339\)](#).

When the `rds.force_ssl` parameter is set to 1 for a DB cluster, you see output similar to the following when you connect, indicating that SSL/TLS is now required:

```
$ psql postgres -h SOMEHOST.amazonaws.com -p 8192 -U someuser
psql (9.3.12, server 9.4.4)
WARNING: psql major version 9.3, server major version 9.4.
Some psql features might not work.
SSL connection (cipher: DHE-RSA-AES256-SHA, bits: 256)
Type "help" for help.

postgres=>
```

Determining the SSL/TLS connection status

The encrypted status of your connection is shown in the logon banner when you connect to the DB cluster.

```
Password for user master:
psql (9.3.12)
SSL connection (cipher: DHE-RSA-AES256-SHA, bits: 256)
Type "help" for help.

postgres=>
```

You can also load the `sslinfo` extension and then call the `ssl_is_used()` function to determine if SSL/TLS is being used. The function returns `t` if the connection is using SSL/TLS, otherwise it returns `f`.

```
postgres=> create extension sslinfo;
CREATE EXTENSION

postgres=> select ssl_is_used();
 ssl_is_used
 -----
 t
(1 row)
```

You can use the `select ssl_cipher()` command to determine the SSL/TLS cipher:

```
postgres=> select ssl_cipher();
ssl_cipher
-----
DHE-RSA-AES256-SHA
(1 row)
```

If you enable `set rds.force_ssl` and restart your DB cluster, non-SSL connections are refused with the following message:

```
$ export PGSSLMODE=disable
$ psql postgres -h SOMEHOST.amazonaws.com -p 8192 -U someuser
psql: FATAL: no pg_hba.conf entry for host "host.ip", user "someuser", database "postgres",
      SSL off
$
```

For information about the `sslmode` option, see [Database connection control functions](#) in the PostgreSQL documentation.

Updating applications to connect to Aurora PostgreSQL DB clusters using new SSL/TLS certificates

As of September 19, 2019, Amazon RDS has published new Certificate Authority (CA) certificates for connecting to your Aurora DB clusters using Secure Socket Layer or Transport Layer Security (SSL/TLS). Following, you can find information about updating your applications to use the new certificates.

This topic can help you to determine whether any client applications use SSL/TLS to connect to your DB clusters. If they do, you can further check whether those applications require certificate verification to connect.

Note

Some applications are configured to connect to Aurora PostgreSQL DB clusters only if they can successfully verify the certificate on the server.

For such applications, you must update your client application trust stores to include the new CA certificates.

After you update your CA certificates in the client application trust stores, you can rotate the certificates on your DB clusters. We strongly recommend testing these procedures in a development or staging environment before implementing them in your production environments.

For more information about certificate rotation, see [Rotating your SSL/TLS certificate \(p. 1715\)](#). For more information about downloading certificates, see [Using SSL/TLS to encrypt a connection to a DB cluster \(p. 1714\)](#). For information about using SSL/TLS with PostgreSQL DB clusters, see [Securing Aurora PostgreSQL data with SSL/TLS \(p. 1277\)](#).

Topics

- [Determining whether applications are connecting to Aurora PostgreSQL DB clusters using SSL \(p. 1280\)](#)
- [Determining whether a client requires certificate verification in order to connect \(p. 1281\)](#)
- [Updating your application trust store \(p. 1281\)](#)
- [Using SSL/TLS connections for different types of applications \(p. 1282\)](#)

Determining whether applications are connecting to Aurora PostgreSQL DB clusters using SSL

Check the DB cluster configuration for the value of the `rds.force_ssl` parameter. By default, the `rds.force_ssl` parameter is set to 0 (off). If the `rds.force_ssl` parameter is set to 1 (on), clients are required to use SSL/TLS for connections. For more information about parameter groups, see [Working with DB parameter groups and DB cluster parameter groups \(p. 339\)](#).

If `rds.force_ssl` isn't set to 1 (on), query the `pg_stat_ssl` view to check connections using SSL. For example, the following query returns only SSL connections and information about the clients using SSL.

```
select datname, username, ssl, client_addr from pg_stat_ssl inner join pg_stat_activity on pg_stat_ssl.pid = pg_stat_activity.pid where ssl is true and username <> 'rdsadmin';
```

Only rows using SSL/TLS connections are displayed with information about the connection. The following is sample output.

```
datname | username | ssl | client_addr
-----+-----+-----+
benchdb | pgadmin | t | 53.95.6.13
postgres | pgadmin | t | 53.95.6.13
(2 rows)
```

The preceding query displays only the current connections at the time of the query. The absence of results doesn't indicate that no applications are using SSL connections. Other SSL connections might be established at a different time.

Determining whether a client requires certificate verification in order to connect

When a client, such as `psql` or JDBC, is configured with SSL support, the client first tries to connect to the database with SSL by default. If the client can't connect with SSL, it reverts to connecting without SSL. The default `sslmode` mode used is different between libpq-based clients (such as `psql`) and JDBC. The libpq-based clients default to `prefer`, where JDBC clients default to `verify-full`. The certificate on the server is verified only when `sslrootcert` is provided with `sslmode` set to `require`, `verify-ca`, or `verify-full`. An error is thrown if the certificate is invalid.

Use `PGSSLROOTCERT` to verify the certificate with the `PGSSLMODE` environment variable, with `PGSSLMODE` set to `require`, `verify-ca`, or `verify-full`.

```
PGSSLMODE=require PGSSLROOTCERT=/fullpath/rds-ca-2019-root.pem psql -h pgdbidentifier.cxxxxxxxxx.us-east-2.rds.amazonaws.com -U primaryuser -d postgres
```

Use the `sslrootcert` argument to verify the certificate with `sslmode` in connection string format, with `sslmode` set to `require`, `verify-ca`, or `verify-full`.

```
psql "host=pgdbidentifier.cxxxxxxxxx.us-east-2.rds.amazonaws.com sslmode=require sslrootcert=/full/path/rds-ca-2019-root.pem user=primaryuser dbname=postgres"
```

For example, in the preceding case, if you use an invalid root certificate, you see an error similar to the following on your client.

```
psql: SSL error: certificate verify failed
```

Updating your application trust store

For information about updating the trust store for PostgreSQL applications, see [Secure TCP/IP connections with SSL in the PostgreSQL documentation](#).

Note

When you update the trust store, you can retain older certificates in addition to adding the new certificates.

Updating your application trust store for JDBC

You can update the trust store for applications that use JDBC for SSL/TLS connections.

To update the trust store for JDBC applications

1. Download the 2019 root certificate that works for all AWS Regions and put the file in your trust store directory.

For information about downloading the root certificate, see [Using SSL/TLS to encrypt a connection to a DB cluster \(p. 1714\)](#).

2. Convert the certificate to .der format using the following command.

```
openssl x509 -outform der -in rds-ca-2019-root.pem -out rds-ca-2019-root.der
```

Replace the file name with the one that you downloaded.

3. Import the certificate into the key store using the following command.

```
keytool -import -alias rds-root -keystore clientkeystore -file rds-ca-2019-root.der
```

4. Confirm that the key store was updated successfully.

```
keytool -list -v -keystore clientkeystore.jks
```

Enter the key store password when you are prompted for it.

Your output should contain the following.

```
rds-root,date, trustedCertEntry,  
Certificate fingerprint (SHA1):  
D4:0D:DB:29:E3:75:0D:FF:A6:71:C3:14:0B:BF:5F:47:8D:1C:80:96  
# This fingerprint should match the output from the below command  
openssl x509 -fingerprint -in rds-ca-2019-root.pem -noout
```

Using SSL/TLS connections for different types of applications

The following provides information about using SSL/TLS connections for different types of applications:

- **psql**

The client is invoked from the command line by specifying options either as a connection string or as environment variables. For SSL/TLS connections, the relevant options are `sslmode` (environment variable `PGSSLMODE`), `sslrootcert` (environment variable `PGSSLROOTCERT`).

For the complete list of options, see [Parameter key words](#) in the PostgreSQL documentation. For the complete list of environment variables, see [Environment variables](#) in the PostgreSQL documentation.

- **pgAdmin**

This browser-based client is a more user-friendly interface for connecting to a PostgreSQL database.

For information about configuring connections, see the [pgAdmin documentation](#).

- **JDBC**

JDBC enables database connections with Java applications.

For general information about connecting to a PostgreSQL database with JDBC, see [Connecting to the database](#) in the PostgreSQL documentation. For information about connecting with SSL/TLS, see [Configuring the client](#) in the PostgreSQL documentation.

- **Python**

A popular Python library for connecting to PostgreSQL databases is `psycopg2`.

For information about using `psycopg2`, see the [psycopg2 documentation](#). For a short tutorial on how to connect to a PostgreSQL database, see [Psycopg2 tutorial](#). You can find information about the options the `connect` command accepts in [The psycopg2 module content](#).

Important

After you have determined that your database connections use SSL/TLS and have updated your application trust store, you can update your database to use the rds-ca-2019 certificates. For instructions, see step 3 in [Updating your CA certificate by modifying your DB instance \(p. 1716\)](#).

Migrating data to Amazon Aurora with PostgreSQL compatibility

You have several options for migrating data from your existing database to an Amazon Aurora PostgreSQL-Compatible Edition DB cluster. Your migration options also depend on the database that you are migrating from and the size of the data that you are migrating. Following are your options:

[Migrating an RDS for PostgreSQL DB instance using a snapshot \(p. 1284\)](#)

You can migrate data directly from an RDS for PostgreSQL DB snapshot to an Aurora PostgreSQL DB cluster.

[Migrating an RDS for PostgreSQL DB instance using an Aurora read replica \(p. 1288\)](#)

You can also migrate from an RDS for PostgreSQL DB instance by creating an Aurora PostgreSQL read replica of an RDS for PostgreSQL DB instance. When the replica lag between the RDS for PostgreSQL DB instance and the Aurora PostgreSQL read replica is zero, you can stop replication. At this point, you can make the Aurora read replica a standalone Aurora PostgreSQL DB cluster for reading and writing.

[Importing S3 data into Aurora PostgreSQL \(p. 1438\)](#)

You can migrate data by importing it from Amazon S3 into a table belonging to an Aurora PostgreSQL DB cluster.

[Migrating from a database that is not PostgreSQL-compatible](#)

You can use AWS Database Migration Service (AWS DMS) to migrate data from a database that is not PostgreSQL-compatible. For more information on AWS DMS, see [What is AWS Database Migration Service?](#) in the *AWS Database Migration Service User Guide*.

For a list of AWS Regions where Aurora is available, see [Amazon Aurora](#) in the *AWS General Reference*.

Important

If you plan to migrate an RDS for PostgreSQL DB instance to an Aurora PostgreSQL DB cluster in the near future, we strongly recommend that you turn off auto minor version upgrades for the DB instance early in the migration planning phase. Migration to Aurora PostgreSQL might be delayed if the RDS for PostgreSQL version isn't yet supported by Aurora PostgreSQL.

For information about Aurora PostgreSQL versions, see [Engine versions for Amazon Aurora PostgreSQL](#).

Migrating a snapshot of an RDS for PostgreSQL DB instance to an Aurora PostgreSQL DB cluster

To create an Aurora PostgreSQL DB cluster, you can migrate a DB snapshot of an RDS for PostgreSQL DB instance. The new Aurora PostgreSQL DB cluster is populated with the data from the original RDS for PostgreSQL DB instance. For information about creating a DB snapshot, see [Creating a DB snapshot](#).

In some cases, the DB snapshot might not be in the AWS Region where you want to locate your data. If so, use the Amazon RDS console to copy the DB snapshot to that AWS Region. For information about copying a DB snapshot, see [Copying a DB snapshot](#).

You can migrate RDS for PostgreSQL snapshots that are compatible with the Aurora PostgreSQL versions available in the given AWS Region. For example, you can migrate a snapshot from an RDS for PostgreSQL 11.1 DB instance to Aurora PostgreSQL version 11.4, 11.7, 11.8, or 11.9 in the US West (N. California) Region. You can migrate RDS for PostgreSQL 10.11 snapshot to Aurora PostgreSQL 10.11, 10.12, 10.13, and 10.14. In other words, the RDS for PostgreSQL snapshot must use the same or a lower minor version as the Aurora PostgreSQL.

You can also choose for your new Aurora PostgreSQL DB cluster to be encrypted at rest by using an AWS KMS key. This option is available only for unencrypted DB snapshots.

To migrate an RDS for PostgreSQL DB snapshot to an Aurora PostgreSQL DB cluster, you can use the AWS Management Console, the AWS CLI, or the RDS API. When you use the AWS Management Console, the console takes the actions necessary to create both the DB cluster and the primary instance.

Console

To migrate a PostgreSQL DB snapshot by using the RDS console

1. Sign in to the AWS Management Console and open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. Choose **Snapshots**.
3. On the **Snapshots** page, choose the RDS for PostgreSQL snapshot that you want to migrate into an Aurora PostgreSQL DB cluster.
4. Choose **Actions** then choose **Migrate snapshot**.
5. Set the following values on the **Migrate database** page:
 - **DB engine version:** Choose a DB engine version you want to use for the new migrated instance.
 - **DB instance identifier:** Enter a name for the DB cluster that is unique for your account in the AWS Region that you chose. This identifier is used in the endpoint addresses for the instances in your DB cluster. You might choose to add some intelligence to the name, such as including the AWS Region and DB engine that you chose, for example **aurora-cluster1**.

The DB instance identifier has the following constraints:

- It must contain 1–63 alphanumeric characters or hyphens.
- Its first character must be a letter.
- It can't end with a hyphen or contain two consecutive hyphens.
- It must be unique for all DB instances per AWS account, per AWS Region.
- **DB instance class:** Choose a DB instance class that has the required storage and capacity for your database, for example **db.r6g.large**. Aurora cluster volumes automatically grow as the amount

of data in your database increases. So you only need to choose a DB instance class that meets your current storage requirements. For more information, see [Overview of Aurora storage \(p. 64\)](#).

- **Virtual private cloud (VPC):** If you have an existing VPC, then you can use that VPC with your Aurora PostgreSQL DB cluster by choosing your VPC identifier, for example `vpc-a464d1c1`. For information on using an existing VPC, see [How to create a VPC for use with Amazon Aurora \(p. 1793\)](#).

Otherwise, you can choose to have Amazon RDS create a VPC for you by choosing [Create new VPC](#).

- **Subnet group:** If you have an existing subnet group, then you can use that subnet group with your Aurora PostgreSQL DB cluster by choosing your subnet group identifier, for example `gs-subnet-group1`.
- **Public access:** Choose **No** to specify that instances in your DB cluster can only be accessed by resources inside of your VPC. Choose **Yes** to specify that instances in your DB cluster can be accessed by resources on the public network.

Note

Your production DB cluster might not need to be in a public subnet, because only your application servers require access to your DB cluster. If your DB cluster doesn't need to be in a public subnet, set **Public access** to **No**.

- **VPC security group:** Choose a VPC security group to allow access to your database.
- **Availability Zone:** Choose the Availability Zone to host the primary instance for your Aurora PostgreSQL DB cluster. To have Amazon RDS choose an Availability Zone for you, choose **No preference**.
- **Database port:** Enter the default port to be used when connecting to instances in the Aurora PostgreSQL DB cluster. The default is 5432.

Note

You might be behind a corporate firewall that doesn't allow access to default ports such as the PostgreSQL default port, 5432. In this case, provide a port value that your corporate firewall allows. Remember that port value later when you connect to the Aurora PostgreSQL DB cluster.

- **Enable Encryption:** Choose **Enable Encryption** for your new Aurora PostgreSQL DB cluster to be encrypted at rest. Also choose a KMS key as the **AWS KMS key** value.
- **Auto minor version upgrade:** Choose **Enable auto minor version upgrade** to enable your Aurora PostgreSQL DB cluster to receive minor PostgreSQL DB engine version upgrades automatically when they become available.

The **Auto minor version upgrade** option only applies to upgrades to PostgreSQL minor engine versions for your Aurora PostgreSQL DB cluster. It doesn't apply to regular patches applied to maintain system stability.

6. Choose **Migrate** to migrate your DB snapshot.
7. Choose **Databases** to see the new DB cluster. Choose the new DB cluster to monitor the progress of the migration. On the **Connectivity & security** tab, you can find the cluster endpoint to use for connecting to the primary writer instance of the DB cluster. For more information on connecting to an Aurora PostgreSQL DB cluster, see [Connecting to an Amazon Aurora DB cluster \(p. 281\)](#).

AWS CLI

Using the AWS CLI to migrate an RDS for PostgreSQL DB snapshot to an Aurora PostgreSQL involves two separate AWS CLI commands. First, you use the `restore-db-cluster-from-snapshot` AWS CLI command create a new Aurora PostgreSQL DB cluster. You then use the `create-db-instance` command to create the primary DB instance in the new cluster to complete the migration. The following procedure creates an Aurora PostgreSQL DB cluster with primary DB instance that has the same configuration as the DB instance used to create the snapshot.

To migrate an RDS for PostgreSQL DB snapshot to an Aurora PostgreSQL DB cluster

1. Use the [describe-db-snapshots](#) command to obtain information about the DB snapshot you want to migrate. You can specify either the `--db-instance-identifier` parameter or the `--db-snapshot-identifier` in the command. If you don't specify one of these parameters, you get all snapshots.

```
aws rds describe-db-snapshots --db-instance-identifier <your-db-instance-name>
```

2. The command returns all configuration details for any snapshots created from the DB instance specified. In the output, find the snapshot that you want to migrate and locate its Amazon Resource Name (ARN). To learn more about Amazon RDS ARNs, see [Amazon Relational Database Service \(Amazon RDS\)](#). An ARN looks similar to the output following.

```
"DBSnapshotArn": "arn:aws:rds:<aws-region>:111122223333:snapshot:<snapshot_name>"
```

Also in the output you can find configuration details for the RDS for PostgreSQL DB instance, such as the engine version, allocated storage, whether or not the DB instance is encrypted, and so on.

3. Use the [restore-db-cluster-from-snapshot](#) command to start the migration. Specify the following parameters:
 - `--db-cluster-identifier` – The name that you want to give to the Aurora PostgreSQL DB cluster. This Aurora DB cluster is the target for your DB snapshot migration.
 - `--snapshot-identifier` – The Amazon Resource Name (ARN) of the DB snapshot to migrate.
 - `--engine` – Specify `aurora-postgresql` for the Aurora DB cluster engine.
 - `--kms-key-id` – This optional parameter lets you create an encrypted Aurora PostgreSQL DB cluster from an unencrypted DB snapshot. It also lets you choose a different encryption key for the DB cluster than the key used for the DB snapshot.

Note

You can't create an unencrypted Aurora PostgreSQL DB cluster from an encrypted DB snapshot.

Without the `--kms-key-id` parameter specified as shown following, the [restore-db-cluster-from-snapshot](#) AWS CLI command creates an empty Aurora PostgreSQL DB cluster that's either encrypted using the same key as the DB snapshot or is unencrypted if the source DB snapshot isn't encrypted.

For Linux, macOS, or Unix:

```
aws rds restore-db-cluster-from-snapshot \
  --db-cluster-identifier cluster-name \
  --snapshot-identifier arn:aws:rds:<aws-region>:111122223333:snapshot:<your-snapshot-name> \
  --engine aurora-postgresql
```

For Windows:

```
aws rds restore-db-cluster-from-snapshot ^
  --db-cluster-identifier new_cluster ^
  --snapshot-identifier arn:aws:rds:<aws-region>:111122223333:snapshot:<your-snapshot-name> ^
  --engine aurora-postgresql
```

4. The command returns details about the Aurora PostgreSQL DB cluster that's being created for the migration. You can check the status of the Aurora PostgreSQL DB cluster by using the [describe-db-clusters](#) AWS CLI command.

```
aws rds describe-db-clusters --db-cluster-identifier cluster-name
```

5. When the DB cluster becomes "available", you use [create-db-instance](#) command to populate the Aurora PostgreSQL DB cluster with the DB instance based on your Amazon RDS DB snapshot. Specify the following parameters:

- `--db-cluster-identifier` – The name of the new Aurora PostgreSQL DB cluster that you created in the previous step.
- `--db-instance-identifier` – The name you want to give to the DB instance. This instance becomes the primary node in your Aurora PostgreSQL DB cluster.
- `--db-instance-class` – Specify the DB instance class to use. Choose from among the DB instance classes supported by the Aurora PostgreSQL version to which you're migrating. For more information, see [DB instance class types \(p. 54\)](#) and [Supported DB engines for DB instance classes \(p. 54\)](#).
- `--engine` – Specify `aurora-postgresql` for the DB instance.

You can also create the DB instance with a different configuration than the source DB snapshot, by passing in the appropriate options in the `create-db-instance` AWS CLI command. For more information, see the [create-db-instance](#) command.

For Linux, macOS, or Unix:

```
aws rds create-db-instance \  
  --db-cluster-identifier cluster-name \  
  --db-instance-identifier --db-instance-class db.instance.class \  
  --engine aurora-postgresql
```

For Windows:

```
aws rds create-db-instance ^  
  --db-cluster-identifier cluster-name ^  
  --db-instance-identifier --db-instance-class db.instance.class ^  
  --engine aurora-postgresql
```

When the migration process completes, the Aurora PostgreSQL cluster has a populated primary DB instance.

Migrating data from an RDS for PostgreSQL DB instance to an Aurora PostgreSQL DB cluster using an Aurora read replica

You can use an RDS for PostgreSQL DB instance as the basis for a new Aurora PostgreSQL DB cluster by using an Aurora read replica for the migration process. The Aurora read replica option is available only for migrating within the same AWS Region and account. This option is available only to an RDS for PostgreSQL DB instance for which the Region offers a compatible version of Aurora PostgreSQL. In this case, *compatible* means that the Aurora PostgreSQL version is the same as the RDS for PostgreSQL version, or that it is a higher minor version in the same major version family.

For example, to use this technique to migrate an RDS for PostgreSQL 11.14 DB instance, the Region must offer Aurora PostgreSQL version 11.14 or a higher minor version in the PostgreSQL version 11 family.

Topics

- [Overview of migrating data by using an Aurora read replica \(p. 1288\)](#)
- [Preparing to migrate data by using an Aurora read replica \(p. 1288\)](#)
- [Creating an Aurora read replica \(p. 1289\)](#)
- [Promoting an Aurora read replica \(p. 1294\)](#)

Overview of migrating data by using an Aurora read replica

Migrating from an RDS for PostgreSQL DB instance to an Aurora PostgreSQL DB cluster is a multistep procedure. First, you create an Aurora read replica of your source RDS for PostgreSQL DB instance. That starts a replication process from your RDS for PostgreSQL DB instance to a special-purpose DB cluster known as a *Replica cluster*. The Replica cluster consists solely of an Aurora read replica (a reader instance).

Once the Replica cluster exists, you monitor the lag between it and the source RDS for PostgreSQL DB instance. When the replica lag is zero (0), you can promote the Replica cluster. Replication stops, the Replica cluster is promoted to a standalone Aurora DB cluster, and the reader is promoted to writer instance for the cluster. You can then add instances to the Aurora PostgreSQL DB cluster to size your Aurora PostgreSQL DB cluster for your use case. You can also delete the RDS for PostgreSQL DB instance if you have no further need of it.

Note

It can take several hours per terabyte (TiB) of data for the migration to complete.

You can't create an Aurora read replica if your RDS for PostgreSQL DB instance already has an Aurora read replica or if it has a cross-Region read replica.

Preparing to migrate data by using an Aurora read replica

During the migration process using Aurora read replica, updates made to the source RDS for PostgreSQL DB instance are asynchronously replicated to the Aurora read replica of the Replica cluster. The process uses PostgreSQL's native streaming replication functionality which stores write-ahead logs (WAL) segments on the source instance. Before starting this migration process, make sure that your instance has sufficient storage capacity by checking values for the metrics listed in the table.

Metric	Description
FreeStorageSpace	The available storage space. Units: Bytes

Metric	Description
OldestReplicationSlotLag	The size of the lag for WAL data in the replica that is lagging the most. Units: Megabytes
RDSToAuroraPostgreSQLReplicaLag	The amount of time in seconds that an Aurora PostgreSQL DB cluster lags behind the source RDS DB instance.
TransactionLogsDiskUsage	The disk space used by the transaction logs. Units: Megabytes

For more information about monitoring your RDS instance, see [Monitoring](#) in the *Amazon RDS User Guide*.

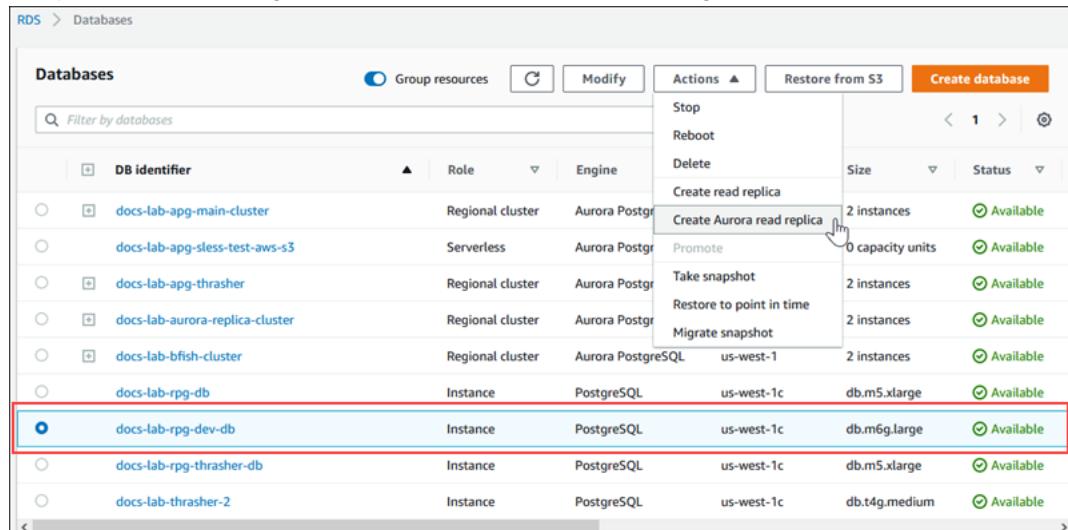
Creating an Aurora read replica

You can create an Aurora read replica for an RDS for PostgreSQL DB instance by using the console or the AWS CLI. The option to create an Aurora read replica is available only if the AWS Region offers a compatible Aurora PostgreSQL version. That is, it's available only if there's an Aurora PostgreSQL version that is the same as the RDS for PostgreSQL version or a higher minor version in the same major version family.

Console

To create an Aurora read replica from a source PostgreSQL DB instance

1. Sign in to the AWS Management Console and open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. In the navigation pane, choose **Databases**.
3. Choose the RDS for PostgreSQL DB instance that you want to use as the source for your Aurora read replica. For **Actions**, choose **Create Aurora read replica**. If this choice doesn't display, it means that a compatible Aurora PostgreSQL version isn't available in the Region.



4. On the Create Aurora read replica settings page, you configure the properties for the Aurora PostgreSQL DB cluster as shown in the following table. The Replica DB cluster is created from a

snapshot of the source DB instance using the same 'master' user name and password as the source, so you can't change these at this time.

Option	Description
DB instance class	Choose a DB instance class that meets the processing and memory requirements primary instance in the DB cluster. For more information, see Aurora DB instance classes (p. 54) .
Multi-AZ deployment	Not available during the migration
DB instance identifier	Enter the name that you want to give to the DB instance. This identifier is used in the endpoint address for the primary instance of the new DB cluster. The DB instance identifier has the following constraints: <ul style="list-style-type: none">• It must contain 1–63 alphanumeric characters or hyphens.• Its first character must be a letter.• It can't end with a hyphen or contain two consecutive hyphens.• It must be unique for all DB instances for each AWS account, for each AWS Region.
Virtual Private Cloud (VPC)	Choose the VPC to host the DB cluster. Choose Create new VPC to have Amazon RDS create a VPC for you. For more information, see DB cluster prerequisites (p. 125) .
Subnet group	Choose the DB subnet group to use for the DB cluster. Choose Create new DB Subnet Group to have Amazon RDS create a DB subnet group for you. For more information, see DB cluster prerequisites (p. 125) .
Public accessibility	Choose Yes to give the DB cluster a public IP address; otherwise, choose No . The instances in your DB cluster can be a mix of both public and private DB instances. For more information about hiding instances from public access, see Hiding a DB instance in a VPC from the internet (p. 1789) .
Availability zone	Determine if you want to specify a particular Availability Zone. For more information about Availability Zones, see Regions and Availability Zones (p. 11) .
VPC security groups	Choose one or more VPC security groups to secure network access to the DB cluster. Choose Create new VPC security group to have Amazon RDS create a VPC security group for you. For more information, see DB cluster prerequisites (p. 125) .

Option	Description
Database port	Specify the port for applications and utilities to use to access the database. Aurora PostgreSQL DB clusters default to the default PostgreSQL port, 5432. Firewalls at some companies block connections to this port. If your company firewall blocks the default port, choose another port for the new DB cluster.
DB parameter group	Choose a DB parameter group for the Aurora PostgreSQL DB cluster. Aurora has a default DB parameter group you can use, or you can create your own DB parameter group. For more information about DB parameter groups, see Working with DB parameter groups and DB cluster parameter groups (p. 339) .
DB cluster parameter group	Choose a DB cluster parameter group for the Aurora PostgreSQL DB cluster. Aurora has a default DB cluster parameter group you can use, or you can create your own DB cluster parameter group. For more information about DB cluster parameter groups, see Working with DB parameter groups and DB cluster parameter groups (p. 339) .
Encryption	Choose Enable encryption for your new Aurora DB cluster to be encrypted at rest. If you choose Enable encryption , also choose a KMS key as the AWS KMS key value.
Priority	Choose a failover priority for the DB cluster. If you don't choose a value, the default is tier-1 . This priority determines the order in which Aurora Replicas are promoted when recovering from a primary instance failure. For more information, see Fault tolerance for an Aurora DB cluster (p. 69) .
Backup retention period	Choose the length of time, 1–35 days, for Aurora to retain backup copies of the database. Backup copies can be used for point-in-time restores (PITR) of your database down to the second.
Enhanced monitoring	Choose Enable enhanced monitoring to enable gathering metrics in real time for the operating system that your DB cluster runs on. For more information, see Monitoring OS metrics with Enhanced Monitoring (p. 626) .
Monitoring Role	Only available if you chose Enable enhanced monitoring . The AWS Identity and Access Management (IAM) role to use for Enhanced Monitoring. For more information, see Setting up and enabling Enhanced Monitoring (p. 627) .
Granularity	Only available if you chose Enable enhanced monitoring . Set the interval, in seconds, between when metrics are collected for your DB cluster.

Option	Description
Auto minor version upgrade	<p>Choose Yes to enable your Aurora PostgreSQL DB cluster to receive minor PostgreSQL DB engine version upgrades automatically when they become available.</p> <p>The Auto minor version upgrade option only applies to upgrades to PostgreSQL minor engine versions for your Aurora PostgreSQL DB cluster. It doesn't apply to regular patches applied to maintain system stability.</p>
Maintenance window	Choose the weekly time range during which system maintenance can occur.

5. Choose **Create read replica**.

AWS CLI

To create an Aurora read replica from a source RDS for PostgreSQL DB instance, use the [create-db-cluster](#) and [create-db-instance](#) AWS CLI commands to create a new Aurora PostgreSQL DB cluster. When you call the **create-db-cluster** command, include the `--replication-source-identifier` parameter to identify the Amazon Resource Name (ARN) for the source RDS for PostgreSQL DB instance. For more information about Amazon RDS ARNs, see [Amazon Relational Database Service \(Amazon RDS\)](#) in the [AWS General Reference](#).

Don't specify the master user name, master password, or database name. The Aurora read replica uses the same master user name, master password, and database name as the source RDS for PostgreSQL DB instance.

For Linux, macOS, or Unix:

```
aws rds create-db-cluster --db-cluster-identifier sample-replica-cluster --engine aurora-postgresql \
    --db-subnet-group-name mysubnetgroup --vpc-security-group-ids sg-c7e5b0d2 \
    --replication-source-identifier arn:aws:rds:us-west-2:123456789012:db:master-postgresql-instance
```

For Windows:

```
aws rds create-db-cluster --db-cluster-identifier sample-replica-cluster --engine aurora-postgresql ^
    --db-subnet-group-name mysubnetgroup --vpc-security-group-ids sg-c7e5b0d2 ^
    --replication-source-identifier arn:aws:rds:us-west-2:123456789012:db:master-postgresql-instance
```

If you use the console to create an Aurora read replica, then RDS automatically creates the primary instance for your DB cluster Aurora Read Replica. If you use the CLI to create an Aurora read replica, you must explicitly create the primary instance for your DB cluster. The primary instance is the first instance that is created in a DB cluster.

You can create a primary instance for your DB cluster by using the [create-db-instance](#) CLI command with the following parameters:

- **--db-cluster-identifier**
The name of your DB cluster.
- **--db-instance-class**
The name of the DB instance class to use for your primary instance.

- **--db-instance-identifier**
The name of your primary instance.
- **--engine aurora-postgresql**
The database engine to use.

In the following example, you create a primary instance named *myreadreplicainstance* for the DB cluster named *myreadreplicacluster*. You do this using the DB instance class specified in *myinstanceclass*.

Example

For Linux, macOS, or Unix:

```
aws rds create-db-instance \
  --db-cluster-identifier myreadreplicacluster \
  --db-instance-class myinstanceclass \
  --db-instance-identifier myreadreplicainstance \
  --engine aurora-postgresql
```

For Windows:

```
aws rds create-db-instance ^
  --db-cluster-identifier myreadreplicacluster ^
  --db-instance-class myinstanceclass ^
  --db-instance-identifier myreadreplicainstance ^
  --engine aurora-postgresql
```

RDS API

To create an Aurora read replica from a source RDS for PostgreSQL DB instance, use the RDS API operations [CreateDBCluster](#) and [CreateDBInstance](#) to create a new Aurora DB cluster and primary instance. Don't specify the master user name, master password, or database name. The Aurora read replica uses the same master user name, master password, and database name as the source RDS for PostgreSQL DB instance.

You can create a new Aurora DB cluster for an Aurora read replica from a source RDS for PostgreSQL DB instance. To do so, use the RDS API operation [CreateDBCluster](#) with the following parameters:

- **DBClusterIdentifier**
The name of the DB cluster to create.
- **DBSubnetGroupName**
The name of the DB subnet group to associate with this DB cluster.
- **Engine=aurora-postgresql**
The name of the engine to use.
- **ReplicationSourceIdentifier**

The Amazon Resource Name (ARN) for the source PostgreSQL DB instance. For more information about Amazon RDS ARNs, see [Amazon Relational Database Service \(Amazon RDS\)](#) in the *Amazon Web Services General Reference*.

- **VpcSecurityGroupIds**
The list of Amazon EC2 VPC security groups to associate with this DB cluster.

See an example with the RDS API operation [CreateDBCluster](#).

If you use the console to create an Aurora read replica, then Amazon RDS automatically creates the primary instance for your DB cluster Aurora Read Replica. If you use the CLI to create an Aurora read replica, you must explicitly create the primary instance for your DB cluster. The primary instance is the first instance that is created in a DB cluster.

You can create a primary instance for your DB cluster by using the RDS API operation [CreateDBInstance](#) with the following parameters:

- **DBClusterIdentifier**
The name of your DB cluster.
- **DBInstanceClass**
The name of the DB instance class to use for your primary instance.
- **DBInstanceIdentifier**
The name of your primary instance.
- **Engine=aurora-postgresql**
The name of the engine to use.

See an example with the RDS API operation [CreateDBInstance](#).

Promoting an Aurora read replica

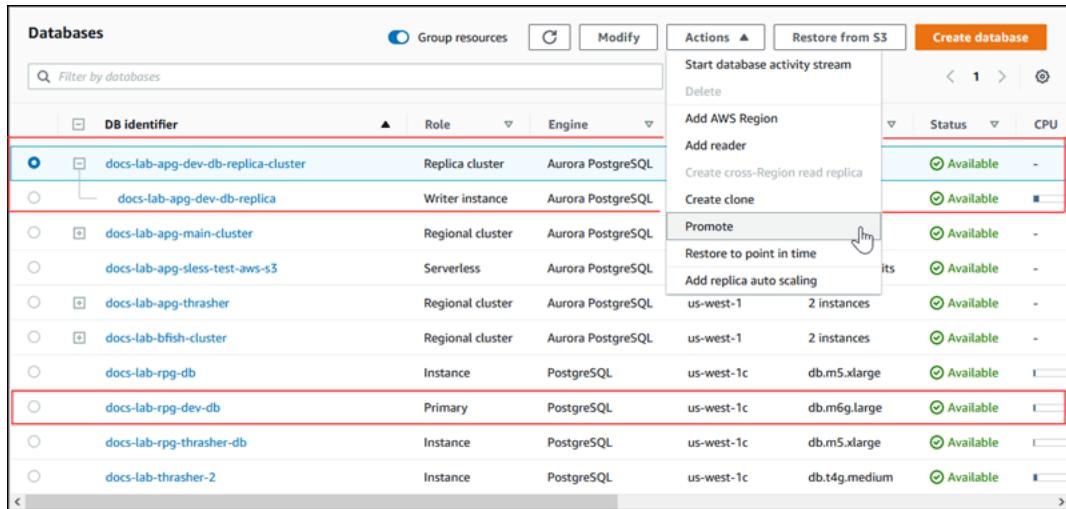
The migration to Aurora PostgreSQL isn't complete until you promote the Replica cluster, so don't delete the RDS for PostgreSQL source DB instance just yet.

Before promoting the Replica cluster, make sure that the RDS for PostgreSQL DB instance doesn't have any in-process transactions or other activity writing to the database. When the replica lag on the Aurora read replica reaches zero (0), you can promote the Replica cluster. For more information about monitoring replica lag, see [Monitoring Aurora PostgreSQL replication \(p. 1432\)](#) and [Instance-level metrics for Amazon Aurora \(p. 639\)](#).

Console

To promote an Aurora read replica to an Aurora DB cluster

1. Sign in to the AWS Management Console and open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. In the navigation pane, choose **Databases**.
3. Choose the Replica cluster.



- For **Actions**, choose **Promote**. This may take a few minutes.

When the process completes, the Aurora Replica cluster is a Regional Aurora PostgreSQL DB cluster, with a Writer instance containing the data from the RDS for PostgreSQL DB instance.

AWS CLI

To promote an Aurora read replica to a stand-alone DB cluster, use the [promote-read-replica-db-cluster](#) AWS CLI command.

Example

For Linux, macOS, or Unix:

```
aws rds promote-read-replica-db-cluster \
--db-cluster-identifier myreadreplicacluster
```

For Windows:

```
aws rds promote-read-replica-db-cluster ^
--db-cluster-identifier myreadreplicacluster
```

RDS API

To promote an Aurora read replica to a stand-alone DB cluster, use the RDS API operation [PromoteReadReplicaDBCluster](#).

After you promote the Replica cluster, you can confirm that the promotion has completed by checking the event log, as follows.

To confirm that the Aurora Replica cluster was promoted

- Sign in to the AWS Management Console and open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
- In the navigation pane, choose **Events**.
- On the **Events** page, find the name of your cluster in the **Source** list. Each event has a source, type, time, and message. You can see all events that have occurred in your AWS Region for your account. A successful promotion generates the following message.

Promoted Read Replica cluster to a stand-alone database cluster.

After promotion is complete, the source RDS for PostgreSQL DB instance and the Aurora PostgreSQL DB cluster are unlinked. You can direct your client applications to the endpoint for the Aurora read replica. For more information on the Aurora endpoints, see [Amazon Aurora connection management \(p. 32\)](#). At this point, you can safely delete the DB instance.

Working with Babelfish for Aurora PostgreSQL

Babelfish for Aurora PostgreSQL extends your Amazon Aurora PostgreSQL-Compatible Edition database with the ability to accept database connections from Microsoft SQL Server clients. Doing this allows applications originally built for SQL Server to work directly with Aurora PostgreSQL with few code changes compared to a traditional migration and without changing database drivers. For more information about migrating, see [Using Babelfish to migrate to PostgreSQL \(p. 1301\)](#).

Babelfish provides an additional endpoint for an Aurora PostgreSQL database cluster that allows it to understand the SQL Server wire-level protocol and commonly used SQL Server statements. This approach allows client applications that use the Tabular Data Stream (TDS) wire protocol to connect natively to the TDS listener port on Aurora PostgreSQL. Babelfish supports TDS versions 7.1 and higher. For more information on the SQL Server wire-level protocol, see [\[MS-TDS\]: Tabular Data Stream Protocol](#) on the Microsoft website.

You can access your data simultaneously using a Babelfish TDS connection from one application and a native PostgreSQL connection from another application. You can customize the ports used for each client connection when you create the cluster, or later in your Aurora PostgreSQL parameter group. For more information about the parameters that control Babelfish, see [Configuring a database for Babelfish \(p. 1345\)](#).

By default, to use the following dialects use the following ports:

- SQL Server dialect, clients connect to port 1433.
- PostgreSQL dialect, clients connect to port 5432.

Babelfish runs the Transact-SQL (T-SQL) language with the exceptions documented in [Differences between Aurora PostgreSQL with Babelfish and SQL Server \(p. 1322\)](#).

Following, you can find information about how to work with Babelfish.

Topics

- [Babelfish architecture \(p. 1297\)](#)
- [Using Babelfish to migrate to PostgreSQL \(p. 1301\)](#)
- [Creating an Aurora PostgreSQL cluster with Babelfish \(p. 1303\)](#)
- [Connecting to a DB cluster with Babelfish turned on \(p. 1310\)](#)
- [Querying a database for object information \(p. 1318\)](#)
- [Querying Babelfish to find Babelfish details \(p. 1319\)](#)
- [Differences between Aurora PostgreSQL with Babelfish and SQL Server \(p. 1322\)](#)
- [Using Aurora PostgreSQL extensions with Babelfish \(p. 1335\)](#)
- [Managing Babelfish error handling \(p. 1340\)](#)
- [Configuring a database for Babelfish \(p. 1345\)](#)
- [Babelfish collation support \(p. 1349\)](#)
- [Troubleshooting for Babelfish \(p. 1355\)](#)
- [Turning off Babelfish \(p. 1357\)](#)
- [Babelfish versions \(p. 1358\)](#)

Babelfish architecture

When you create an Aurora PostgreSQL cluster with Babelfish turned on, Aurora provisions the cluster with a PostgreSQL database named `babelfish_db`. This database is where all migrated SQL Server objects and structures reside.

Note

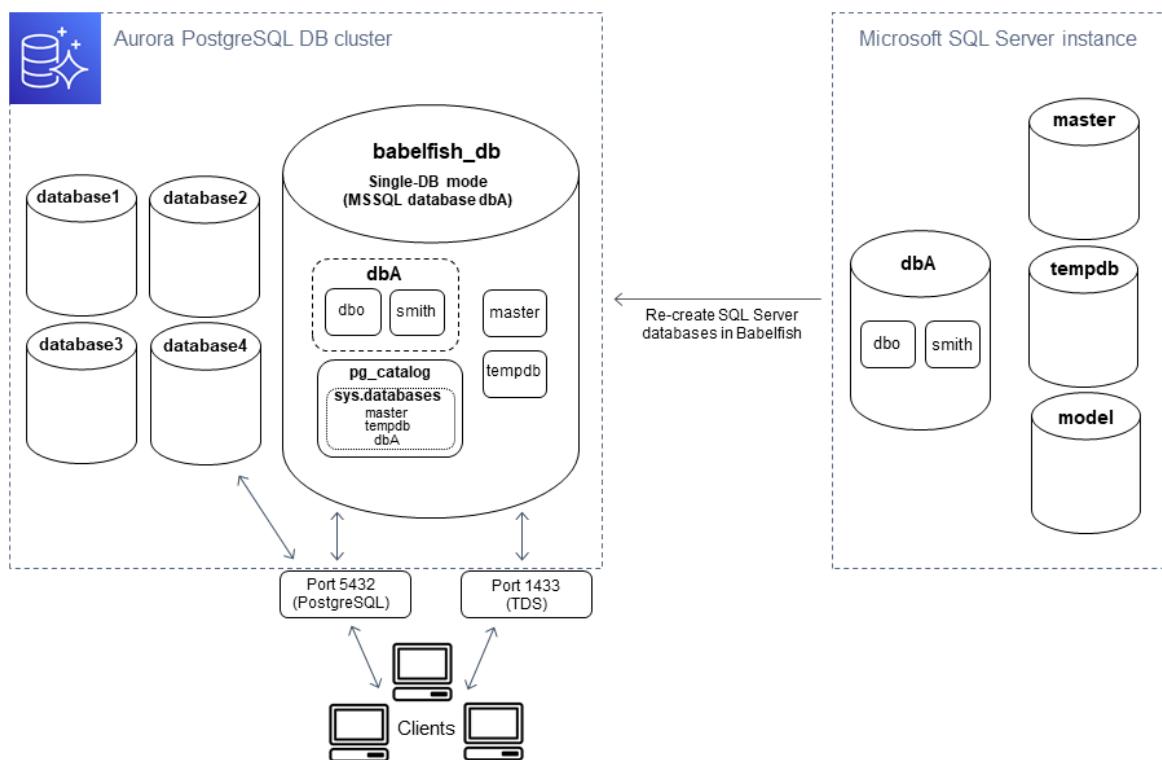
In an Aurora PostgreSQL cluster, the `babelfish_db` database name is reserved for Babelfish. Creating your own "babelfish_db" database on a Babelfish for Aurora PostgreSQL prevents Aurora from successfully provisioning Babelfish.

When you connect to the TDS port, the session is placed in the `babelfish_db` database. From T-SQL, the structure looks similar to being connected to a SQL Server instance. You can see the `master` and `tempdb` databases and the `sys.databases` catalog. You can create additional user databases and switch between databases with the `USE` statement. When you create a SQL Server user database, it's flattened into the `babelfish_db` PostgreSQL database. Your database retains cross-database syntax and semantics equal to or similar to those provided by SQL Server.

Using Babelfish with a single database or multiple databases

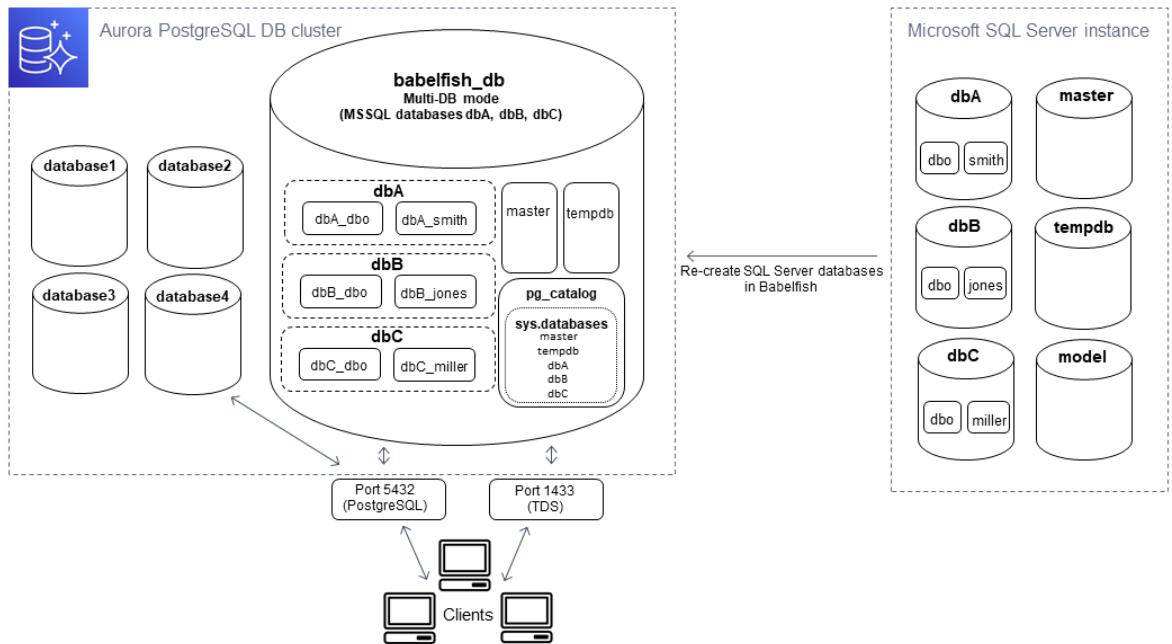
When you create an Aurora PostgreSQL cluster to use with Babelfish, you choose between using a single SQL Server database on its own or multiple SQL Server databases together. Your choice affects how the names of SQL Server schemas inside the `babelfish_db` database appear from Aurora PostgreSQL. The migration mode is stored in the `migration_mode` parameter. You can't change this parameter after creating your cluster.

In single-database mode, the schema names of the user database in the `babelfish_db` database remain the same as in SQL Server. If you choose to move a single database, schemas are recreated inside of the database and can be referenced with the same name used with SQL Server. For example, the `dbo` and `smith` schemas reside inside the `dbA` database.



When connecting through TDS, you can run `USE dbA` to see schemas `dbo` and `smith` from T-SQL, as you would in SQL Server. The unchanged schema names are also visible from PostgreSQL.

In multiple-database mode, the schema names of user databases become `dbname_schemaname` when seen from PostgreSQL. The schema names remain the same when seen from T-SQL.



When connecting through TDS, you can run `USE dbA`, to see schemas `dbo` and `smith` from T-SQL, as you would in SQL Server. The mapped schema names, such as `dbA_dbo` and `dbA_smith`, are visible from PostgreSQL.

Each database still contains your schemas. The name of each database is prepended to the name of the SQL Server schema, using an underscore as a delimiter, for example:

- dbA contains dbA_dbo and dbA_smith.
- dbB contains dbB_dbo and dbB_jones.
- dbC contains dbC_dbo and dbC_miller.

Inside the `babelfish_db` database, the T-SQL user still needs to run `USE dbname` to change database context, so the look and feel remains similar to SQL Server.

Choosing a migration mode

Each migration mode has advantages and disadvantages. Choose your migration mode based on the number of user databases you have, and your migration plans. After you create a cluster for use with Babelfish, you can't change the migration mode. When choosing a migration mode, consider the requirements of your user databases and clients.

When you create a cluster for use with Babelfish, Aurora PostgreSQL creates the system databases, `master` and `tempdb`. If you created or modified objects in the system databases (`master` or `tempdb`), make sure to recreate those objects in your new cluster. Unlike SQL Server, Babelfish doesn't reinitialize `tempdb` after a cluster reboot.

Use single database migration mode in the following cases:

- If you are migrating a single SQL Server database. In single database mode, migrated schema names are identical to the original SQL Server schema names. When you migrate your application, you make fewer changes to your SQL code.
- If your end goal is a complete migration to native Aurora PostgreSQL. Before migrating, consolidate your schemas into a single schema (`dbo`) and then migrate into a single cluster to lessen required changes.

Use multiple database migration mode in the following cases:

- If you are trying out Babelfish and you aren't sure of your future needs.
- If multiple user databases need to be migrated together, and the end goal isn't to perform a fully native PostgreSQL migration.
- If you might be migrating multiple databases in the future.

Using Babelfish to migrate to PostgreSQL

You can use Babelfish for Aurora PostgreSQL to ease migration from a SQL Server database to an Amazon Aurora PostgreSQL DB cluster. Before migrating, review [Using Babelfish with a single database or multiple databases \(p. 1298\)](#).

The following high-level overview lists the steps required to make your SQL Server application work with Babelfish:

1. Create a new Aurora PostgreSQL DB cluster with Babelfish turned on, providing support for SQL Server T-SQL syntax and features. For details, see [Creating an Aurora PostgreSQL cluster with Babelfish \(p. 1303\)](#).
2. To connect to the new database, use a native SQL Server tool such as `sqlcmd`. For details, see [Using a SQL Server client to connect to your DB cluster \(p. 1313\)](#).
3. Export the data definition language (DDL) for your SQL Server databases that you want to migrate. The DDL is SQL code that describes database objects that contain user data (such as tables, indexes, and views) and user-written database code (such as stored procedures, user-defined functions, and triggers).

You can use SQL Server Management Studio (SSMS) to export the DDL. After connecting to your existing SQL Server instance, complete the following steps:

- a. Open the context menu (right-click) for a database name.
 - b. Choose **Tasks, Generate Scripts** from the context menu.
 - c. On the **Choose Objects** page, select the entire database or specific objects.
 - d. On the **Set Scripting Options** page, choose **Advanced** and make sure that you turn on triggers, logins, owners, and permissions. These are turned off by default in SSMS.
 - e. Save the script.
4. Export the data manipulation language (DML) for your SQL Server databases that you want to migrate. The DML is SQL code that inserts rows into the tables in your database.

You can use SQL Server Management Studio (SSMS) to export the DML. After connecting to your existing SQL Server instance, complete the following steps:

- a. Open the context (right-click) menu for a database name.
 - b. Choose **Tasks, Generate Scripts** from the context menu.
 - c. On the **Choose Objects** page, select the entire database or specific objects.
 - d. On the **Set Scripting Options** page, choose **Advanced** and for **Types of data to script**, choose **Data only**.
 - e. Save the script.
5. Run an assessment tool. For example, you can run the [Babelfish Compass tool](#). You run this tool on the DDL to determine to what extent the T-SQL code is supported by Babelfish. Identify T-SQL code that might require modifications before running on Babelfish.

Note

Because Babelfish Compass is an open-source tool, report any issues through GitHub. Don't report issues with Babelfish Compass to AWS Support.

You can also use the AWS Schema Conversion Tool to help with your migration. The AWS Schema Conversion Tool supports Babelfish as a virtual target. To learn more, see [Using virtual targets](#) in the [AWS Schema Conversion Tool User Guide](#).

6. Run the DDL on your new Babelfish server to recreate your schemas on Babelfish using SSMS or `sqlcmd`. Make code adjustments as needed. This process might require multiple iterations.
7. Run the DML on your new Babelfish server to insert rows into the tables in your database.
8. Reconfigure your client application to connect to the Babelfish endpoint instead of SQL Server. For details, see [Connecting to a DB cluster with Babelfish turned on \(p. 1310\)](#).

9. Modify your application where necessary and retest. For more information, see [Differences between Aurora PostgreSQL with Babelfish and SQL Server \(p. 1322\)](#).

10 When you're satisfied with your application test results, start using your Babelfish database for production.

When you're ready, stop the original database and redirect live client applications to use the Babelfish TDS port.

11 (Optional) Capture client-side SQL queries, and run these queries through an assessment tool (such as Babelfish Compass). A reverse-engineered schema only converts server-side SQL code. For applications with complex client-side SQL queries, we recommend that you also analyze these for Babelfish compatibility. If the analysis indicates that the client-side SQL statements contain unsupported SQL features, review the SQL aspects in the client application and make modifications if necessary.

Creating an Aurora PostgreSQL cluster with Babelfish

You can use Babelfish on Aurora PostgreSQL. Babelfish is currently supported on Aurora PostgreSQL version 13.4 and higher.

You can use the AWS Management Console or the AWS CLI to create an Aurora PostgreSQL cluster with Babelfish.

Note

In an Aurora PostgreSQL cluster, the `babelfish_db` database name is reserved for Babelfish. Creating your own "babelfish_db" database on a Babelfish for Aurora PostgreSQL prevents Aurora from successfully provisioning Babelfish.

Console

To create a cluster with Babelfish running with the AWS Management Console

1. Open the Amazon RDS console at <https://console.aws.amazon.com/rds/>, and choose **Create database**.

The screenshot shows the Amazon RDS console interface. On the left, there's a sidebar with 'Resources' and various metrics like DB Instances (0/40), Allocated storage (0 TB/10 TB), Click here to increase DB instances limit, DB Clusters (0/40), Reserved instances (0/40), Snapshots (0), Manual (0/50), Automated (0), Recent events (0), and Event subscriptions (0/20). Below this is a 'Create database' section with a 'Create database' button. On the right, there's a 'Recommended for you' sidebar with links to 'Build RDS Operational Tasks', 'Amazon RDS Backup and Restore using AWS Backup', 'Time-Series Tables in PostgreSQL', 'Implementing Cross-Region DR', and 'Additional information'.

2. For **Choose a database creation method**, do one of the following:
 - To specify detailed engine options, choose **Standard create**.
 - To use preconfigured options that support best practices for an Aurora cluster, choose **Easy create**.
3. For **Engine type**, choose **Amazon Aurora**.
4. For **Edition**, choose **Amazon Aurora PostgreSQL**.
5. Choose **Show filters**, and then choose **Show versions that support the Babelfish for PostgreSQL feature** to list the engine types that support Babelfish. Babelfish is currently supported on Aurora PostgreSQL 13.4 and higher.
6. For **Available versions**, choose an Aurora PostgreSQL version.

Engine version [Info](#)
View the engine versions that support the following database features.

▼ Hide filters

Show versions that support the global database feature
Allows a single Amazon Aurora database to span multiple AWS Regions.

Show versions that support the Babelfish for PostgreSQL feature
Makes possible faster, cheaper, and lower-risk migrations from Microsoft SQL Server to Aurora PostgreSQL.

Available versions (2/20) [Info](#)

Aurora PostgreSQL (Compatible with PostgreSQL 13.4) ▾

7. For **Templates**, choose the template that matches your use case.
8. For **DB cluster identifier**, enter a name that you can easily find later in the DB cluster list.
9. For **Master username**, enter an administrator user name.

Unlike SQL Server, Babelfish doesn't create an `sa` login. To create a login named `sa`, enter the name for **Master username**.

If you don't create a user named `sa` at this time, you can create one later with your choice of client. After creating the user, use the `ALTER SERVER ROLE` command to add it to `sysadmin`.

10. For **Master password**, enter a strong password for the administrative user that you just named, and confirm the password.
11. For the options that follow, until the **Babelfish settings** section, specify your DB cluster settings. For information about each setting, see [Settings for Aurora DB clusters \(p. 137\)](#).
12. To make Babelfish functionality available, select the **Turn on Babelfish** box.

Babelfish settings - new [Info](#)

Turn on Babelfish
Makes possible faster, cheaper, and lower-risk migrations from Microsoft SQL Server to Aurora PostgreSQL.

i **Babelfish default configurations**
By default, RDS creates a DB cluster parameter group for you to store the Babelfish settings. Babelfish uses default values if you don't modify these settings in the "Additional configuration" section below.

13. For **DB cluster parameter group**, do one of the following:
 - Choose **Create new** to create a new parameter group with Babelfish turned on.
 - Choose **Choose existing** to use an existing parameter group. If you use an existing group, make sure to modify the group before creating the cluster and add values for Babelfish parameters. For information about Babelfish parameters, see [Configuring a database for Babelfish \(p. 1345\)](#).

If you use an existing group, provide the group name in the box that follows.
14. For **Database migration mode**, choose one of the following:
 - **Single database** to migrate a single SQL Server database.

In some cases, you might migrate multiple user databases together, with your end goal a complete migration to native Aurora PostgreSQL without Babelfish. If the final applications require consolidated schemas (a single dbo schema), make sure to first consolidate your SQL Server databases into a single SQL server database. Then migrate to Babelfish using **Single database** mode.

- **Multiple databases** to migrate multiple SQL Server databases (originating from a single SQL Server installation). Multiple database mode doesn't consolidate multiple databases that don't originate from a single SQL Server installation. For information about migrating multiple databases, see [Using Babelfish with a single database or multiple databases \(p. 1298\)](#).

▼ Additional configuration

Database options, encryption enabled, failover, backup enabled, backtrack disabled, Performance Insights enabled, Enhanced Monitoring enabled, maintenance, CloudWatch Logs, delete protection disabled.

Database options

DB cluster parameter group [Info](#)

Choose a compatible DB Cluster parameter group to turn on Babelfish feature for your database.

Create new

Creates a custom DB cluster parameter group with Babelfish parameters turned on.

Choose existing

Choose an existing DB cluster parameter group with Babelfish parameters turned on.

New custom DB cluster parameter group name

custom-aurora-postgresql13-babelfish-compat-1

Babelfish configuration

Database migration mode [Info](#)

Single database

Use for migrating a single SQL Server database. Migrated schema names are identical between TDS connections and PostgreSQL connections.

Multiple databases

Use for migrating multiple SQL Server databases together. Migrated database and schema names are mapped to similar schema names in PostgreSQL.

15. For **Default collation locale**, enter your server locale. The default is en-US. For detailed information about collations, see [Babelfish collation support \(p. 1349\)](#).
16. For **Collation name** field, enter your default collation. The default is sql_latin1_general_cp1_ci_as. For detailed information, see [Babelfish collation support \(p. 1349\)](#).
17. For **Babelfish TDS port**, enter the port number for your SQL Server client connect to. The default is 1433.
18. For **DB parameter group**, choose a parameter group or have Aurora create a new group for you with default settings.
19. For **Failover priority**, choose a failover priority for the instance. If you don't choose a value, the default is tier-1. This priority determines the order in which replicas are promoted when recovering from a primary instance failure. For more information, see [Fault tolerance for an Aurora DB cluster \(p. 69\)](#).
20. For **Backup retention period**, choose the length of time (1–35 days) that Aurora retains backup copies of the database. You can use backup copies for point-in-time restores (PITR) of your database down to the second. The default retention period is seven days.

Default collation locale [Info](#)

en-US

Collation name [Info](#)

sql_latin1_general_cp1_ci_as

Babelfish TDS port [Info](#)

TDS port that the database will use for application connections.

1433

DB parameter group [Info](#)

default.aurora-postgresql13

Option group [Info](#)

default:aurora-postgresql-13

Failover priority

No preference

Backup

Backup retention period [Info](#)

Choose the number of days that RDS should retain automatic backups for this instance.

7 days

21. Choose **Copy tags to snapshots** to copy any DB instance tags to a DB snapshot when you create a snapshot.
22. Choose **Enable encryption** to turn on encryption at rest (Aurora storage encryption) for this DB cluster.
23. Choose **Enable Performance Insights** to turn on Amazon RDS Performance Insights.
24. Choose **Enable Enhanced monitoring** to start gathering metrics in real time for the operating system that your DB cluster runs on.
25. Choose **PostgreSQL log** to publish the log files to Amazon CloudWatch Logs.
26. Choose **Enable auto minor version upgrade** to automatically update your Aurora DB cluster when a minor version upgrade is available.
27. For **Maintenance window**, do the following:
 - To choose a time for Amazon RDS to make modifications or perform maintenance, choose **Select window**.

- To perform Amazon RDS maintenance at an unscheduled time, choose **No preference**.
28. Select the **Enable deletion protection** box to protect your database from being deleted by accident.

If you turn on this feature, you can't directly delete the database. Instead, you need to modify the database cluster and turn off this feature before deleting the database.

Maintenance

Auto minor version upgrade [Info](#)

Enable auto minor version upgrade
Enabling auto minor version upgrade will automatically upgrade to new minor versions as they are released. The automatic upgrades occur during the maintenance window for the database.

Maintenance window [Info](#)
Select the period you want pending modifications or maintenance applied to the database by Amazon RDS.

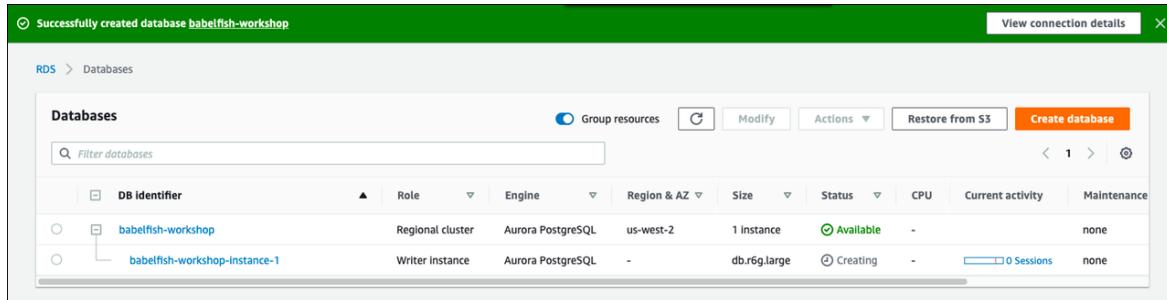
Select window
 No preference

Deletion protection

Enable deletion protection
Protects the database from being deleted accidentally. While this option is enabled, you can't delete the database.

29. Choose **Create database**.

You can find your new database set up for Babelfish in the **Databases** listing. The **Status** column displays **Available** when the deployment is complete.



The screenshot shows the AWS RDS Databases page. At the top, a green banner indicates "Successfully created database babelfish-workshop". Below this, the page title is "RDS > Databases". A search bar labeled "Filter databases" is present. The main table has columns: DB identifier, Role, Engine, Region & AZ, Size, Status, CPU, Current activity, and Maintenance. There are two entries:

DB identifier	Role	Engine	Region & AZ	Size	Status	CPU	Current activity	Maintenance
babelfish-workshop	Regional cluster	Aurora PostgreSQL	us-west-2	1 instance	Available	-	-	none
babelfish-workshop-instance-1	Writer instance	Aurora PostgreSQL	-	db.r6g.large	Creating	-	0 Sessions	none

AWS CLI

Before you create an Aurora DB cluster with Babelfish running using the AWS CLI, make sure to fulfill the required prerequisites, such as creating a parameter group. For more information, see [DB cluster prerequisites \(p. 125\)](#).

Before you can use the AWS CLI to create an Aurora PostgreSQL cluster with Babelfish, do the following:

- Choose your endpoint URL from the list of services at [Amazon Aurora endpoints and quotas](#).
- Create a parameter group for the cluster. For more information about parameter groups, see [Working with DB parameter groups and DB cluster parameter groups \(p. 339\)](#).
- Modify the parameter group, adding the parameter that turns on Babelfish.

To create an Aurora PostgreSQL DB cluster with Babelfish using the AWS CLI

1. Create a parameter group.

For Linux, macOS, or Unix:

```
aws rds create-db-cluster-parameter-group \
--endpoint-url my_endpoint_URL \
--db-cluster-parameter-group-name my_parameter_group \
--db-parameter-group-family aurora-postgresql13 \
--description "parameter_group_description"
```

For Windows:

```
aws rds create-db-cluster-parameter-group ^
--endpoint-url my_endpoint_URL ^
--db-cluster-parameter-group-name my_parameter_group ^
--db-parameter-group-family aurora-postgresql13 ^
--description "parameter_group_description"
```

2. Modify your parameter group to turn on Babelfish.

For Linux, macOS, or Unix:

```
aws rds modify-db-cluster-parameter-group \
--endpoint-url my_endpoint_URL \
--db-cluster-parameter-group-name my_parameter_group \
--parameters "ParameterName=rds.babelfish_status,ParameterValue=on,ApplyMethod=pending-reboot"
```

For Windows:

```
aws rds modify-db-cluster-parameter-group ^
--endpoint-url my_endpoint_URL ^
--db-cluster-parameter-group-name my_parameter_group ^
--parameters "ParameterName=rds.babelfish_status,ParameterValue=on,ApplyMethod=pending-reboot"
```

3. Identify your DB subnet group and the virtual private cloud (VPC) security group ID for your new DB cluster, and then call the [create-db-cluster](#) command.

For Linux, macOS, or Unix:

```
aws rds create-db-cluster \
--db-cluster-identifier my_cluster_name \
--master-username user_name \
--master-user-password my_password \
--engine aurora-postgresql \
--engine-version 13.4 \
--vpc-security-group-ids my_security_group \
--db-subnet-group-name my_subnet_group \
--db-cluster-parameter-group-name my_parameter_group
```

For Windows:

```
aws rds create-db-cluster ^
--db-cluster-identifier my_cluster_name ^
--master-username user_name ^
--master-user-password my_password ^
```

```
--engine aurora-postgresql ^
--engine-version 13.4 ^
--vpc-security-group-ids my_security_group ^
--db-subnet-group-name my_subnet_group ^
--db-cluster-parameter-group-name my_parameter_group
```

4. Explicitly create the primary instance. Use the name of the cluster that you created preceding for the value of the --db-cluster-identifier option and run the [create-db-instance](#) command as shown following.

For Linux, macOS, or Unix:

```
aws rds create-db-instance \
--db-instance-identifier my_instance_name \
--db-instance-class db.r5.4xlarge \
--db-subnet-group-name my_subnet_group \
--db-cluster-identifier my_cluster_name \
--engine aurora-postgresql
```

For Windows:

```
aws rds create-db-instance ^
--db-instance-identifier my_instance_name ^
--db-instance-class db.r5.4xlarge ^
--db-subnet-group-name my_subnet_group ^
--db-cluster-identifier my_cluster_name ^
--engine aurora-postgresql
```

Connecting to a DB cluster with Babelfish turned on

To connect to Babelfish, modify your database client configuration to connect to the endpoint of the Aurora PostgreSQL cluster running Babelfish. Your client can use one of the following client drivers compliant with TDS version 7.1 or higher:

- Open Database Connectivity (ODBC)
- OLE DB Driver/MSOLEDBSQL
- Java Database Connectivity (JDBC)
- Microsoft SqlClient Data Provider for SQL Server
- .NET Data Provider for SQL Server
- SQL Server Native Client 11.0 (deprecated)
- OLEDB Provider/SQLOLEDB (deprecated)

With Babelfish, you run the following:

- SQL Server tools, applications, and syntax on the TDS port, by default port 1433.
- PostgreSQL tools, applications, and syntax on the PostgreSQL port, by default port 5432.

Note

Babelfish for Aurora PostgreSQL doesn't support MARS (Multiple Active Result Sets). Be sure that any client applications that you use to connect to Babelfish aren't set to use MARS.

If you're new to connecting to an Amazon Aurora PostgreSQL database, see also [Connecting to an Amazon Aurora PostgreSQL DB cluster \(p. 285\)](#).

Finding the DNS writer endpoint and port number

Use the following procedure to find your database endpoint.

To find your database endpoint

1. Open the console for Babelfish.
2. Choose **Databases** from the navigation pane.

Your database should have a status of **Available**. If it doesn't, wait until it displays as **Available**. The status updates automatically without requiring you to refresh the page. This process can take up to 20 minutes after creating a DB cluster.

3. Choose your DB cluster that supports Babelfish to show its details.
4. On the **Connectivity & security** tab, note the available cluster **Endpoints** values. Use the cluster endpoint for the writer instance in your connection strings for any applications that perform database write or read operations.

The screenshot shows the AWS RDS console for the 'babelfish-workshop' database. In the 'Related' section, the cluster identifier 'babelfish-workshop' is highlighted with a red circle. In the 'Endpoints' section, the first endpoint entry, which is the writer instance, is also highlighted with a red circle.

Endpoint name	Status	Type	Port
<code>babelfish-workshop.cluster-ro-</code>	Available	Reader instance	5432, 1433 (Babelfish)
<code>babelfish-workshop.cluster-</code>	Available	Writer instance	5432, 1433 (Babelfish)

For more information about DB cluster details, see [Creating an Amazon Aurora DB cluster \(p. 125\)](#).

Performing client authentication

Aurora PostgreSQL with Babelfish supports password authentication. Passwords are stored in encrypted form on disk. For more information about authentication on an Aurora PostgreSQL cluster, see [Security with Amazon Aurora PostgreSQL \(p. 1276\)](#).

You might be prompted for credentials each time you connect to Babelfish. Any user migrated to or created on Aurora PostgreSQL can use the same credentials on both the SQL Server port and the PostgreSQL port. Babelfish doesn't enforce password policies, but we recommend you do the following:

- Require a complex password that is at least eight characters long.
- Enforce a password expiration policy.

To review a complete list of database users, use the command `SELECT * FROM pg_user;`.

Configuring a client to connect to the DB cluster

To see how to connect a client to a DB cluster that supports Babelfish, see the code examples following.

Example Using C# code to connect to a DB cluster

```
string dataSource = 'babelfishServer_11_24';

//Create connection
connectionString = @"Data Source=" + dataSource
    +";Initial Catalog=your-DB-name"
    +";User ID=user-id;Password=password";

SqlConnection cnn = new SqlConnection(connectionString);
cnn.Open();
```

Example Using generic JDBC API classes and interfaces to connect to a DB cluster

```
String dbServer =
    "database-babelfish.cluster-123abc456def.us-east-1-rds.amazonaws.com";
String connectionUrl = "jdbc:sqlserver://" + dbServer + ":1433;" +
    "databaseName=your-DB-name;user=user-id;password=password";

// Load the SQL Server JDBC driver and establish the connection.
System.out.print("Connecting Babelfish Server ... ");
Connection cnn = DriverManager.getConnection(connectionUrl);
```

Example Using SQL Server-specific JDBC classes and interfaces to connect to a DB cluster

```
// Create datasource.
SQLServerDataSource ds = new SQLServerDataSource();
ds.setUser("user-id");
ds.setPassword("password");
String babelfishServer =
    "database-babelfish.cluster-123abc456def.us-east-1-rds.amazonaws.com";

ds.setServerName(babelfishServer);
ds.setPortNumber(1433);
ds.setDatabaseName("your-DB-name");

Connection con = ds.getConnection();
```

Using a SQL Server client to connect to your DB cluster

You can use a SQL Server client to connect with Babelfish on the TDS port.

Using sqlcmd to connect to the DB cluster

You can connect to and interact with an Aurora PostgreSQL DB cluster that supports Babelfish by using the SQL Server `sqlcmd` command line client. Use the following command to connect.

```
sqlcmd -S endpoint, port -U login-id -P password -d your-DB-name
```

The options are as follows:

- `-S` is the endpoint and (optional) TDS port of the DB cluster.
- `-U` is the login name of the user.
- `-P` is the password associated with the user.
- `-d` is the name of your Babelfish database.

After connecting, you can use many of the same commands that you use with SQL Server. For some examples, see [Querying a database for object information \(p. 1318\)](#).

To review a list of exceptions, see [Differences between Aurora PostgreSQL with Babelfish and SQL Server \(p. 1322\)](#).

Using SSMS to connect to the DB cluster

You can connect to an Aurora PostgreSQL DB cluster that supports Babelfish by using Microsoft SQL Server Management Studio (SSMS). SSMS includes a variety of tools. By default, SSMS might be configured to launch the SSMS Object Explorer. For connecting to your Babelfish database with SSMS, you can use only the SSMS Query Editor. Currently, only the Query Editor is supported.

To connect to your Babelfish database with SSMS

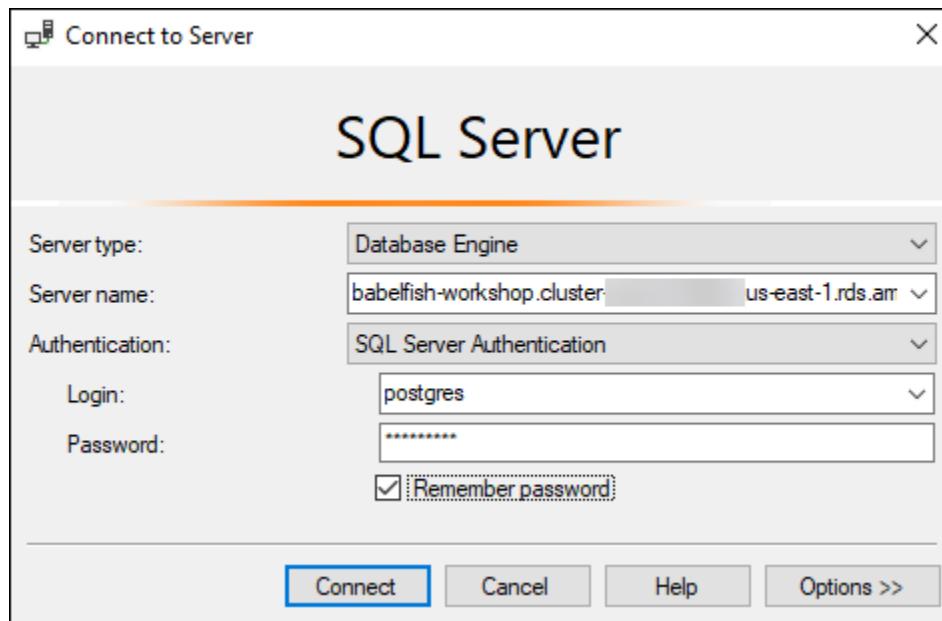
1. Start SSMS.
2. Open the **Connect to Server** dialog box. Be sure that the Query Editor connection dialog opens, not the Object Explorer. To continue with the connection, do one of the following:
 - Choose **New Query**.
 - If the Query Editor is open, choose **Query, Connection, Connect**.

Note

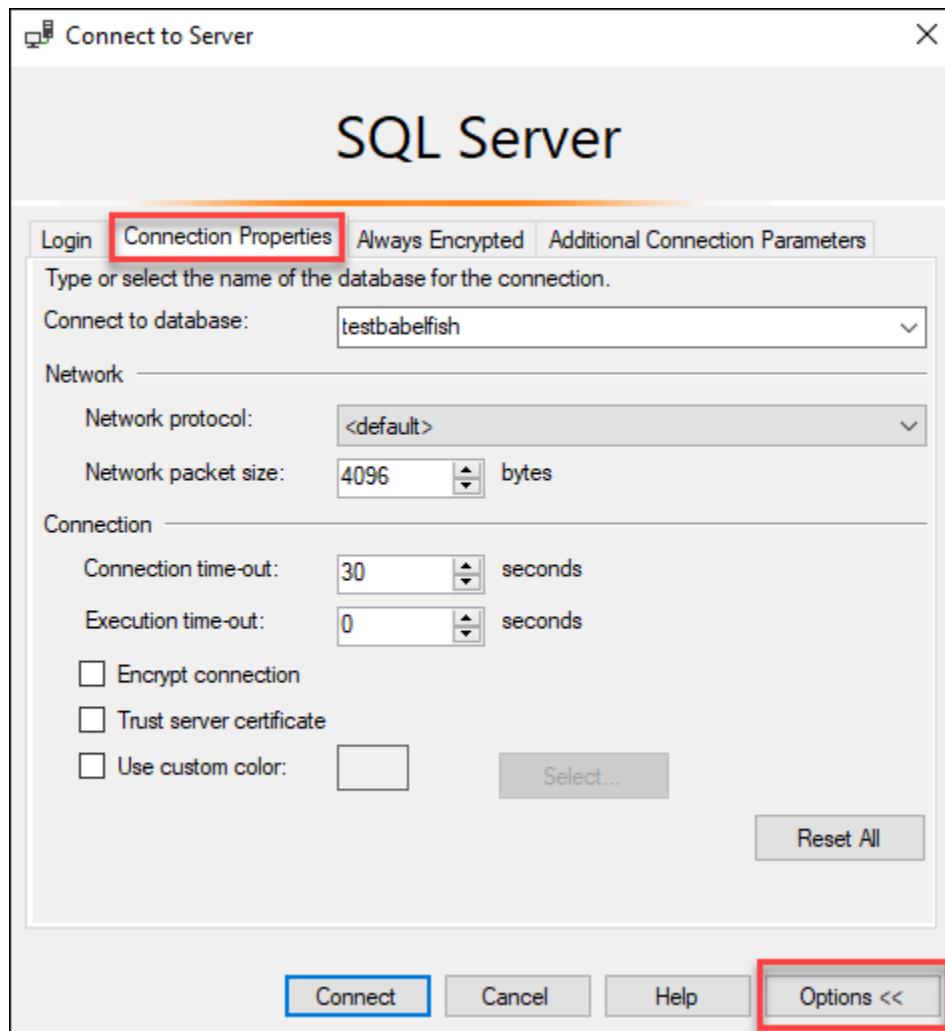
If the Object Explorer's dialog opens, cancel the dialog and re-open the Query Editor.

3. Provide the following information for your database:
 - a. For **Server type**, choose **Database Engine**.
 - b. For **Server name**, enter the DNS name. For example, your server name should look similar to the following.

```
cluster-name.cluster-555555555555.aws-region.rds.amazonaws.com,1433
```
 - c. For **Authentication**, choose **SQL Server Authentication**.
 - d. For **Login**, enter the user name that you chose when you created your database.
 - e. For **Password**, enter the password that you chose when you created your database.



4. (Optional) Choose **Options**, and then choose the **Connection Properties** tab.



5. (Optional) For **Connect to database**, specify the name of the migrated SQL Server database to connect to, and choose **Connect**.

If a message appears indicating that SSMS can't apply connection strings, choose **OK**.

If you are having trouble connecting, see [Troubleshooting connections to your SQL Server DB instance](#) in the *Amazon RDS User Guide*.

Using a PostgreSQL client to connect to your DB cluster

You can use a PostgreSQL client to connect to Babelfish on the PostgreSQL port.

Using psql to connect to the DB cluster

You can query an Aurora PostgreSQL DB cluster that supports Babelfish with the `psql` command line client. When connecting, use the PostgreSQL port. Use the following command to connect to Babelfish with the `psql` client:

```
psql "host=babelfish_db.cluster-123456789012  
port=portNumber dbname=babelfish_db user=userName"
```

The parameters are as follows:

- `host` – The host name of the DB cluster (cluster endpoint) that you want to access
- `port` – The PostgreSQL port number used to connect to your DB instance
- `dbname` – `babelfish_db`
- `user` – The database user account that you want to access
- `password` – The password of the database user

When you run a SQL command on the `psql` client, you end the command with a semicolon. For example, the following SQL command queries the [pg_tables system view](#) to return information about each table in the database.

```
SELECT * FROM pg_tables;
```

The `psql` client also has a set of built-in metacommands. A *metacommand* is a shortcut that adjusts formatting or provides a shortcut that returns meta-data in an easy-to-use format. For example, the following metacommand returns similar information to the previous SQL command:

```
\d
```

Metacommands don't need to be terminated with a semicolon (;).

To exit the `psql` client, enter `\q`.

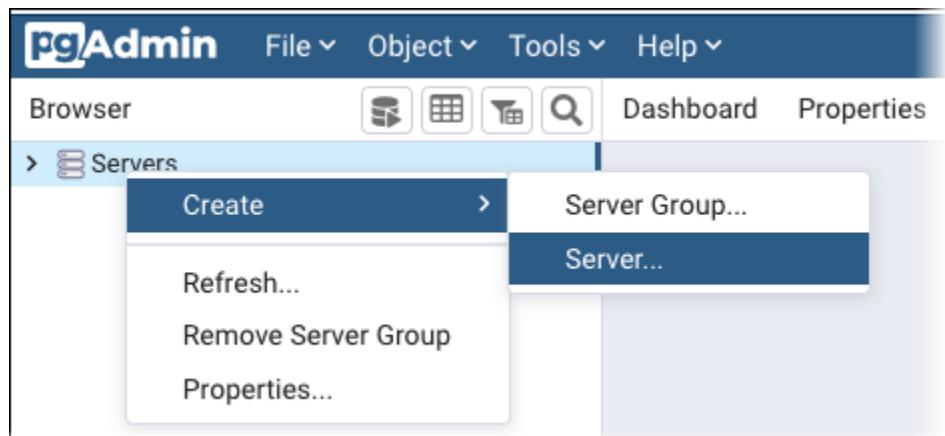
For more information about using the `psql` client to query an Aurora PostgreSQL cluster, see the [PostgreSQL documentation](#).

Using pgAdmin to connect to the DB cluster

You can use the pgAdmin client to access your data in native PostgreSQL dialect.

To connect to the cluster with the pgAdmin client

1. Download and install the pgAdmin client from the [pgAdmin website](#).
2. Open the client and authenticate with pgAdmin.
3. Open the context (right-click) menu for **Servers**, and then choose **Create, Server**.



4. Enter information in the **Create - Server** dialog box.

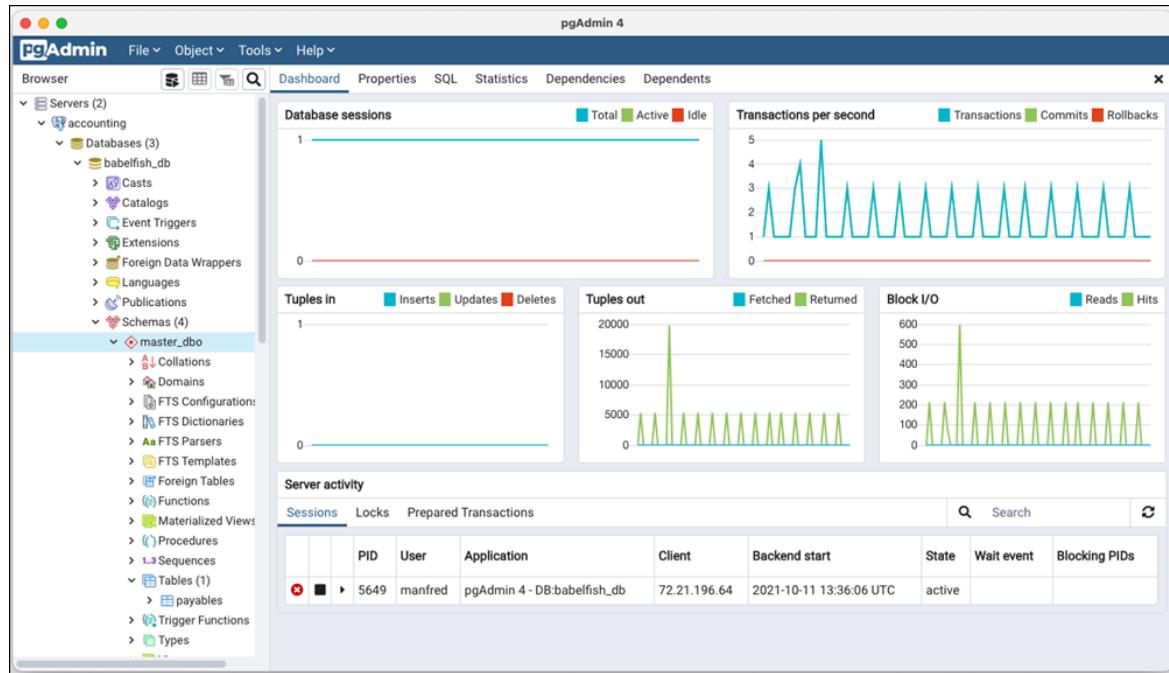
On the **Connection** tab, add the Aurora PostgreSQL cluster address for **Host** and the PostgreSQL port number (by default, 5432) for **Port**. Provide authentication details, and choose **Save**.

The screenshot shows the "Create - Server" dialog box. The title bar says "Create - Server". The tabs at the top are "General", "Connection", "SSL", "SSH Tunnel", and "Advanced". The "Connection" tab is selected. The fields in the "Connection" tab are:

Host name/address	babelfish_db.cluster- .us-east-1.rds.ama
Port	5432
Maintenance database	babelfish_db
Username	postgres
Kerberos authentication?	(Toggle switch)
Password
Save password?	(Toggle switch)
Role	
Service	

At the bottom of the dialog box are three buttons: "i" (Info), "?", and "Close". To the right of the "Close" button are "Reset" and "Save" buttons.

After connecting, you can use pgAdmin functionality to monitor and manage your Aurora PostgreSQL cluster on the PostgreSQL port.



For more details about using pgAdmin, visit the [pgAdmin web site](#).

Querying a database for object information

To return information about database objects that are stored in your Aurora PostgreSQL cluster, you can query many of the same system views that you use on SQL Server. You can access these views from either the TDS port or the PostgreSQL port.

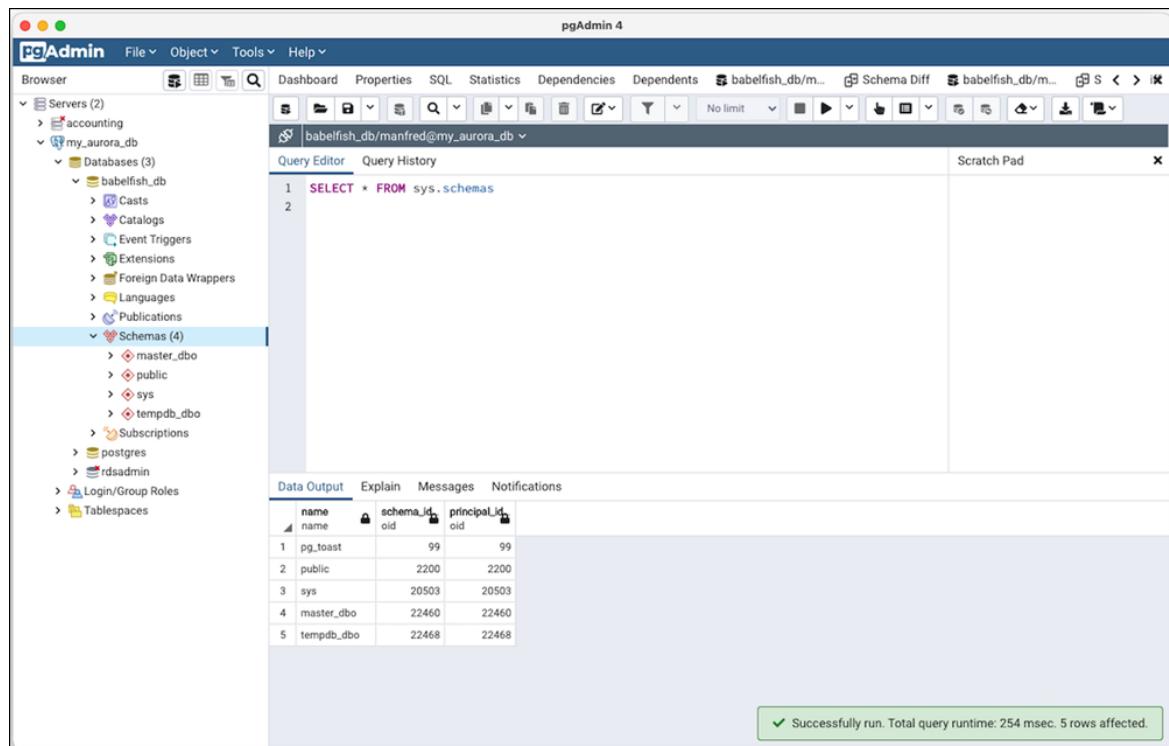
For example, to find a list of schemas in your migrated database on the T-SQL port, connect to the TDS port with `sqlcmd`, and use the following command.

```
SELECT * FROM sys.schemas
```

If you migrate a single-db or multi-db database, Babelfish returns a list of schema names formatted in Babelfish style that includes both the SQL Server and PostgreSQL system schemas:

```
mydb_dbo
public
sys
master_dbo
temp_dbo
```

You get the same result set if you connect with a PostgreSQL client on the database port (by default, 5432). For example, querying the database with pgAdmin returns the following.



Use SQL Server and PostgreSQL views to return information about objects in your Aurora PostgreSQL cluster. A few of the SQL Server views implemented by Babelfish follow:

View name	Description
sys.all_views	All views in all schemas
sys.schemas	All schemas
sys.databases	All databases in all schemas
sys.server_principals	All logins and roles
sys.all_objects	All objects in all schemas
sys.tables	All tables in a schema
sys.all_columns	All columns in all tables and views
sys.columns	All columns in user-defined tables and views

PostgreSQL implements system catalogs that are similar to the SQL Server object catalog views. For a complete list of system catalogs, see [System Catalogs](#) in the PostgreSQL documentation.

Querying Babelfish to find Babelfish details

You can query Babelfish to find details about the Babelfish version, the Aurora PostgreSQL version, and the compatible Microsoft SQL Server version.

Run these queries while connected to the TDS port.

To identify the Babelfish version, run the following query:

```
SELECT CAST(serverproperty('babelfishversion') AS VARCHAR)
```

The query returns results similar to the following:

```
1.0.0
```

To identify the version of the Aurora PostgreSQL DB cluster, run the following query:

```
SELECT aurora_version() AS aurora_version
```

The query returns results similar to the following:

```
13.4.0
```

To identify the compatible Microsoft SQL Server version, run the following query:

```
SELECT @@VERSION AS version
```

The query returns results similar to the following:

```
Babelfish for Aurora Postgres with SQL Server Compatibility - 12.0.2000.8
Sep 28 2021 14:37:26
Copyright (c) Amazon Web Services
PostgreSQL 13.4 on x86_64-pc-linux-gnu
```

In addition, the following query returns 1 when executed on Babelfish, and NULL when executed on Microsoft SQL Server:

```
SELECT CAST(serverproperty('babelfish') AS VARCHAR) AS runs_on_babelfish
```

To query `babelfish_db` the same way using the PostgreSQL port, connect to the `babelfish_db` and run the following.

```
\x
SELECT
aurora_version() as aurora_version,
version() as postgresql_version,
sys.version() as Babelfish_compatibility,
sys.SERVERPROPERTY('BabelfishVersion') as Babelfish_Version
```

The query returns the following.

```
babelfish_db=> \x
Expanded display is on.
babelfish_db=> SELECT
babelfish_db-> aurora_version() as aurora_version,
babelfish_db-> version() as postgresql_version,
babelfish_db-> sys.version() as Babelfish_compatibility,
babelfish_db-> sys.SERVERPROPERTY('BabelfishVersion') as Babelfish_Version ;
-[ RECORD 1 ]
aurora_version      | 13.4.0
postgresql_version | PostgreSQL 13.4 on aarch64-unknown-linux-gnu, compiled by
aarch64-unknown-linux-gnu-gcc (GCC) 7.4.0, 64-bit
```

```
babelfish_compatibility | Babelfish for Aurora Postgres with SQL Server Compatibility -  
12.0.2000.8  
+  
| Oct 13 2021 17:34:47  
+  
| Copyright (c) Amazon Web Services  
+  
| PostgreSQL 13.4 on aarch64-unknown-linux-gnu  
babelfish_version | 1.0.0
```

Differences between Aurora PostgreSQL with Babelfish and SQL Server

Babelfish provides support for T-SQL and Microsoft SQL Server behavior by supporting SQL Server data types, syntax, and functions for Aurora PostgreSQL. This approach allows Aurora to support both Aurora PostgreSQL and SQL Server SQL dialects. Also, Babelfish supports the SQL Server wire-level protocol (TDS), allowing a SQL Server application to communicate natively with Aurora PostgreSQL. Doing this helps migrate database objects, stored procedures, and application code with fewer changes.

Although Babelfish doesn't offer complete support for T-SQL, you can use Aurora PostgreSQL SQL commands to perform many of the tasks normally handled by these commands. For example, suppose that you regularly use a specific T-SQL command that isn't supported by Babelfish. In this case, you can connect to the Aurora PostgreSQL port and use a PostgreSQL SQL command instead. For more information, see [SQL Commands](#) in the PostgreSQL documentation.

Aurora PostgreSQL offers functionality to replace many commonly used SQL Server features. Some examples of SQL Server functionality that can be replaced by the PostgreSQL functionality available in Aurora PostgreSQL follow. In the table, references are to the PostgreSQL documentation.

SQL Server functionality	Comparable PostgreSQL functionality
SQL Server bulk copy	PostgreSQL COPY (optimized for fast data loading)
SQL Server GROUP BY clauses (not supported in Babelfish)	PostgreSQL GROUPING SETS
SQL Server JSON support	PostgreSQL JSON functions and operators
SQL Server XML support	PostgreSQL XML functions
SQL Server full-text search	PostgreSQL full-text search
SQL Server GEOGRAPHY data type	PostGIS extension (for working with geographical data)

To help with cluster management in Aurora PostgreSQL, you can use its scalability, high-availability with failover support, and built-in replication. For more information about these capabilities, see [Managing performance and scaling for Aurora DB clusters \(p. 396\)](#), [High availability for Amazon Aurora \(p. 68\)](#), and [Replication with Amazon Aurora \(p. 70\)](#). You also have access to other AWS tools and utilities:

- [Amazon CloudWatch](#) is a monitoring and observability service that provides you with data and actionable insights.
- [Amazon RDS Performance Insights](#) is a database performance tuning and monitoring feature that helps you quickly assess the load on your database.
- [Amazon RDS Multi-AZ deployments](#) provide enhanced availability and durability for your database cluster.
- [Amazon RDS global databases](#) allow a single Amazon Aurora database to span multiple AWS Regions, offering scalable, cross-Region replication.
- Automatic software patching keeps your database up-to-date with the latest security and feature patches when they become available.
- [Overview of Amazon RDS event notification \(p. 675\)](#) Amazon RDS events notify you by email or SMS message of important database events, such as an automated failover.

Topics

- [T-SQL limitations and unsupported functionality \(p. 1323\)](#)
- [Unsupported functionality in Babelfish \(p. 1330\)](#)

T-SQL limitations and unsupported functionality

Following, you can find a table of limitations or partially supported T-SQL syntax for Babelfish. Unless otherwise specified, these limitations apply to Babelfish 1.0.0. For more information about Babelfish releases, see [Babelfish versions \(p. 1358\)](#).

Functionality or syntax	Notes
<code>@@version</code>	The format of the value returned by <code>@@version</code> is slightly different from the value returned by SQL Server. Your code might not work correctly if it depends on the formatting of <code>@@version</code> .
Aggregate functions (partially supported)	<code>APPROX_COUNT_DISTINCT</code> , <code>CHECKSUM_AGG</code> , <code>GROUPING_ID</code> , <code>ROWCOUNT_BIG</code> , <code>STDEV</code> , <code>STDEVP</code> , <code>VAR</code> , and <code>VARP</code> aren't supported.
<code>ALTER TABLE</code>	Supports adding or dropping a single column or constraint only.
Assembly modules and SQL Common Language Runtime (CLR) routines	Functionality related to assembly modules and CLR routines isn't supported.
<code>BACKUP</code> statement	Aurora PostgreSQL snapshots of a database are dissimilar to backup files created in SQL Server. Also, the granularity of when a backup and restore occurs might be different between SQL Server and Aurora PostgreSQL.
Blank column names with no column alias	The <code>sqlcmd</code> and <code>psql</code> utilities handle columns with blank names differently: <ul style="list-style-type: none">• SQL Server <code>sqlcmd</code> returns a blank column name.• PostgreSQL <code>psql</code> returns a generated column name.
Collation, index on type dependent on the ICU library	An index on a user-defined type that depends on the International Components for Unicode (ICU) collation library (the library used by Babelfish) isn't invalidated when the version of the library is changed. For more information about collations, see Babelfish collation support (p. 1349) .
<code>COLLATIONPROPERTY</code> function	Collation properties are implemented only for the supported Babelfish BBF collations. For more information about collations, see Babelfish collation support (p. 1349) .
Column default	When creating a column default, the constraint name is ignored. To drop a column default, use the following syntax: <code>ALTER TABLE...ALTER COLUMN..DROP DEFAULT...</code>
Column name case	Column names are stored as lowercase in the PostgreSQL catalogs and are returned to the client in lowercase if you run a <code>SELECT</code> statement. In general, all schema identifiers are stored in lowercase in the PostgreSQL catalogs. For more information, see SQL-SYNTAX-IDENTIFIERS in the PostgreSQL documentation.

Functionality or syntax	Notes
Column attributes	ROWGUIDCOL, SPARSE, FILESTREAM, and MASKED aren't supported.
CONNECTIONPROPERTY function	The unsupported properties include local_net_address, client_net_address, and physical_net_transport.
Constraints	PostgreSQL doesn't support turning on and turning off individual constraints. The statement is ignored and a warning is raised.
Constraints created with DESC (descending) columns	Constraints are created with ASC (ascending) columns.
Constraints with IGNORE_DUP_KEY	Constraints are created without this property.
Contained databases	Contained databases with logins authenticated at the database level rather than at the server level aren't supported.
CREATE, ALTER, DROP SERVER ROLE	<p>ALTER SERVER ROLE is supported only for sysadmin. All other syntax is unsupported.</p> <p>The T-SQL user in Babelfish has an experience that is similar to SQL Server for the concepts of a login (server principal), a database, and a database user (database principal).</p> <p>Only the dbo user is available in Babelfish user databases. To operate as the dbo user, a login must be a member of the server-level sysadmin role (<code>ALTER SERVER ROLE sysadmin ADD MEMBER <i>login</i></code>). Logins without sysadmin role can currently access only master and tempdb as the guest user.</p> <p>Currently, because Babelfish supports only the dbo user in user databases, all application users must use a login that is a sysadmin member. You can't create a user with lesser privileges, such as read-only on certain tables.</p>
CREATE DATABASE case-sensitive collation	Case-sensitive collations aren't supported with the CREATE DATABASE statement.
CREATE DATABASE keywords and clauses	Options except COLLATE and CONTAINMENT=NONE aren't supported. The COLLATE clause is accepted and is always set to the value of <code>babelfishpg_tsql.server_collation_name</code> .
CREATE SCHEMA... supporting clauses	You can use the CREATE SCHEMA command to create an empty schema. Use additional commands to create schema objects.
CREATE USER	This syntax isn't supported. The PostgreSQL statement CREATE USER doesn't create a user that is equivalent to the SQL Server CREATE USER syntax.
CREATE, ALTER LOGIN clauses are supported with limited syntax	The CREATE LOGIN... PASSWORD clause, ...DEFAULT_DATABASE clause, and ...DEFAULT_LANGUAGE clause are supported. The ALTER LOGIN... PASSWORD clause is supported, but ALTER LOGIN... OLD_PASSWORD clause isn't supported. Only a login that is a sysadmin member can modify a password.
LOGIN objects	All options for LOGIN objects except: PASSWORD, DEFAULT_DATABASE, ENABLE, DISABLE

Functionality or syntax	Notes
CROSS APPLY	Lateral joins aren't supported.
Cross-database object references	Objects with three-part names aren't supported. For more information, see: Using Babelfish with a single database or multiple databases (p. 1298) .
Cursors (updatable)	Updatable cursors aren't supported.
Cursors (global)	GLOBAL cursors aren't supported.
Cursor (fetch behaviors)	The following cursor fetch behaviors aren't supported: FETCH PRIOR, FIRST, LAST, ABSOLUTE, and RELATIVE
Cursor-typed (variables and parameters)	Cursor-typed variables and parameters aren't supported.
Cursor options	SCROLL, KEYSET, DYNAMIC, FAST_FORWARD, SCROLL_LOCKS, OPTIMISTIC, TYPE_WARNING, and FOR UPDATE
Database ID values are different on Babelfish	The master and tempdb databases will not be database IDs 1 and 2.
Data encryption	Data encryption isn't supported.
DBCC commands	Microsoft SQL Server Database Console Commands (DBCC) aren't supported.
DROP IF EXISTS	This syntax isn't supported for USER and SCHEMA objects. It's supported for the objects TABLE, VIEW, PROCEDURE, FUNCTION, and DATABASE.
DROP INDEX	This syntax is supported only in the form <code>DROP index_name ON table_name</code> .
DROP statements that drop multiple objects	This functionality is supported only for tables, views, functions, and procedures.
Encryption	Built-in functions and statements don't support encryption.
ENCRYPT_CLIENT_CERT connections	Client certificate connections aren't supported.
EXECUTE AS statement	This statement isn't supported.
EXECUTE AS SELF clause	This clause isn't supported in functions, procedures, or triggers.
CREATE... EXECUTE AS OWNER clause	EXECUTE AS OWNER or CALLER is supported for permission, but not for name resolution.
EXECUTE AS USER clause	This clause isn't supported in functions, procedures, or triggers.
EXECUTE with AS LOGIN or AT option	This syntax isn't supported.
Foreign key constraints referencing database name	Foreign key constraints that reference the database name aren't supported.
Full-text search	Full-text search built-in Functions and statements aren't supported.

Functionality or syntax	Notes
Function declarations with greater than 100 parameters	Function declarations that contain more than 100 parameters aren't supported.
Function calls that include DEFAULT as a parameter value	DEFAULT isn't a supported parameter value for a function call.
Function calls that include ::	Function calls that include :: aren't supported.
Functions, externally defined	External functions, including SQL CLR functions, aren't supported.
GEOMETRY	This data type and all associated functionality isn't supported.
GEOGRAPHY	This data type and all associated functionality isn't supported.
Global temporary tables (tables with names that start with ##)	Global temporary tables aren't supported.
Graph functionality	All SQL graph functionality isn't supported.
HASHBYTES function	The only supported algorithms are: MD5, SHA1, and SHA256
HIERARCHYID	The data type and methods aren't supported.
Hints	Hints aren't supported for joins, queries, or tables.
Identifiers exceeding 63 characters	PostgreSQL supports a maximum of 63 characters for identifiers. Babelfish converts identifiers longer than 63 characters to a name that includes a hash of the original name.
Identifiers with leading dot characters	Identifiers that start with a . aren't supported in version 1.0.0, but are supported in Babelfish 1.1.0.
Identifiers (variables or parameters) with multiple leading @ characters	Identifiers that start with more than one leading @ aren't supported.
Identifiers, table or column names that contain @ or]] characters	Table or column names that contain an @ sign or square brackets aren't supported.
IDENTITY columns support	<p>IDENTITY columns are supported for data types tinyint, smallint, int, bigint, numeric, and decimal.</p> <p>SQL Server supports precision to 38 places for data types numeric and decimal in IDENTITY columns.</p> <p>PostgreSQL supports precision to 19 places for data types numeric and decimal in IDENTITY columns.</p>
Indexes with IGNORE_DUP_KEY	Syntax that creates an index that includes IGNORE_DUP_KEY creates an index as if this property is omitted.
Indexes with more than 32 columns	An index can't include more than 32 columns. Included index columns count toward the maximum in PostgreSQL but not in SQL Server.
INFORMATION_SCHEMA catalog	Information schema views aren't supported.
Inline indexes	Inline indexes aren't supported.

Functionality or syntax	Notes
Indexes (clustered)	Clustered indexes are created as if NONCLUSTERED was specified.
Index clauses	The following clauses are ignored: FILLFACTOR, ALLOW_PAGE_LOCKS, ALLOW_ROW_LOCKS, PAD_INDEX, STATISTICS_NORECOMPUTE, OPTIMIZE_FOR_SEQUENTIAL_KEY, SORT_IN_TEMPDB, DROP_EXISTING, ONLINE, COMPRESSION_DELAY, MAXDOP, and DATA_COMPRESSION
Invoking a procedure whose name is in a variable	Using a variable as a procedure name isn't supported.
Materialized views	Materialized views aren't supported.
NEWSEQUENTIALID function	Implemented as NEWID; sequential behavior isn't guaranteed. When calling NEWSEQUENTIALID, PostgreSQL generates a new GUID value.
NEXT VALUE FOR sequence clause	This syntax isn't supported.
NOT FOR REPLICATION clause	This syntax is accepted and ignored.
ODBC escape functions	ODBC escape functions aren't supported.
OUTER APPLY	SQL Server lateral joins aren't supported. PostgreSQL provides SQL syntax that allows a lateral join, but the behavior isn't identical. For more information about PostgreSQL lateral joins, see the PostgreSQL documentation .
OUTPUT clause is supported with the following limitations	OUTPUT and OUTPUT INTO aren't supported in the same DML query. References to non-target table of UPDATE or DELETE operations in an OUTPUT clause aren't supported. OUTPUT...DELETED *, INSERTED * aren't supported in the same query.
Partitioning	Table and index partitioning isn't supported.
Procedure calls that includes DEFAULT as a parameter value	DEFAULT isn't a supported parameter value.
Procedure declarations with more than 100 parameters	Declarations with more than 100 parameters aren't supported.
Procedures, externally defined	Externally defined procedures, including SQL CLR procedures, aren't supported.
Procedure versioning	Procedure versioning isn't supported.
Procedures WITH RECOMPILE	WITH RECOMPILE (when used in conjunction with the DECLARE and EXECUTE statements) isn't supported.
Procedure or function parameter limit	Babelfish supports a maximum of 100 parameters for a procedure or function.
Remote object references	Objects with four-part names aren't supported.. For more information, see: Configuring a database for Babelfish (p. 1345) .
Server-level roles other than sysadmin	Server-level roles (other than sysadmin) aren't supported.

Functionality or syntax	Notes
Database-level roles other than db_owner	Database-level roles other than db_owner aren't supported.
RESTORE statement	Aurora PostgreSQL snapshots of a database are dissimilar to backup files created in SQL Server. Also, the granularity of when the backup and restore occurs might be different between SQL Server and Aurora PostgreSQL.
ROLLBACK: table variables don't support transactional rollback	Processing might be interrupted if a rollback occurs in a session with table variables.
ROWGUIDCOL	This clause is currently ignored. Queries referencing \$GUIDCOL cause a syntax error.
Row-level security	Row-level security with CREATE SECURITY POLICY and inline table-valued functions isn't supported.
ROWSET functions	The following ROWSET functions aren't supported: OPENXML, OPENJSON, OPENROWSET, OPENQUERY, OPENDATASOURCE
SELECT... FOR XML PATH, ELEMENTS	Syntax is supported without the ELEMENTS clause.
SELECT... FOR XML RAW, ELEMENTS	Syntax is supported without the ELEMENTS clause.
SERVERPROPERTY	Unsupported properties: BuildClrVersion, ComparisonStyle, ComputerNamePhysicalNetBIOS, EditionID, HadrManagerStatus, InstanceDefaultDataPath, InstanceDefaultLogPath, InstanceName, IsAdvancedAnalyticsInstalled, IsBigDataCluster, IsClustered, IsFullTextInstalled, IsHadrEnabled, IsIntegratedSecurityOnly, IsLocalDB, IsPolyBaseInstalled, IsXTPSupported, LCID, LicenseType, MachineName, NumLicenses, ProcessID, ProductBuild, ProductBuildType, ProductLevel, ProductUpdateLevel, ProductUpdateReference, ResourceLastUpdateDateTime, ResourceVersion, ServerName, SqlCharSet, SqlCharSetName, SqlSortOrder, SqlSortOrderName, FilestreamShareName, FilestreamConfiguredLevel, and FilestreamEffectiveLevel
Service broker functionality	Service broker functionality isn't supported.
SESSIONPROPERTY	Unsupported properties: ANSI_NULLS, ANSI_PADDING, ANSI_WARNINGS, ARITHABORT, CONCAT_NULL_YIELDS_NULL, and NUMERIC_ROUNDABORT
SET LANGUAGE	This syntax isn't supported with any value other than english or us_english.
SEQUENCE object support	SEQUENCE objects are supported for the data types tinyint, smallint, int, bigint, numeric, and decimal. Aurora PostgreSQL supports precision to 19 places for data types numeric and decimal in a SEQUENCE.
SET NUMERIC_ROUNDABORT ON	This setting isn't supported.
SP_CONFIGURE	This system stored procedure isn't supported.

Functionality or syntax	Notes
SQL keyword SPARSE	The keyword SPARSE is accepted and ignored.
SQL keyword clause ON <i>filegroup</i>	This clause is currently ignored.
SQL keywords CLUSTERED and NONCLUSTERED for indexes and constraints	Babelfish accepts and ignores the CLUSTERED and NONCLUSTERED keywords.
System-defined @@ variables	Version 1.0.0 supports the following system-defined @@variables only: @@DATEFIRST, @@ERROR, @@FETCH_STATUS, @@IDENTITY, @@MAX_PRECISION, @@ROWCOUNT, @@SERVERNAME, @@SPID, @@TRANCOUNT, and @@VERSION. Version 1.1.0 adds support for @@CURSOR_ROWS, @@LOCK_TIMEOUT, @@MAX_CONNECTIONS, @@MICROSOFTVERSION, @@NESTLEVEL, and @@PROCID.
<code>sysdatabases.cmptlevel</code>	<code>sysdatabases.cmptlevel</code> are always NULL.
System-provided stored procedures are partially supported	Babelfish version 1.0.0 supported these three stored procedures only: sp_getapplock, sp_helpdb, and sp_releaseapplock. Babelfish 1.1.0 adds support for the following: sp_columns, sp_columns_100, sp_columns_managed, sp_cursor, sp_cursor_list, sp_cursorclose, sp_cursorexecute, sp_cursorfetch, sp_cursoropen, sp_cursoroption, sp_cursorprepare, sp_cursorprepexec, sp_cursorunprepare, sp_databases, sp_datatype_info, sp_datatype_info_100, sp_describe_cursor, sp_describe_first_result_set, sp_describe_declared_parameters, sp_oledb_ro_usename, sp_pkeys, sp_prepare, sp_statistics, sp_statistics_100, sp_tablecollations_100, sp_tables, sp_unprepare, and sp_updatestats.
Table value constructor syntax (FROM clause)	The unsupported syntax is for a derived table constructed with the FROM clause.
tempdb isn't reinitialized at restart	Permanent objects (like tables and procedures) created in tempdb aren't removed when the database is restarted.
Temporal tables	Temporal tables aren't supported.
Temporary procedures aren't dropped automatically	This functionality isn't supported.
TEXTIMAGE_ON filegroup	Babelfish ignores the TEXTIMAGE_ON <i>filegroup</i> clause.
Time precision	Babelfish supports 6-digit precision for fractional seconds. No adverse effects are anticipated with this behavior.
Transaction isolation levels	READUNCOMMITTED is treated the same as READCOMMITTED. REPEATABLEREAD, and SERIALIZABLE aren't supported.
TIMESTAMP data type	This data type isn't supported. The SQL Server TIMESTAMP type is unrelated to PostgreSQL TIMESTAMP.
Triggers, externally defined	These triggers aren't supported, including SQL Common Language Runtime (CLR).

Functionality or syntax	Notes
Trigger for multiple DML actions cannot reference transition tables	Triggers that reference multiple DML actions can't reference transition tables in version 1.0.0 of Babelfish. Version 1.1.0 provides support for this functionality.
Unquoted string values in stored procedure calls and default values	String parameters to stored procedure calls, and defaults for string parameters in CREATE PROCEDURE, are not supported.
Virtual computed columns (non-persistent)	Virtual computed columns are created as persistent.
WITH ENCRYPTION clause	This syntax isn't supported for functions, procedures, triggers, or views.
WITHOUT SCHEMABINDING clause	This clause isn't supported in functions, procedures, triggers, or views. The object is created, but as if WITH SCHEMABINDING was specified.
XML data type with schema (xmlschema)	XML type without schema is supported.
XML indexes	XML indexes aren't supported.
XML methods	XML methods aren't supported, including .VALUES, .NODES, and other methods.
XPATH expressions	This syntax isn't supported.
WITH XMLNAMESPACES construct	This syntax isn't supported.

Unsupported functionality in Babelfish

In the following lists, you can find functionality that isn't supported in Babelfish version 1.0.0. For information about updates to Babelfish, see [Babelfish versions \(p. 1358\)](#).

Settings that aren't supported

The following settings aren't supported:

- SET ANSI_NULL_DFLT_OFF ON
- SET ANSI_NULL_DFLT_ON OFF
- SET ANSI_PADDING OFF
- SET ANSI_WARNINGS OFF
- SET ARITHABORT OFF
- SET ARITHIGNORE ON
- SET CURSOR_CLOSE_ON_COMMIT ON
- SET NUMERIC_ROUNDABORT ON

Commands for which certain functionality isn't supported

Certain functionality for the following commands isn't supported:

- ADD SIGNATURE

- ALTER DATABASE, ALTER DATABASE SET
- CREATE, ALTER, DROP AUTHORIZATION
- CREATE, ALTER, DROP AVAILABILITY GROUP
- CREATE, ALTER, DROP BROKER PRIORITY
- CREATE, ALTER, DROP COLUMN ENCRYPTION KEY
- CREATE, ALTER, DROP DATABASE ENCRYPTION KEY
- CREATE, ALTER, DROP BACKUP CERTIFICATE
- CREATE AGGREGATE
- CREATE CONTRACT
- GRANT

Column names that aren't supported

The following column names aren't supported:

- \$IDENTITY
- \$ROWGUID
- IDENTITYCOL

Data types that aren't supported

The following data types aren't supported:

- ROWVERSION
- ROWVERSION data type
- TIMESTAMP data type. The SQL Server TIMESTAMP date is unrelated to PostgreSQL TIMESTAMP.

Object types that aren't supported

The following object types aren't supported:

- COLUMN MASTER KEY
- CREATE, ALTER EXTERNAL DATA SOURCE
- CREATE, ALTER, DROP DATABASE AUDIT SPECIFICATION
- CREATE, ALTER, DROP EXTERNAL LIBRARY
- CREATE, ALTER, DROP SERVER AUDIT
- CREATE, ALTER, DROP SERVER AUDIT SPECIFICATION
- CREATE, ALTER, DROP, OPEN/CLOSE SYMMETRIC KEY
- CREATE, DROP DEFAULT
- CREDENTIAL
- CRYPTOGRAPHIC PROVIDER
- DIAGNOSTIC SESSION
- Indexed views
- SERVICE MASTER KEY
- SYNONYM
- USER

Functions that aren't supported

The following functions aren't supported:

- CERTENCODED function
- CERTID function
- CERTPRIVATEKEY function
- CERTPROPERTY function
- COLUMNPROPERTY
- EVENTDATA function
- GET_TRANSMISSION_STATUS
- LOGINPROPERTY function
- OBJECTPROPERTY
- OBJECTPROPERTYEX
- OPENXML
- TYPEPROPERTY function

Syntax for which certain functionality isn't supported

Certain functionality for the following syntax isn't supported:

- ALTER SERVICE, BACKUP/RESTORE SERVICE MASTER KEY clause
- BEGIN DISTRIBUTED TRANSACTION
- CREATE EXTERNAL TABLE
- CREATE TABLE... GRANT clause
- CREATE TABLE... IDENTITY clause
- CREATE, ALTER, DROP APPLICATION ROLE
- CREATE, ALTER, DROP ASSEMBLY
- CREATE, ALTER, DROP ASYMMETRIC KEY
- CREATE, ALTER, DROP EVENT SESSION
- CREATE, ALTER, DROP EXTERNAL RESOURCE POOL
- CREATE, ALTER, DROP FULLTEXT CATALOG
- CREATE, ALTER, DROP FULLTEXT INDEX
- CREATE, ALTER, DROP FULLTEXT STOPLIST
- CREATE, ALTER, DROP QUEUE
- CREATE, ALTER, DROP RESOURCE GOVERNOR
- CREATE, ALTER, DROP ROUTE
- CREATE, ALTER, DROP SERVICE
- CREATE, ALTER, DROP WORKLOAD GROUP
- CREATE, ALTER, DROP, OPEN/CLOSE, BACKUP/RESTORE MASTER KEY
- CREATE/DROP RULE

Syntax that isn't supported

The following syntax isn't supported:

- ALTER DATABASE
- ALTER DATABASE SCOPED CONFIGURATION

- ALTER DATABASE SCOPED CREDENTIAL
- ALTER DATABASE SET HADR
- ALTER FUNCTION
- ALTER INDEX
- ALTER PROCEDURE statement
- ALTER SCHEMA
- ALTER SERVER CONFIGURATION
- ALTER VIEW
- BEGIN CONVERSATION TIMER
- BEGIN DIALOG CONVERSATION
- BULK INSERT
- CREATE COLUMNSTORE INDEX
- CREATE EXTERNAL FILE FORMAT
- CREATE, ALTER, DROP CREDENTIAL
- CREATE, ALTER, DROP CRYPTOGRAPHIC PROVIDER
- CREATE, ALTER, DROP ENDPOINT
- CREATE, ALTER, DROP EXTERNAL LANGUAGE
- CREATE, ALTER, DROP MESSAGE TYPE
- CREATE, ALTER, DROP PARTITION FUNCTION
- CREATE, ALTER, DROP PARTITION SCHEME
- CREATE, ALTER, DROP RESOURCE POOL
- CREATE, ALTER, DROP ROLE
- CREATE, ALTER, DROP SEARCH PROPERTY LIST
- CREATE, ALTER, DROP SECURITY POLICY
- CREATE, ALTER, DROP SELECTIVE XML INDEX clause
- CREATE, ALTER, DROP SPATIAL INDEX
- CREATE, ALTER, DROP TYPE
- CREATE, ALTER, DROP XML INDEX
- CREATE, ALTER, DROP XML SCHEMA COLLECTION
- CREATE, DROP WORKLOAD CLASSIFIER
- CREATE/ALTER/ENABLE/DISABLE TRIGGER
- DENY
- END, MOVE CONVERSATION
- EXECUTE with AS LOGIN or AT option
- GET CONVERSATION GROUP
- GROUP BY ALL clause
- GROUP BY CUBE clause
- GROUP BY ROLLUP clause
- INSERT... DEFAULT VALUES
- INSERT... TOP
- KILL
- MERGE
- NEXT VALUE FOR sequence clause
- READTEXT
- REVERT
- REVOKE

- SELECT PIVOT/UNPIVOT
- SELECT TOP x PERCENT WHERE x <> 100
- SELECT TOP... WITH TIES
- SELECT... FOR XML AUTO
- SELECT... FOR XML EXPLICIT
- SEND
- SET CONTEXT_INFO
- SET DATEFORMAT
- SET DEADLOCK_PRIORITY
- SET FMTONLY
- SET FORCEPLAN
- SET LOCK_TIMEOUT
- SET NUMERIC_ROUNDABORT ON
- SET OFFSETS
- SET REMOTE_PROC_TRANSACTIONS
- SET ROWCOUNT @variable
- SET ROWCOUNT n WHERE n != 0
- SET SHOWPLAN_ALL
- SET SHOWPLAN_TEXT
- SET SHOWPLAN_XML
- SET STATISTICS
- SET STATISTICS IO
- SET STATISTICS PROFILE
- SET STATISTICS TIME
- SET STATISTICS XML
- SET TRANSACTION ISOLATION LEVEL REPEATABLE READ
- SET TRANSACTION ISOLATION LEVEL SERIALIZABLE
- SHUTDOWN statement
- UPDATE STATISTICS
- UPDATETEXT
- Using EXECUTE to call a SQL function
- VIEW... CHECK OPTION clause
- VIEW... VIEW_METADATA clause
- WAITFOR DELAY
- WAITFOR TIME
- WAITFOR, RECEIVE
- WITH XMLNAMESPACES construct
- WRITETEXT
- XPATH expressions

Using Aurora PostgreSQL extensions with Babelfish

Aurora PostgreSQL provides extensions for working with other AWS services. These are optional extensions that support various use cases, such as using Amazon S3 with your DB cluster for importing or exporting data.

- To import data from an Amazon S3 bucket to your Babelfish for Aurora PostgreSQL DB cluster, you set up the `aws_s3` Aurora PostgreSQL extension. This extension also lets you export data from your Aurora PostgreSQL DB cluster to an Amazon S3 bucket.
- AWS Lambda is a compute service that lets you run code without provisioning or managing servers. You can use Lambda functions to do things like process event notifications from your DB instance. To learn more about Lambda, see [What is AWS Lambda?](#) in the *AWS Lambda Developer Guide*. To invoke Lambda functions from your Babelfish for Aurora PostgreSQL DB cluster, you set up the `aws_lambda` Aurora PostgreSQL extension.

To set up these extensions for your Babelfish cluster, you first need to grant permission to the internal Babelfish user to load the extensions. After granting permission, you can then load Aurora PostgreSQL extensions.

Enabling Aurora PostgreSQL extensions in your Babelfish DB cluster

Before you can load the `aws_s3` or the `aws_lambda` extensions, you grant the needed privileges to your Babelfish DB cluster.

The procedure following uses the `psql` PostgreSQL command line tool to connect to the DB cluster. For more information, see [Using psql to connect to the DB cluster \(p. 1316\)](#). You can also use pgAdmin. For details, see [Using pgAdmin to connect to the DB cluster \(p. 1316\)](#).

This procedure loads both `aws_s3` and `aws_lambda`, one after the other. You don't need to load both if you want to use only one of these extensions. The `aws_commons` extension is required by each, and it's loaded by default as shown in the output.

To set up your Babelfish DB cluster with privileges for the Aurora PostgreSQL extensions

1. Connect to your Babelfish for Aurora PostgreSQL DB cluster. Use the name for the "master" user (-U) that you specified when you created the Babelfish DB cluster. The default (`postgres`) is shown in the examples.

For Linux, macOS, or Unix:

```
psql -h your-Babelfish.cluster.44445556666-us-east-1.rds.amazonaws.com \
-U postgres \
-d babelfish_db \
-p 5432
```

For Windows:

```
psql -h your-Babelfish.cluster.44445556666-us-east-1.rds.amazonaws.com ^
-U postgres ^
-d babelfish_db ^
-p 5432
```

The command responds with a prompt to enter the password for the user name (-U).

Password:

Enter the password for the user name (-U) for the DB cluster. When you successfully connect, you see output similar to the following.

```
psql (13.4)
SSL connection (protocol: TLSv1.3, cipher: TLS_AES_256_GCM_SHA384, bits: 256,
compression: off)
Type "help" for help.

postgres=>
```

2. Grant privileges to the internal Babelfish user to create and load extensions.

```
babelfish_db=> GRANT rds_superuser TO master_dbo;
GRANT ROLE
```

3. Create and load the aws_s3 extension. The aws_commons extension is required, and it's installed automatically when the aws_s3 is installed.

```
babelfish_db=> create extension aws_s3 cascade;
NOTICE:  installing required extension "aws_commons"
CREATE EXTENSION
```

4. Create and load the aws_lambda extension.

```
babelfish_db=> create extension aws_lambda cascade;
CREATE EXTENSION
babelfish_db=>
```

Using Babelfish with Amazon S3

If you don't already have an Amazon S3 bucket to use with your Babelfish DB cluster, you can create one. For any Amazon S3 bucket that you want to use, you provide access.

Before trying to import or export data using an Amazon S3 bucket, complete the following one-time steps.

To set up access for your Babelfish DB instance to your Amazon S3 bucket

1. Create an Amazon S3 bucket for your Babelfish instance, if needed. To do so, follow the instructions in [Create a bucket](#) in the *Amazon Simple Storage Service User Guide*.
2. Upload files to your Amazon S3 bucket. To do so, follow the steps in [Add an object to a bucket](#) in the *Amazon Simple Storage Service User Guide*.
3. Set up permissions as needed:
 - To import data from Amazon S3, the Babelfish for Aurora PostgreSQL DB cluster needs permission to access the bucket. We recommend using an AWS Identity and Access Management (IAM) role and attaching an IAM policy to that role for your cluster. To do so, follow the steps in [Using an IAM role to access an Amazon S3 bucket \(p. 1440\)](#).
 - To export data from your Babelfish for Aurora PostgreSQL DB cluster, your cluster must be granted access to the Amazon S3 bucket. As with importing, we recommend using an IAM role and policy. To do so, follow the steps in [Setting up access to an Amazon S3 bucket \(p. 1452\)](#).

You can now use Amazon S3 with the `aws_s3` extension with your Babelfish for Aurora PostgreSQL DB cluster.

To import data from Amazon S3 to Babelfish and to export Babelfish data to Amazon S3

1. Use the `aws_s3` extension with your Babelfish DB cluster.

When you do, make sure to reference the tables as they exist in the context of PostgreSQL. That is, if you want to import into a Babelfish table named `[database].[schema].[tableA]`, refer to that table as `database_schema_tableA` in the `aws_s3` function:

- For an example of using an `aws_s3` function to import data, see [Using the `aws_s3.table_import_from_s3` function to import Amazon S3 data \(p. 1444\)](#).
 - For examples of using `aws_s3` functions to export data, see [Exporting query data using the `aws_s3.query_export_to_s3` function \(p. 1455\)](#).
2. Make sure to reference Babelfish tables using PostgreSQL naming when using the `aws_s3` extension and Amazon S3, as shown in the following table.

Babelfish table	Aurora PostgreSQL table
<code>database.schema.table</code>	<code>database_schema_table</code>

To learn more about using Amazon S3 with Aurora PostgreSQL, see [Importing Amazon S3 data into an Aurora PostgreSQL DB cluster \(p. 1438\)](#) and [Exporting data from an Aurora PostgreSQL DB cluster to Amazon S3 \(p. 1450\)](#).

Using Babelfish with AWS Lambda

After the `aws_lambda` extension is loaded in your Babelfish DB cluster but before you can invoke Lambda functions, you give Lambda access to your DB cluster by following this procedure.

To set up access for your Babelfish DB cluster to work with Lambda

This procedure uses the AWS CLI to create the IAM policy, role, and associate these with the Babelfish DB cluster.

1. Create an IAM policy that allows access to Lambda from your Babelfish DB cluster.

```
aws iam create-policy --policy-name rds-lambda-policy --policy-document '{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "AllowAccessToExampleFunction",  
            "Effect": "Allow",  
            "Action": "lambda:InvokeFunction",  
            "Resource": "arn:aws:lambda:aws-region:444455556666:function:my-function"  
        }  
    ]  
}'
```

2. Create an IAM role that the policy can assume at runtime.

```
aws iam create-role --role-name rds-lambda-role --assume-role-policy-document '{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": "sts:AssumeRole",  
            "Principal": "rds-lambda-role"  
        }  
    ]  
}'
```

```
        "Principal": {  
            "Service": "rds.amazonaws.com"  
        },  
        "Action": "sts:AssumeRole"  
    }  
}  
}'
```

3. Attach the policy to the role.

```
aws iam attach-role-policy \  
    --policy-arn arn:aws:iam::444455556666:policy/rds-lambda-policy \  
    --role-name rds-lambda-role --region aws-region
```

4. Attach the role to your Babelfish for Aurora PostgreSQL DB cluster

```
aws rds add-role-to-db-cluster \  
    --db-cluster-identifier my-cluster-name \  
    --feature-name Lambda \  
    --role-arn arn:aws:iam::444455556666:role/rds-lambda-role \  
    --region aws-region
```

After you complete these tasks, you can invoke your Lambda functions. For more information and examples of setting up AWS Lambda for Aurora PostgreSQL DB cluster with AWS Lambda, see [Step 2: Configure IAM for your Aurora PostgreSQL DB cluster and AWS Lambda \(p. 1518\)](#).

To invoke a Lambda function from your Babelfish DB cluster

AWS Lambda supports functions written in Java, Node.js, Python, Ruby, and other languages. If the function returns text when invoked, you can invoke it from your Babelfish DB cluster. The following example is a placeholder python function that returns a greeting.

```
lambda_function.py  
import json  
def lambda_handler(event, context):  
    #TODO implement  
    return {  
        'statusCode': 200,  
        'body': json.dumps('Hello from Lambda!')}
```

Currently, Babelfish for Aurora PostgreSQL doesn't support JSON. If your function returns JSON, you use a wrapper to handle the JSON. For example, say that the `lambda_function.py` shown preceding is stored in Lambda as `my-function`.

1. Connect to your Babelfish DB cluster using the `psql` client (or the `pgAdmin` client). For more information, see [Using psql to connect to the DB cluster \(p. 1316\)](#).
2. Create the wrapper. This example uses PostgreSQL's procedural language for SQL, `PL/pgSQL`. To learn more, see [PL/pgSQL–SQL Procedural Language](#)

```
create or replace function master_dbo.lambda_wrapper()  
returns text  
language plpgsql  
as  
$$  
declare  
    r_status_code integer;  
    r_payload text;  
begin  
    SELECT payload INTO r_payload
```

```
FROM aws_lambda.invoke( aws_commons.create_lambda_function_arn('my-function',
'us-east-1')
                      , '{"body": "Hello from Postgres!"}':json );
      return r_payload ;
end;
$$;
```

The function can now be run from the Babelfish TDS port (1433) or from the PostgreSQL port (5433).

- To invoke (call) this function from your PostgreSQL port:

```
SELECT * from aws_lambda.invoke(aws_commons.create_lambda_function_arn('my-
function', 'us-east-1'), '{"body": "Hello from Postgres!"}':json );
```

The output is similar to the following:

status_code payload	executed_version log_result
200 {"statusCode": 200, "body": "\"Hello from Lambda!\""} \$LATEST	(1 row)

- To invoke (call) this function from the TDS port, connect to the port using the SQL Server `sqlcmd` command line client. For details, see [Using a SQL Server client to connect to your DB cluster \(p. 1313\)](#). When connected, run the following:

```
1> select lambda_wrapper();
2> go
```

The command returns output similar to the following:

```
{"statusCode": 200, "body": "\"Hello from Lambda!\""}  
-----
```

To learn more about using Lambda with Aurora PostgreSQL, see [Invoking an AWS Lambda function from an Aurora PostgreSQL DB cluster \(p. 1517\)](#). For more information about working with Lambda functions, see [Getting started with Lambda](#) in the *AWS Lambda Developer Guide*.

Managing Babelfish error handling

Babelfish mimics SQL behavior in terms of control flow and transaction state whenever possible. If Babelfish encounters an error, it returns an error code similar to the SQL Server error code if possible. If Babelfish can't map the error, it returns a fixed error code (33557097). If an unmapped error is one of the following, the indicated result happens:

- If it's a compile time error, Babelfish rolls back the transaction.
- If it's a runtime error, Babelfish ends the batch and rolls back the transaction.
- If it's a protocol error between the client and server, the transaction isn't rolled back.

If an error code can't be mapped to an equivalent code and the code for a similar error is available, the error code is mapped to the alternative code. For example, the behaviors that cause SQL Server codes 8143 and 8144 are both mapped to 8143.

Errors that can't be mapped don't respect a `TRY... CATCH` construct.

You can use `@@ERROR` to return a SQL Server error code, or the `@@PGERROR` function to return a PostgreSQL error code. You can also use the `fn_mapped_system_error_list` function to return a list of mapped error codes. For information about PostgreSQL error codes, see [the PostgreSQL website](#).

Babelfish escape hatches

To better deal with statements that might fail, Babelfish defines certain options called escape hatches. An *escape hatch* is an option that specifies Babelfish behavior when it encounters an unsupported feature or syntax.

You can use the `sp_babelfish_configure` stored procedure to control the settings of an escape hatch. Use the script to set the escape hatch to `ignore` or `strict`. If it's set to `strict`, Babelfish returns an error that you need to correct before continuing.

Include the `server` keyword to apply the changes to the current session and on a cluster level.

The usage is as follows:

- To list all escape hatches and their status, plus usage information, run `sp_babelfish_configure`.
- To list the named escape hatches and their values, for the current session or cluster-wide, run the command `sp_babelfish_configure 'hatch_name'` where `hatch_name` is the identifier of one or more escape hatches. `hatch_name` can use SQL wildcards, such as '%'.
- To set one or more escape hatches to the value specified, run `sp_babelfish_configure ['hatch_name' [, 'strict' | 'ignore' [, 'server']]]`. To make the settings permanent on a cluster-wide level, include the `server` keyword. To set them for the current session only, don't use `server`.

The string identifying the hatch (or hatches) might contain SQL wildcards. For example, the following sets all syntax escape hatches to `ignore` for the Aurora PostgreSQL cluster.

```
EXECUTE sp_babelfish_configure '%', 'ignore', 'server'
```

The Babelfish predefined escape hatches are as follows.

Escape hatch	Description	Default
<code>escape_hatch_storage_options</code>	Escape hatch on any storage option used in CREATE,	<code>ignore</code>

Escape hatch	Description	Default
	ALTER DATABASE, TABLE, INDEX. This includes clauses (LOG) ON, TEXTIMAGE_ON, FILESTREAM_ON that define storage locations (partitions, file groups) for tables, indexes, and constraints, and also for a database. This escape hatch setting applies to all of these clauses (including ON [PRIMARY] and ON "DEFAULT"). The exception is when a partition is specified for a table or index with ON partition_scheme (column).	
<code>escape_hatch_storage_on_partition</code>	Controls Babelfish behavior related to the <code>ON partition_scheme</code> column clause when defining partitioning. Babelfish currently doesn't implement partitioning.	strict
<code>escape_hatch_database_misc_options</code>	Controls Babelfish behavior related to the following options on CREATE or ALTER DATABASE: CONTAINMENT, DB_CHAINING, TRUSTWORTHY, PERSISTENT_LOG_BUFFER.	ignore
<code>escape_hatch_language_non_english</code>	Controls Babelfish behavior related to languages other than English for onscreen messages. Babelfish currently supports only <code>us_english</code> for onscreen messages. SET LANGUAGE might use a variable containing the language name, so the actual language being set can only be detected at run time.	strict
<code>escape_hatch_login_hashed_password</code>	When ignored, suppresses the error for the <code>HASHED</code> keyword for <code>CREATE LOGIN</code> and <code>ALTER LOGIN</code> .	strict
<code>escape_hatch_login_misc_options</code>	When ignored, suppresses the error for other keywords besides <code>HASHED</code> , <code>MUST_CHANGE</code> , <code>OLD_PASSWORD</code> , and <code>UNLOCK</code> for <code>CREATE LOGIN</code> and <code>ALTER LOGIN</code> .	strict

Escape hatch	Description	Default
<code>escape_hatch_login_old_password</code>	When ignored, suppresses the error for the <code>OLD_PASSWORD</code> keyword for <code>CREATE LOGIN</code> and <code>ALTER LOGIN</code> .	strict
<code>escape_hatch_login_password_must_change</code>	When ignored, suppresses the error for the <code>MUST_CHANGE</code> keyword for <code>CREATE LOGIN</code> and <code>ALTER LOGIN</code> .	strict
<code>escape_hatch_login_password_unlock</code>	When ignored, suppresses the error for the <code>UNLOCK</code> keyword for <code>CREATE LOGIN</code> and <code>ALTER LOGIN</code> .	strict
<code>escape_hatch_fulltext</code>	Controls Babelfish behavior related to FULLTEXT features, such a <code>DEFAULT_FULLTEXT_LANGUAGE</code> in <code>CREATE/ALTER DATABASE</code> , <code>CREATE FULLTEXT INDEX</code> , or <code>sp_fulltext_database</code> .	strict
<code>escape_hatch_schemabinding_function</code>	Controls Babelfish behavior related to the <code>WITH SCHEMABINDING</code> clause. By default, the <code>WITH SCHEMABINDING</code> clause is ignored when specified with the <code>CREATE</code> or <code>ALTER FUNCTION</code> command.	ignore
<code>escape_hatch_schemabinding_procedure</code>	Controls Babelfish behavior related to the <code>WITH SCHEMABINDING</code> clause. By default, the <code>WITH SCHEMABINDING</code> clause is ignored when specified with the <code>CREATE</code> or <code>ALTER PROCEDURE</code> command.	ignore
<code>escape_hatch_schemabinding_view</code>	Controls Babelfish behavior related to the <code>WITH SCHEMABINDING</code> clause. By default, the <code>WITH SCHEMABINDING</code> clause is ignored when specified with the <code>CREATE</code> or <code>ALTER VIEW</code> command.	ignore

Escape hatch	Description	Default
<code>escape_hatch_schemabinding_trigger</code>	Controls Babelfish behavior related to the WITH SCHEMABINDING clause. By default, the WITH SCHEMABINDING clause is ignored when specified with the CREATE or ALTER TRIGGER command.	ignore
<code>escape_hatch_index_clustering</code>	Controls Babelfish behavior related to the CLUSTERED or NONCLUSTERED keywords for indexes and PRIMARY KEY or UNIQUE constraints. When CLUSTERED is ignored, the index or constraint is still created as if NONCLUSTERED was specified.	ignore
<code>escape_hatch_index_columnstore</code>	Controls Babelfish behavior related to the COLUMNSTORE clause. If you specify ignore, Babelfish creates a regular B-tree index.	strict
<code>escape_hatch_for_replication</code>	Controls Babelfish behavior related to the [NOT] FOR REPLICATION clause when creating or altering a table.	strict
<code>escape_hatch_for_replication</code>	Controls Babelfish behavior related to the [NOT] FOR REPLICATION clause when creating or altering a procedure.	strict
<code>escape_hatch_rowguidcol_column</code>	Controls Babelfish behavior related to the ROWGUIDCOL clause when creating or altering a table.	strict
<code>escape_hatch_nocheck_add_constraint</code>	Controls Babelfish behavior related to the WITH CHECK or NOCHECK clause for constraints.	strict
<code>escape_hatch_nocheck_existing_constraint</code>	Controls Babelfish behavior related to FOREIGN KEY or CHECK constraints.	strict
<code>escape_hatch_constraint_name_for_default</code>	Controls Babelfish behavior related to default constraint names.	ignore
<code>escape_hatch_table_hints</code>	Controls the behavior of table hints specified using the WITH (...) clause.	ignore

Escape hatch	Description	Default
<code>escape_hatch_query_hints</code>	Controls Babelfish behavior related to query hints. When this option is set to ignore, the server ignores hints that use the OPTION (...) clause to specify query processing aspects. Examples include <code>SELECT FROM ... OPTION(MERGE JOIN HASH, MAXRECURSION 10)</code> .	ignore
<code>escape_hatch_join_hints</code>	Controls the behavior of keywords in a JOIN operator: LOOP, HASH, MERGE, REMOTE, REDUCE, REDISTRIBUTE, REPLICATE.	ignore
<code>escape_hatch_session_settings</code>	Controls Babelfish behavior toward unsupported session-level SET statements.	ignore
<code>escape_hatch_unique_constraint</code>	Controls Babelfish behavior when creating a unique index or constraint on a nullable column.	strict

Configuring a database for Babelfish

When you create a Babelfish for Aurora PostgreSQL DB cluster, you can use a parameter group in one of two ways. You can create a new parameter group that configures a cluster with Babelfish running. Or you can use a pre-existing Amazon Aurora parameter group.

To use an existing parameter group, edit the group and set the `babelfish_status` parameter to `on`. Specify any Babelfish options before creating your Aurora PostgreSQL cluster. For information about modifying your parameter group, see [Working with DB parameter groups and DB cluster parameter groups \(p. 339\)](#).

The following parameters control Babelfish preferences.

Parameter	Type	Default value	Values allowed	Description	Modifiable?
<code>rds.babelfish_status</code>	String	off	on, off, datatypesonly	Sets the state of Babelfish for Aurora PostgreSQL; functionality. When this parameter is set to <code>datatypesonly</code> , Babelfish is turned off but SQL Server data types are still available.	Yes
<code>babelfishpg_tds.default_server_name</code>	String	Microsoft SQL Server		The default name of the Babelfish server.	Yes
<code>babelfishpg_tds.port</code>	Integer	1433	1-65535	Sets the TCP port used for requests in SQL Server syntax.	Yes
<code>babelfishpg_tds.tds_default_protocol_version</code>	String	0	TDSv7.0, TDSv7.1, TDSv7.1.1, TDSv7.2, TDSv7.3A, TDSv7.3B, TDSv7.4, DEFAULT	Sets a default TDS protocol version for connecting clients.	Yes
<code>babelfishpg_tds.tds_ssl_encrypt</code>	Boolean	0	0/1	Turns encryption on (0) or off (1) for data traversing the TDS listener port. For detailed information about using SSL for client connections, see How Babelfish interprets SSL settings (p. 1348) .	Yes
<code>babelfishpg_tds.tds_ssl_max_protocol_version</code>	String	0	"TLSv1, TLSv1.1, TLSv1.2"	Sets the minimum SSL/TLS protocol version to use for the TDS session.	Yes

Parameter	Type	Default value	Values allowed	Description	Modifiable?
babelfishpg_tds.tds_ssl_min_protocol	String	'TLSv1'	'TLSv1, TLSv1.1, TLSv1.2'	Sets the minimum SSL/TLS protocol version to use for the TDS session.	Yes
babelfishpg_tds.tds_default_packet_size	Integer	4096	512-32767	Sets the default packet size for connecting SQL Server clients.	Yes
babelfishpg_tds.tds_default_numeric_scale	Integer	8	0-38	Sets the default scale of numeric type to be sent in the TDS column metadata if the engine doesn't specify one.	Yes
babelfishpg_tds.tds_default_numeric_precision	Integer	38	1-38	Sets the default precision of numeric type to be sent in the TDS column metadata if the engine doesn't specify one.	Yes
babelfishpg_tsql.version	String	null	default	Sets the output of @@VERSION variable. Don't modify this value for Aurora PostgreSQL DB clusters.	Yes
babelfishpg_tsql.default_locale	String	en_US	Allowed	Default locale used for Babelfish collations. The default locale is only the locale and doesn't include any qualifiers. Set this parameter when you provision a Babelfish DB cluster. After the DB cluster is provisioned, modifications to this parameter are ignored.	Yes
babelfishpg_tsql.migration_mode	List	single-db	single-db, multi-db, null	Defines if multiple user databases are supported. Set this parameter when you provision a Babelfish DB cluster. After the DB cluster is provisioned, don't modify the value of this parameter.	No

Parameter	Type	Default value	Values allowed	Description	Modifiable?
babelfishpg_tsql.server_collation_name	String	bbf_unicode_ci	Babelfish_general_ci_and_Babelfish_general_ci_as	The name of the collation used for server-level actions. Set once at provisioning time. Set this parameter when you provision a Babelfish DB cluster. After the DB cluster is provisioned, don't modify the value of this parameter.	Yes
babelfishpg_tds.listen_addresses	String	*	null	Sets the host name or IP address or addresses to listen for TDS on.	No
babelfishpg_tds.enable_tds_debug_level	Integer	'0, 1, 2, 3'	'0, 1, 2, 3'	Sets the logging level in TDS; 0 turns off logging.	Yes
babelfishpg_tds.unix_socket_directories	String	/tmp	NULL	TDS server Unix socket directories.	No
babelfishpg_tds.unix_socket_group	String	rdsdb	NULL	TDS server Unix socket group.	No
unix_socket_permissions	Integer	0700	0 - 511	TDS server Unix socket permissions.	No

How Babelfish interprets SSL settings

When a client connects to port 1433, Babelfish compares the Secure Sockets Layer (SSL) setting sent during the client handshake to the Babelfish SSL parameter setting (`tds_ssl_encrypt`). Babelfish then determines if a connection is allowed. If a connection is allowed, encryption behavior is either enforced or not, depending on your parameter settings and the support for encryption offered by the client.

The table following shows how Babelfish behaves for each combination.

Client SSL setting	Babelfish SSL setting	Connection allowed?	Value returned to client
ENCRYPT_OFF	<code>tds_ssl_encrypt=false</code>	Allowed, the login packet is encrypted	ENCRYPT_OFF
ENCRYPT_OFF	<code>tds_ssl_encrypt=true</code>	Allowed, the entire connection is encrypted	ENCRYPT_REQ
ENCRYPT_ON	<code>tds_ssl_encrypt=false</code>	Allowed, the entire connection is encrypted	ENCRYPT_ON
ENCRYPT_ON	<code>tds_ssl_encrypt=true</code>	Allowed, the entire connection is encrypted	ENCRYPT_ON
ENCRYPT_NOT_SUP	<code>tds_ssl_encrypt=false</code>	Yes	ENCRYPT_NOT_SUP
ENCRYPT_NOT_SUP	<code>tds_ssl_encrypt=true</code>	No, connection closed	ENCRYPT_REQ
ENCRYPT_REQ	<code>tds_ssl_encrypt=false</code>	Allowed, the entire connection is encrypted	ENCRYPT_ON
ENCRYPT_REQ	<code>tds_ssl_encrypt=true</code>	Allowed, the entire connection is encrypted	ENCRYPT_ON
ENCRYPT_CLIENT_CERT	<code>tds_ssl_encrypt=false</code>	No, connection closed	Unsupported
ENCRYPT_CLIENT_CERT	<code>tds_ssl_encrypt=true</code>	No, connection closed	Unsupported

Babelfish collation support

A *collation* specifies the sort order and presentation format of data. Babelfish maps SQL Server collations to comparable collations provided by Babelfish. Babelfish predefines Unicode collations with culturally sensitive string comparisons and sort orders. Babelfish also provides a way to translate the collations in your SQL Server DB to the closest-matching Babelfish collation. Locale-specific collations are provided for different languages and regions.

Some collations specify a code page that corresponds to a client-side encoding. Babelfish automatically translates from the server encoding to the client encoding depending on the collation of each output column.

Babelfish uses version 153.80 of the ICU collation library. For detailed information about PostgreSQL collation behavior, see [the PostgreSQL documentation](#).

Babelfish supports deterministic and nondeterministic collations:

- A *deterministic collation* considers two characters as equal if they have the exact same byte sequence. A deterministic collation evaluates `x` and `X` as not equal. Collations that are deterministic are case-sensitive (CS) and accent-sensitive (AS).
- A *nondeterministic collation* doesn't require an identical match. A nondeterministic collation evaluates `x` and `X` as equal. Nondeterministic collations are case-insensitive (CI) and accent-insensitive (AI).

Babelfish and SQL Server follow a naming convention for collations that describe the collation attributes, as shown in the table following.

Attribute	Description
AI	Accent-insensitive.
AS	Accent-sensitive.
BIN2	BIN2 requests data to be sorted in code point order. Unicode code point order is the same character order for UTF-8, UTF-16, and UCS-2 encodings. Code point order is a fast deterministic collation.
CI	Case-insensitive.
CS	Case-sensitive.
PREF	To sort uppercase letters before lowercase letters, use a PREF collation. If comparison is case-insensitive, the uppercase version of a letter sorts before the lowercase version, if there is no other distinction. The ICU library supports uppercase preference with <code>colCaseFirst=upper</code> , but not for CI_AS collations. PREF can be applied only to CS_AS deterministic collations.

PostgreSQL doesn't support the `LIKE` clause on nondeterministic collations, but Babelfish supports it for CI_AS collations. Babelfish doesn't support the `LIKE` clause on AI collations. Pattern matching operations on nondeterministic collations also aren't supported.

To establish Babelfish collation behavior, set the following parameters.

Parameter	Description
<code>default_locale</code>	The <code>default_locale</code> parameter is used in combination with the collation attributes in the table preceding to customize

Parameter	Description
	<p>collations for a specific language and region. The default value is en-US.</p> <p>The default locale applies to all Babelfish collations that start with the letters BBF, and to all SQL Server collations that are mapped to Babelfish collations. You can change this parameter after initial Babelfish database creation, but it doesn't affect the locale of existing collations.</p>
<code>server_collation_name</code>	<p>The collation used as the default collation at both the server level and the database level. The default value is <code>sql_latin1_general_cp1_ci_as</code>. The <code>server_collation_name</code> has to be a CI_AS collation because in T-SQL, the server collation determines how identifiers are compared.</p> <p>You can choose from the collations in the table that follows for the <code>Collation_name</code> field when you create your Aurora PostgreSQL cluster for use with Babelfish. Don't modify the <code>server_collation_name</code> after the Babelfish database is created.</p>

Use the following collations as a server collation or an object collation.

Collation ID	Notes
<code>BBF_Uncode_General_CI_AS</code>	Supports case-insensitive comparison and the LIKE operator.
<code>BBF_Uncode_CP1_CI_AS</code>	Nondeterministic collation also known as CP1252.
<code>BBF_Uncode_CP1250_CI_AS</code>	Nondeterministic collation used to represent texts in Central European and Eastern European languages that use Latin script.
<code>BBF_Uncode_CP1251_CI_AS</code>	Nondeterministic collation for languages that use the Cyrillic script.
<code>BBF_Uncode_CP1253_CI_AS</code>	Nondeterministic collation used to represent modern Greek.
<code>BBF_Uncode_CP1254_CI_AS</code>	Nondeterministic collation that supports Turkish.
<code>BBF_Uncode_CP1255_CI_AS</code>	Nondeterministic collation that supports Hebrew.
<code>BBF_Uncode_CP1256_CI_AS</code>	Nondeterministic collation used to write languages that use Arabic script.
<code>BBF_Uncode_CP1257_CI_AS</code>	Nondeterministic collation used to support Estonian, Latvian, and Lithuanian languages.
<code>BBF_Uncode_CP1258_CI_AS</code>	Nondeterministic collation used to write Vietnamese characters.
<code>BBF_Uncode_CP874_CI_AS</code>	Nondeterministic collation used to write Thai characters.
<code>sql_latin1_general_cp1250_ci_as</code>	Nondeterministic single-byte character encoding used to represent Latin characters.
<code>sql_latin1_general_cp1251_ci_as</code>	Nondeterministic collation that supports Latin characters.
<code>sql_latin1_general_cp1_ci_as</code>	Nondeterministic collation that supports Latin characters.

Collation ID	Notes
sql_latin1_general_cp1253_ci_as	Nondeterministic collation that supports Latin characters.
sql_latin1_general_cp1254_ci_as	Nondeterministic collation that supports Latin characters.
sql_latin1_general_cp1255_ci_as	Nondeterministic collation that supports Latin characters.
sql_latin1_general_cp1256_ci_as	Nondeterministic collation that supports Latin characters.
sql_latin1_general_cp1257_ci_as	Nondeterministic collation that supports Latin characters.
sql_latin1_general_cp1258_ci_as	Nondeterministic collation that supports Latin characters.
chinese_prc_ci_as	Nondeterministic collation that supports Chinese (PRC).
cyrillic_general_ci_as	Nondeterministic collation that supports Cyrillic.
finnish_swedish_ci_as	Nondeterministic collation that supports Finnish.
french_ci_as	Nondeterministic collation that supports French.
korean_wansung_ci_as	Nondeterministic collation that supports Korean (with dictionary sort).
latin1_general_ci_as	Nondeterministic collation that supports Latin characters.
modern_spanish_ci_as	Nondeterministic collation that supports Modern Spanish.
polish_ci_as	Nondeterministic collation that supports Polish.
thai_ci_as	Nondeterministic collation that supports Thai.
traditional_spanish_ci_as	Nondeterministic collation that supports Spanish (traditional sort).
turkish_ci_as	Nondeterministic collation that supports Turkish.
ukrainian_ci_as	Nondeterministic collation that supports Ukrainian.
vietnamese_ci_as	Nondeterministic collation that supports Vietnamese.

You can use the following collations as object collations.

Dialect	Deterministic options	Nondeterministic options
Arabic	Arabic_CS_AS	Arabic_CI_AS, Arabic_CI_AI
Chinese	Chinese_CS_AS	Chinese_CI_AS, Chinese_CI_AI
Cyrillic_General	Cyrillic_General_CS_AS	Cyrillic_General_CI_AS, Cyrillic_General_CI_AI
Estonian	Estonian_CS_AS	Estonian_CI_AS, Estonian_CI_AI
Finnish_Swedish	Finnish_Swedish_CS_AS	Finnish_Swedish_CI_AS, Finnish_Swedish_CI_AI
French	French_CS_AS	French_CI_AS, French_CI_AI
Greek	Greek_CS_AS	Greek_CI_AS, Greek_CI_AI
Hebrew	Hebrew_CS_AS	Hebrew_CI_AS, Hebrew_CI_AI

Dialect	Deterministic options	Nondeterministic options
Korean_Wamsung	Korean_Wamsung_CS_AS	Korean_Wamsung_CI_AS, Korean_Wamsung_CI_AI
Modern_Spanish	Modern_Spanish_CS_AS	Modern_Spanish_CI_AS, Modern_Spanish_CI_AI
Mongolian	Mongolian_CS_AS	Mongolian_CI_AS, Mongolian_CI_AI
Polish	Polish_CS_AS	Polish_CI_AS, Polish_CI_AI
Thai	Thai_CS_AS	Thai_CI_AS, Thai_CI_AI
Traditional_Spanish	Traditional_Spanish_CS_AS	Traditional_Spanish_CI_AS, Traditional_Spanish_CI_AI
Turkish	Turkish_CS_AS	Turkish_CI_AS, Turkish_CI_AI
Ukrainian	Ukrainian_CS_AS	Ukrainian_CI_AS, Ukrainian_CI_AI
Vietnamese	Vietnamese_CS_AS	Vietnamese_CI_AS, Vietnamese_CI_AI

Managing collations

The ICU library provides collation version tracking to ensure that indexes that depend on collations can be reindexed when a new version of ICU becomes available. You can use the following query to identify all collations in the current database that need to be refreshed and the objects that depend on them.

```
SELECT pg_describe_object(refclassid, refobjid, refobjsubid) AS "Collation",
       pg_describe_object(classid, objid, objsubid) AS "Object" FROM pg_depend d JOIN
       pg_collation c ON refclassid = 'pg_collation'::regclass AND refobjid = c.oid WHERE
       c.colversion < > pg_collation_actual_version(c.oid) ORDER BY 1, 2;
```

Predefined collations are stored in the `sys.fn_helpcollations` table. You can use the following command to display information about a collation (such as its `lcid`, `style`, and `collate` flags). To retrieve the list, connect a psql client to the Aurora PostgreSQL port (by default, 5432) and enter the following.

```
postgres=# set search_path = public, pg_temp, sys;
SET
postgres=# \dO
```

Connect to the T-SQL port (by default 1433) and enter the following.

```
SELECT * FROM fn_helpcollation()
```

Collation limitations and behaviors

Babelfish uses the ICU library for collation support. The following section lists some of the known limitations and behavior variations of Babelfish collations:

- **Unicode sorting rules**

SQL Server SQL collations sort Unicode-encoded data (`nchar` and `nvarchar`) one way, but data that isn't Unicode-encoded (`char` and `varchar`) a different way. Babelfish databases are always UTF-8 encoded and always apply Unicode sorting rules consistently, regardless of data type. Thus, the sort order is the same for `char` or `varchar` as it is for `nchar` or `nvarchar`.

- **Secondary-equal collations**

The default ICU Unicode secondary-equal (CI_AS) collation sorts punctuation marks and other nonalphanumeric characters before numeric characters, and numeric characters before alphabetic characters. However, the order of punctuation and other special characters is different.

- **Tertiary collations**

SQL collations, such as SQL_Latin1_General_Pref_CI_AS, support the TERTIARY_WEIGHTS function and the ability to sort strings that compare equally in a CI_AS collation to be sorted uppercase first: ABC, ABc, AbC, Abc, aBC, aBc, abC, and finally abc. Thus, the DENSE_RANK OVER (ORDER BY column) analytic function assesses these strings as having the same rank but orders them uppercase first within a partition.

You can get a similar result with Babelfish by adding a COLLATE clause to the ORDER BY clause that specifies a tertiary CS_AS collation that specifies @colCaseFirst=upper. However, the colCaseFirst modifier applies only to strings that are tertiary-equal (rather than secondary-equal like a CI_AS collation). Thus, you can't emulate tertiary SQL collations using a single ICU collation.

As a workaround, we recommend that you modify applications that use the SQL_Latin1_General_Pref_CI_AS collation to use the BBF_SQL_Latin1_General_CI_AS collation first. Then add COLLATE BBF_SQL_Latin1_General_Pref_CI_CS_AS to any ORDER BY clause for this column.

- PostgreSQL is built with a specific version of ICU and can match at most one version of a collation. Variations across versions are unavoidable, as are minor variations across time as languages evolve.

- **Character expansion**

A character expansion treats a single character as equal to a sequence of characters at the primary level. SQL Server's default CI_AS collation supports character expansion; ICU collations support character expansion only for accent-insensitive collations.

When character expansion is required, then use a AI collation for comparisons. However, such collations aren't currently supported by the LIKE operator.

- **char and varchar encoding**

When collations that begin with SQL are used for char or varchar data types, the sort order for characters preceding ASCII 127 is determined by the specific code page for that SQL collation. For SQL collations, strings declared as char or varchar might sort differently than strings declared as nchar or nvarchar.

PostgreSQL encodes all strings with the database encoding so converts all characters to UTF-8 and sorts using Unicode rules.

Because SQL collations sort nchar and nvarchar data types using Unicode rules, Babelfish encodes all strings on the server using UTF-8. Babelfish sorts nchar and nvarchar strings the same way it sorts char and varchar strings, using Unicode rules.

- **Supplementary character**

The SQL Server functions NCHAR, UNICODE, and LEN support characters for code-points outside the Unicode Basic Multilingual Plane (BMP). In contrast, non-SC collations use surrogate pair characters to handle supplementary characters. For Unicode data types, SQL Server can represent up to 65,535 characters using UCS-2, or the full Unicode range (1,114,114 characters) if supplementary characters are used.

- **Kana-sensitive**

When Japanese Kana characters Hiragana and Katakana are treated differently, the collation is called Kana-sensitive (KS). ICU supports the Japanese collation standard JIS X 4061. The now deprecated colhiraganaQ [on | off] locale modifier might provide the same functionality as

KS collations. However, KS collations of the same name as SQL Server aren't currently supported by Babelfish.

- Width-Sensitive

When a single-byte character (half-width) and the same character represented as a double-byte character (full-width) are treated differently, the collation is called *width-sensitive* (WS). WS collations with the same name as SQL Server aren't currently supported by Babelfish.

- Variation-Selector Sensitivity

Variation-Selector Sensitivity (VSS) collations distinguish between ideographic variation selectors in Japanese collations `Japanese_Bushu_Kakusu_140` and `Japanese_XJIS_140`. A variation sequence is made up of a base character plus an additional variation selector. If you don't select the `_vss` option, the variation selector isn't considered in the comparison.

VSS collations aren't currently supported by Babelfish.

- BIN and BIN2

A BIN2 collation sorts characters according to code point order. The byte-by-byte binary order of UTF-8 preserves Unicode code point order, so this is also likely to be the best-performing collation. If Unicode code point order works for an application, consider using a BIN2 collation. However, using a BIN2 collation can result in data being displayed on the client in an order that is culturally unexpected. New mappings to lowercase characters are added to Unicode as time progresses, so the `LOWER` function might perform differently on different versions of ICU. This is a special case of the more general collation versioning problem rather than as something specific to the BIN2 collation.

Babelfish provides the `BBF_Latin1_General_BIN2` collation with the Babelfish distribution to collate in Unicode code point order. In a BIN collation only the first character is sorted as a wchar. Remaining characters are sorted byte-by-byte, effectively in code point order according to its encoding. This approach doesn't follow Unicode collation rules and isn't supported by Babelfish.

Troubleshooting for Babelfish

Following, you can find troubleshooting ideas and workarounds for some Babelfish for Aurora PostgreSQL DB cluster issues.

Topics

- [Connection failure \(p. 1355\)](#)
- [Using pg_dump and pg_restore requires extra setup \(p. 1355\)](#)

Connection failure

Common causes of connection failures to a new Aurora DB cluster with Babelfish include the following:

- **Security group doesn't allow access** – If you're having trouble connecting to a Babelfish, make sure that you added your IP address to the default Amazon EC2 security group. You can use <https://checkip.amazonaws.com/> to determine your IP address and then add it to your in-bound rule for the TDS port and the PostgreSQL port. For more information, see [Add rules to a security group](#) in the [Amazon EC2 User Guide](#).

For more information about troubleshooting Aurora connection issues, see [Can't connect to Amazon RDS DB instance \(p. 1814\)](#).

Using pg_dump and pg_restore requires extra setup

Currently, if you try to use the PostgreSQL utilities `pg_dump` and `pg_restore` to move a database from one Babelfish for Aurora PostgreSQL DB cluster to another, you see the following error message:

```
psql:bbf.sql:29: ERROR:  role "db_owner" does not exist
psql:bbf.sql:49: ERROR:  role "dbo" does not exist
```

To workaround this issue, you first create the same logical database on the target cluster that is on the source. Once that exists, you can create the needed roles to run `pg_dump` and `pg_restore`.

To use pg_dump and pg_restore to move a database between Babelfish DB clusters

1. Use `psql` or `pgAdmin` to connect to the target Babelfish for Aurora PostgreSQL DB cluster. The following examples use `psql`. For more information, see [Using psql to connect to the DB cluster \(p. 1316\)](#).
2. Create the same logical database on the target that is on the source.

```
CREATE DATABASE your-DB-name
```

3. Connect to the Babelfish DB instance and create the necessary roles.

```
CREATE ROLE db_owner;
ALTER ROLE db_owner WITH NOSUPERUSER INHERIT NOCREATEROLE NOCREATEDB NOLOGIN
    NOREPLICATION NOBYPASSRLS;
CREATE ROLE dbo;
ALTER ROLE dbo WITH NOSUPERUSER INHERIT NOCREATEROLE NOCREATEDB NOLOGIN NOREPLICATION
    NOBYPASSRLS;
GRANT db_owner TO dbo GRANTED BY sysadmin;
GRANT dbo TO sysadmin GRANTED BY sysadmin;
```

4. Use `pg_restore` to restore the DB instance from the source to the target.

To learn more about using these PostgreSQL utilities, see [pg_dump](#) and [pg_restore](#).

Turning off Babelfish

When you no longer need Babelfish, you can turn off Babelfish functionality.

Be aware of some considerations:

- In some cases, you might turn off Babelfish before completing a migration to Aurora PostgreSQL. If you do and your DDL depends on SQL Server data types or you use any T-SQL functionality in your code, your code fails.
- If after provisioning a Babelfish instance you turn off the Babelfish extension, you can't provision that same database again on the same cluster.

To turn off Babelfish, modify your parameter group, setting `rds.babelfish_status` to `OFF`. You can continue to use your SQL Server data types with Babelfish off, by setting `rds.babelfish_status` to `datatypeonly`.

If you turn off Babelfish in parameter group, all clusters that use that parameter group lose Babelfish functionality.

For more information about modifying parameter groups, see [Working with DB parameter groups and DB cluster parameter groups \(p. 339\)](#).

Babelfish versions

Following, you can find information about Babelfish for Aurora PostgreSQL updates. Babelfish is an option available with Aurora PostgreSQL version 13.4 and higher releases. Updates to Babelfish become available with certain new releases of the Aurora PostgreSQL database engine, as shown in the following table.

Aurora PostgreSQL version	Babelfish version
Aurora PostgreSQL version 13.5	Babelfish version 1.1.0
Aurora PostgreSQL version 13.4	Babelfish version 1.0.0

If your Aurora PostgreSQL has automatic minor version upgrades (AMVU) turned on, the cluster is upgraded for you during your specified maintenance window. By using this approach, you get the new Babelfish release automatically. To learn more about AMVU, see [Automatic minor version upgrades for PostgreSQL \(p. 1688\)](#).

If your Aurora PostgreSQL DB cluster isn't set up for AMVU, you can upgrade manually. For more information about upgrading an Aurora PostgreSQL DB cluster, see [Upgrading the PostgreSQL DB engine for Aurora PostgreSQL \(p. 1681\)](#).

For information about how to obtain version details for Aurora PostgreSQL or Babelfish, see [Querying Babelfish to find Babelfish details \(p. 1319\)](#).

Topics

- [Babelfish version 1.1.0 \(p. 1358\)](#)
- [Babelfish version 1.0.0 \(p. 1359\)](#)

Babelfish version 1.1.0

This release of Babelfish for Aurora PostgreSQL adds support for the following Microsoft SQL Server functionality and T-SQL commands:

- Support for creation of unique indexes or UNIQUE constraints on nullable columns. To use this capability, you change the escape_hatch_unique_constraint to 'ignore'. For more information, see [Babelfish escape hatches \(p. 1340\)](#).
- Support for referencing transition tables from triggers with multiple DML actions.
- Support for identifiers that have leading dot characters.
- Support for the COLUMNPROPERTY function (limited to CharMaxLen and AllowsNull properties).
- Support for the following system-defined @@ variables:
 - @@CURSOR_ROWS
 - @@LOCK_TIMEOUT
 - @@MAX_CONNECTIONS
 - @@MICROSOFTVERSION
 - @@NESTLEVEL
 - @@PROCID
- Support for the following built-in functions:
 - CHOOSE
 - CONCAT_WS
 - CURSOR_STATUS

- DATEFROMPARTS
- DATETIMEFROMPARTS
- ORIGINAL_LOGIN
- SCHEMA_NAME (now fully supported)
- SESSION_USER
- SQUARE
- TRIGGER_NESTLEVEL supported (but only without arguments)
- Support for the following system stored procedures:
 - sp_columns
 - sp_columns_100
 - sp_columns_managed
 - sp_cursor
 - sp_cursor_list
 - sp_cursorclose
 - sp_cursorexecute
 - sp_cursorfetch
 - sp_cursoropen
 - sp_cursoroption
 - sp_cursorprepare
 - sp_cursorprepexec
 - sp_cursorunprepare
 - sp_databases
 - sp_datatype_info
 - sp_datatype_info_100
 - sp_describe_cursor
 - sp_describe_first_result_set
 - sp_describe_undeclared_parameters
 - sp_msdb_ro_username
 - sp_pkeys
 - sp_prepare
 - sp_statistics
 - sp_statistics_100
 - sp_tablecollations_100
 - sp_tables
 - sp_unprepare

For more information, see the [Babelfish for PostgreSQL](#) documentation.

Babelfish version 1.0.0

Version 1.0.0 is the initial release of Babelfish for Aurora PostgreSQL.

Managing Amazon Aurora PostgreSQL

The following section discusses managing performance and scaling for an Amazon Aurora PostgreSQL DB cluster. It also includes information about other maintenance tasks.

Topics

- [Scaling Aurora PostgreSQL DB instances \(p. 1360\)](#)
- [Maximum connections to an Aurora PostgreSQL DB instance \(p. 1360\)](#)
- [Temporary storage limits for Aurora PostgreSQL \(p. 1362\)](#)
- [Testing Amazon Aurora PostgreSQL by using fault injection queries \(p. 1364\)](#)
- [Displaying volume status for an Aurora PostgreSQL DB cluster \(p. 1367\)](#)
- [Specifying the RAM disk for the stats_temp_directory \(p. 1368\)](#)
- [Scheduling maintenance with the PostgreSQL pg_cron extension \(p. 1370\)](#)

Scaling Aurora PostgreSQL DB instances

You can scale Aurora PostgreSQL DB instances in two ways, instance scaling and read scaling. For more information about read scaling, see [Read scaling \(p. 400\)](#).

You can scale your Aurora PostgreSQL DB cluster by modifying the DB instance class for each DB instance in the DB cluster. Aurora PostgreSQL supports several DB instance classes optimized for Aurora. Don't use db.t2 or db.t3 instance classes for larger Aurora clusters of size greater than 40 terabytes (TB).

Scaling isn't instantaneous. It can take 15 minutes or more to complete the change to a different DB instance class. We recommend that if you use this approach to modify the DB instance class, you apply the change during the next scheduled maintenance window (rather than immediately) to avoid affecting users.

As an alternative to modifying the DB instance class directly, you can minimize downtime by using the high availability features of Amazon Aurora. First, add an Aurora Replica to your cluster. When creating the replica, choose the DB instance class size that you want to use for your cluster. When the Aurora Replica is synchronized with the cluster, you then failover to the newly added Replica. To learn more, see [Aurora Replicas \(p. 70\)](#) and [Fast failover with Amazon Aurora PostgreSQL \(p. 1423\)](#).

For detailed specifications of the DB instance classes supported by Aurora PostgreSQL, see [Supported DB engines for DB instance classes \(p. 54\)](#).

Maximum connections to an Aurora PostgreSQL DB instance

An Aurora DB cluster allocates resources based on the DB instance class and its available memory. The maximum number of connections allowed by an Aurora PostgreSQL DB instance is determined by the `max_connections` parameter value specified in the parameter group for that DB instance.

Keep the following factors in mind before you try to change the `max_connections` parameter setting.

- If the `max_connections` value is too low, the Aurora PostgreSQL DB instance might not have sufficient connections available when clients attempt to connect.
- If the `max_connections` value exceeds the number of connections that are actually needed, the unused connections can cause performance to degrade.

The ideal setting for the `max_connections` parameter is one that supports all the client connections your application needs, without an excess of unused connections, plus at least 3 more connections to support AWS automation.

The value of `max_connections` in the default DB parameter group for Aurora PostgreSQL is set to the lower of two values derived from the following Aurora PostgreSQL `LEAST` function:

```
LEAST({DBInstanceClassMemory/9531392}, 5000)
```

Although you can't change values in default parameter groups, you can create your own custom DB cluster parameter group and modify your Aurora PostgreSQL DB cluster to use it. If you do this, be sure that you reboot the DB cluster after applying your custom parameter group. For more information, see [Amazon Aurora PostgreSQL parameters \(p. 1547\)](#) and [Creating a DB cluster parameter group \(p. 343\)](#). To learn more about Aurora DB cluster and DB parameter groups, see [Working with DB parameter groups and DB cluster parameter groups \(p. 339\)](#).

Following, you can find a table that lists the highest value that should ever be used for `max_connections` for each DB instance class that can be used with Aurora PostgreSQL.

Instance class	max_connections default value
db.x2g.16xlarge	5000
db.x2g.12xlarge	5000
db.x2g.8xlarge	5000
db.x2g.4xlarge	5000
db.x2g.2xlarge	5000
db.x2g.xlarge	5000
db.x2g.large	3479
db.r6g.16xlarge	5000
db.r6g.12xlarge	5000
db.r6g.8xlarge	5000
db.r6g.4xlarge	5000
db.r6g.2xlarge	5000
db.r6g.xlarge	3479
db.r6g.large	1722
db.r5.24xlarge	5000
db.r5.16xlarge	5000
db.r5.12xlarge	5000
db.r5.8xlarge	5000
db.r5.4xlarge	5000
db.r5.2xlarge	5000
db.r5.xlarge	3300

Instance class	max_connections default value
db.r5.large	1600
db.r4.16xlarge	5000
db.r4.8xlarge	5000
db.r4.4xlarge	5000
db.r4.2xlarge	5000
db.r4.xlarge	3200
db.r4.large	1600
db.t4g.large	844
db.t4g.medium	405
db.t3.large	844
db.t3.medium	420

If your application needs more connections than the number listed for your DB instance class, consider the following alternatives.

- Choose a DB instance class that has more memory. If your connection requirements are too high for the DB instance class supporting your Aurora PostgreSQL DB cluster, you can potentially overload your database and decrease performance. For list of DB instance classes for Aurora PostgreSQL, see [Supported DB engines for DB instance classes \(p. 54\)](#). For the amount of memory for each DB instance class, [Hardware specifications for DB instance classes for Aurora \(p. 62\)](#).
- Pool connections by using RDS Proxy with your Aurora PostgreSQL DB cluster. For more information, see [Using Amazon RDS Proxy \(p. 288\)](#).

Temporary storage limits for Aurora PostgreSQL

Aurora PostgreSQL stores tables and indexes in the Aurora storage subsystem. Aurora PostgreSQL uses separate temporary storage for non-persistent temporary files. This includes files that are used for such purposes as sorting large datasets during query processing or for index build operations. For more about storage, see [Amazon Aurora storage and reliability \(p. 64\)](#).

The following table shows the maximum amount of temporary storage available for each Aurora PostgreSQL DB instance class.

DB instance class	Maximum temporary storage available (GiB)
db-x2g-16xlarge	1829
db-x2g-12xlarge	1606
db-x2g-8xlarge	1071
db-x2g-4xlarge	535
db-x2g-2xlarge	268

DB instance class	Maximum temporary storage available (GiB)
db-x2g-xlarge	134
db-x2g-large	67
db.r6g.16xlarge	1008
db.r6g.12xlarge	756
db.r6g.8xlarge	504
db.r6g.4xlarge	252
db.r6g.2xlarge	126
db.r6g.xlarge	63
db.r6g.large	32
db.r5.24xlarge	1500
db.r5.16xlarge	1008
db.r5.12xlarge	748
db.r5.8xlarge	504
db.r5.4xlarge	249
db.r5.2xlarge	124
db.r5.xlarge	62
db.r5.large	31
db.r4.16xlarge	960
db.r4.8xlarge	480
db.r4.4xlarge	240
db.r4.2xlarge	120
db.r4.xlarge	60
db.r4.large	30
db.t4g.large	16.5
db.t4g.medium	8.13
db.t3.large	16
db.t3.medium	7.5

You can monitor the temporary storage available for a DB instance with the `FreeLocalStorage` CloudWatch metric, described in [Amazon CloudWatch metrics for Amazon Aurora \(p. 633\)](#).

For some workloads, you can reduce the amount of temporary storage by allocating more memory to the processes that are performing the operation. To increase the memory available to an operation, increasing the values of the `work_mem` or `maintenance_work_mem` PostgreSQL parameters.

Testing Amazon Aurora PostgreSQL by using fault injection queries

You can test the fault tolerance of your Aurora PostgreSQL DB cluster by using fault injection queries. Fault injection queries are issued as SQL commands to an Amazon Aurora instance. Fault injection queries enable you to schedule simulated tests of the following events:

Topics

- [Testing an instance crash \(p. 1364\)](#)
- [Testing an Aurora Replica failure \(p. 1365\)](#)
- [Testing a disk failure \(p. 1365\)](#)
- [Testing disk congestion \(p. 1366\)](#)

When a fault injection query specifies a crash, it forces a crash of the Aurora PostgreSQL DB instance. The other fault injection queries result in simulations of failure events, but don't cause the event to occur. When you submit a fault injection query, you also specify an amount of time for the failure event simulation to occur.

You can submit a fault injection query to one of your Aurora Replica instances by connecting to the endpoint for the Aurora Replica. For more information, see [Amazon Aurora connection management \(p. 32\)](#).

Note

Fault injection queries for Aurora PostgreSQL are currently supported for the following versions:

- Version 2.4, which is compatible with PostgreSQL version 10.11.
- Version 3.2, which is compatible with PostgreSQL version 11.7.

Testing an instance crash

You can force a crash of an Aurora PostgreSQL instance by using the fault injection query function `aurora_inject_crash()`.

For this fault injection query, a failover does not occur. If you want to test a failover, then you can choose the **Failover** instance action for your DB cluster in the RDS console, or use the `failover-db-cluster` AWS CLI command or the `FailoverDBCluster` RDS API operation.

Syntax

```
SELECT aurora_inject_crash ('instance' | 'dispatcher' | 'node');
```

Options

This fault injection query takes one of the following crash types. The crash type is not case sensitive:

'instance'

A crash of the PostgreSQL-compatible database for the Amazon Aurora instance is simulated.
'dispatcher'

A crash of the dispatcher on the primary instance for the Aurora DB cluster is simulated. The *dispatcher* writes updates to the cluster volume for an Amazon Aurora DB cluster.

'node'

A crash of both the PostgreSQL-compatible database and the dispatcher for the Amazon Aurora instance is simulated.

Testing an Aurora Replica failure

You can simulate the failure of an Aurora Replica by using the fault injection query function `aurora_inject_replica_failure()`.

An Aurora Replica failure blocks replication to the Aurora Replica or all Aurora Replicas in the DB cluster by the specified percentage for the specified time interval. When the time interval completes, the affected Aurora Replicas are automatically synchronized with the primary instance.

Syntax

```
SELECT aurora_inject_replica_failure(  
    percentage_of_failure,  
    time_interval,  
    'replica_name'  
)
```

Options

This fault injection query takes the following parameters:

percentage_of_failure

The percentage of replication to block during the failure event. This value can be a double between 0 and 100. If you specify 0, then no replication is blocked. If you specify 100, then all replication is blocked.

time_interval

The amount of time to simulate the Aurora Replica failure. The interval is in seconds. For example, if the value is 20, the simulation runs for 20 seconds.

Note

Take care when specifying the time interval for your Aurora Replica failure event. If you specify too long an interval, and your writer instance writes a large amount of data during the failure event, then your Aurora DB cluster might assume that your Aurora Replica has crashed and replace it.

replica_name

The Aurora Replica in which to inject the failure simulation. Specify the name of an Aurora Replica to simulate a failure of the single Aurora Replica. Specify an empty string to simulate failures for all Aurora Replicas in the DB cluster.

To identify replica names, see the `server_id` column from the `aurora_replica_status()` function. For example:

```
postgres=> SELECT server_id FROM aurora_replica_status();
```

Testing a disk failure

You can simulate a disk failure for an Aurora PostgreSQL DB cluster by using the fault injection query function `aurora_inject_disk_failure()`.

During a disk failure simulation, the Aurora PostgreSQL DB cluster randomly marks disk segments as faulting. Requests to those segments are blocked for the duration of the simulation.

Syntax

```
SELECT aurora_inject_disk_failure(  
    percentage_of_failure,  
    index,  
    is_disk,  
    time_interval  
)
```

Options

This fault injection query takes the following parameters:

percentage_of_failure

The percentage of the disk to mark as faulting during the failure event. This value can be a double between 0 and 100. If you specify 0, then none of the disk is marked as faulting. If you specify 100, then the entire disk is marked as faulting.

index

A specific logical block of data in which to simulate the failure event. If you exceed the range of available logical blocks or storage nodes data, you receive an error that tells you the maximum index value that you can specify. To avoid this error, see [Displaying volume status for an Aurora PostgreSQL DB cluster \(p. 1367\)](#).

is_disk

Indicates whether the injection failure is to a logical block or a storage node. Specifying true means injection failures are to a logical block. Specifying false means injection failures are to a storage node.

time_interval

The amount of time to simulate the Aurora Replica failure. The interval is in seconds. For example, if the value is 20, the simulation runs for 20 seconds.

Testing disk congestion

You can simulate a disk failure for an Aurora PostgreSQL DB cluster by using the fault injection query function `aurora_inject_disk_congestion()`.

During a disk congestion simulation, the Aurora PostgreSQL DB cluster randomly marks disk segments as congested. Requests to those segments are delayed between the specified minimum and maximum delay time for the duration of the simulation.

Syntax

```
SELECT aurora_inject_disk_congestion(  
    percentage_of_failure,  
    index,  
    is_disk,  
    time_interval,  
    minimum,  
    maximum  
)
```

Options

This fault injection query takes the following parameters:

percentage_of_failure

The percentage of the disk to mark as congested during the failure event. This is a double value between 0 and 100. If you specify 0, then none of the disk is marked as congested. If you specify 100, then the entire disk is marked as congested.

index

A specific logical block of data or storage node to use to simulate the failure event.

If you exceed the range of available logical blocks or storage nodes of data, you receive an error that tells you the maximum index value that you can specify. To avoid this error, see [Displaying volume status for an Aurora PostgreSQL DB cluster \(p. 1367\)](#).

is_disk

Indicates whether the injection failure is to a logical block or a storage node. Specifying true means injection failures are to a logical block. Specifying false means injection failures are to a storage node.

time_interval

The amount of time to simulate the Aurora Replica failure. The interval is in seconds. For example, if the value is 20, the simulation runs for 20 seconds.

minimum, maximum

The minimum and maximum amount of congestion delay, in milliseconds. Valid values range from 0.0 to 100.0 milliseconds. Disk segments marked as congested are delayed for a random amount of time within the minimum and maximum range for the duration of the simulation. The maximum value must be greater than the minimum value.

Displaying volume status for an Aurora PostgreSQL DB cluster

In Amazon Aurora, a DB cluster volume consists of a collection of logical blocks. Each of these represents 10 gigabytes of allocated storage. These blocks are called *protection groups*.

The data in each protection group is replicated across six physical storage devices, called *storage nodes*. These storage nodes are allocated across three Availability Zones (AZs) in the region where the DB cluster resides. In turn, each storage node contains one or more logical blocks of data for the DB cluster volume. For more information about protection groups and storage nodes, see [Introducing the Aurora storage engine](#) on the AWS Database Blog.

Use the `aurora_show_volume_status()` function to return the following server status variables:

- **Disks** — The total number of logical blocks of data for the DB cluster volume.
- **Nodes** — The total number of storage nodes for the DB cluster volume.

You can use the `aurora_show_volume_status()` function to help avoid an error when using the `aurora_inject_disk_failure()` fault injection function. The `aurora_inject_disk_failure()` fault injection function simulates the failure of an entire storage node, or a single logical block of data within a storage node. In the function, you specify the index value of a specific logical block of data or storage node. However, the statement returns an error if you specify an index value greater than the number of logical blocks of data or storage nodes used by the DB cluster volume. For more

information about fault injection queries, see [Testing Amazon Aurora PostgreSQL by using fault injection queries \(p. 1364\)](#).

Note

The `aurora_show_volume_status()` function is available for Aurora PostgreSQL version 10.11. For more information about Aurora PostgreSQL versions, see [Amazon Aurora PostgreSQL releases and engine versions \(p. 1599\)](#).

Syntax

```
SELECT * FROM aurora_show_volume_status();
```

Example

```
customer_database=> SELECT * FROM aurora_show_volume_status();
 disks | nodes
-----+-----
 96   |    45
```

Specifying the RAM disk for the stats_temp_directory

You can use the Aurora PostgreSQL parameter, `rds.pg_stat_ramdisk_size`, to specify the system memory allocated to a RAM disk for storing the PostgreSQL `stats_temp_directory`. The RAM disk parameter is available for all Aurora PostgreSQL versions.

Under certain workloads, setting this parameter can improve performance and decrease IO requirements. For more information about the `stats_temp_directory`, see [the PostgreSQL documentation](#).

To enable a RAM disk for your `stats_temp_directory`, set the `rds.pg_stat_ramdisk_size` parameter to a non-zero value in the DB cluster parameter group used by your DB cluster. The parameter value is in MB. You must restart the DB cluster before the change takes effect. For information about setting parameters, see [Working with DB parameter groups and DB cluster parameter groups \(p. 339\)](#).

For example, the following AWS CLI command sets the RAM disk parameter to 256 MB.

```
aws rds modify-db-cluster-parameter-group \
--db-cluster-parameter-group-name db-cl-pg-ramdisk-testing \
--parameters "ParameterName=rds.pg_stat_ramdisk_size, ParameterValue=256,
ApplyMethod=pending-reboot"
```

After you restart the DB cluster, run the following command to see the status of the `stats_temp_directory`:

```
postgres=>show stats_temp_directory;
```

The command should return the following:

```
stats_temp_directory
-----
/rdsdbramdisk/pg_stat_tmp
(1 row)
```


Scheduling maintenance with the PostgreSQL pg_cron extension

You can use the PostgreSQL pg_cron extension to schedule maintenance commands within a PostgreSQL database. For a complete description, see [What is pg_cron?](#) in the pg_cron documentation.

The pg_cron extension is supported on Aurora PostgreSQL engine version 12.6 and higher 12 versions, and all 13 versions.

Topics

- [Enabling the pg_cron extension \(p. 1370\)](#)
- [Granting permissions to pg_cron \(p. 1370\)](#)
- [Scheduling pg_cron jobs \(p. 1371\)](#)
- [pg_cron reference \(p. 1373\)](#)

Enabling the pg_cron extension

Enable the pg_cron extension as follows:

1. Modify the parameter group associated with your PostgreSQL DB instance and add pg_cron to the shared_preload_libraries parameter value. This change requires a PostgreSQL DB instance restart to take effect. For more information, see [Modifying parameters in a DB parameter group \(p. 347\)](#).
2. After the PostgreSQL DB instance has restarted, run the following command using an account that has the rds_superuser permissions. For example, if you used the default settings when you created your RDS for PostgreSQL DB instance, connect as user postgres and create the extension.

```
CREATE EXTENSION pg_cron;
```

The pg_cron scheduler is set in the default PostgreSQL database named `postgres`. The pg_cron objects are created in this `postgres` database and all scheduling actions run in this database.

3. You can use the default settings, or you can schedule jobs to run in other databases within your PostgreSQL DB instance. To schedule jobs for other databases within your PostgreSQL DB instance, see the example in [Scheduling a cron job for a database other than postgres \(p. 1373\)](#).

Granting permissions to pg_cron

As the rds_superuser role, you can create the pg_cron extension and then grant permissions to other users. For other users to be able to schedule jobs, grant them permissions to objects in the cron schema.

Important

We recommend that you grant access to the cron schema sparingly.

To grant others permission to the cron schema, run the following command.

```
postgres=> GRANT USAGE ON SCHEMA cron TO other-user;
```

This permission provides *other-user* with access to the cron schema to schedule and unschedule cron jobs. However, for the cron jobs to run successfully, the user also needs permission to access the objects in the cron jobs. If the user doesn't have permission, the job fails and errors such as the

following appears in the `postgresql.log`. In this example, the user doesn't have permission to access the `pgbench_accounts` table.

```
2020-12-08 16:41:00 UTC::@[30647]:ERROR: permission denied for table pgbench_accounts
2020-12-08 16:41:00 UTC::@[30647]:STATEMENT: update pgbench_accounts set abalance =
abalance + 1
2020-12-08 16:41:00 UTC::@[27071]:LOG: background worker "pg_cron" (PID 30647) exited with
exit code 1
```

Other messages in the `cron.job_run_details` table appear like the following.

```
postgres=> select jobid, username, status, return_message, start_time from
cron.job_run_details where status = 'failed';
jobid | username | status | return_message | start_time
-----+-----+-----+-----+
143 | unprivuser | failed | ERROR: permission denied for table pgbench_accounts | 2020-12-08 16:41:00.036268+00
143 | unprivuser | failed | ERROR: permission denied for table pgbench_accounts | 2020-12-08 16:40:00.050844+00
143 | unprivuser | failed | ERROR: permission denied for table pgbench_accounts | 2020-12-08 16:42:00.175644+00
143 | unprivuser | failed | ERROR: permission denied for table pgbench_accounts | 2020-12-08 16:43:00.069174+00
143 | unprivuser | failed | ERROR: permission denied for table pgbench_accounts | 2020-12-08 16:44:00.059466+00
(5 rows)
```

For more information, see [The pg_cron tables \(p. 1375\)](#).

Scheduling pg_cron jobs

The following sections demonstrate scheduling pg_cron jobs to perform management tasks.

Note

When creating pg_cron jobs, make sure that the number of `max_worker_processes` is always greater than the number of `cron.max_running_jobs`. A pg_cron job will fail if it runs out of background worker processes. The default number of pg_cron jobs is 5; for more information, see [The pg_cron parameters \(p. 1373\)](#).

Topics

- [Vacuuming a table \(p. 1371\)](#)
- [Purging the pg_cron history table \(p. 1372\)](#)
- [Disabling logging of pg_cron history \(p. 1372\)](#)
- [Scheduling a cron job for a database other than postgres \(p. 1373\)](#)

Vacuuming a table

Autovacuum handles vacuum maintenance for most cases. However, you might want to schedule a vacuum of a specific table at a time of your choosing.

Following is an example of using the `cron.schedule` function to set up a job to use `VACUUM FREEZE` on a specific table every day at 22:00 (GMT).

```
SELECT cron.schedule('manual vacuum', '0 22 * * *', 'VACUUM FREEZE pgbench_accounts');
```

```
schedule
-----
1
(1 row)
```

After the preceding example runs, you can check the history in the `cron.job_run_details` table as follows.

```
postgres=> select * from cron.job_run_details;
   jobid | runid | job_pid | database | username | command | status | return_message | start_time | end_time
-----+-----+-----+-----+-----+-----+-----+-----+
      1 |    1 | 3395 | postgres | adminuser| vacuum freeze pgbench_accounts | succeeded | VACUUM |
2020-12-04 21:10:00.050386+00 | 2020-12-04 21:10:00.072028+00
(1 row)
```

Following is an example of viewing the history in the `cron.job_run_details` table to investigate why a job failed.

```
postgres=> select * from cron.job_run_details where status = 'failed';
   jobid | runid | job_pid | database | username | command | status | return_message | start_time | end_time
-----+-----+-----+-----+-----+-----+-----+-----+
      5 |    4 | 30339 | postgres | adminuser| vacuum freeze pgbench_account | failed | ERROR:
relation "pgbench_account" does not exist | 2020-12-04 21:48:00.015145+00 | 2020-12-04
21:48:00.029567+00
(1 row)
```

For more information, see [The pg_cron tables \(p. 1375\)](#).

Purging the pg_cron history table

The `cron.job_run_details` table contains a history of cron jobs that can become very large over time. We recommend that you schedule a job that purges this table. For example, keeping a week's worth of entries might be sufficient for troubleshooting purposes.

The following example uses the [cron.schedule \(p. 1374\)](#) function to schedule a job that runs every day at midnight to purge the `cron.job_run_details` table. The job keeps only the last seven days. Use your `rds_superuser` account to schedule the job such as the following.

```
SELECT cron.schedule('0 0 * * *', $$DELETE
  FROM cron.job_run_details
 WHERE end_time < now() - interval '7 days'$$);
```

For more information, see [The pg_cron tables \(p. 1375\)](#).

Disabling logging of pg_cron history

To completely disable writing anything to the `cron.job_run_details` table, modify the parameter group associated with the PostgreSQL DB instance and set the `cron.log_run` parameter to off. If you do this, the `pg_cron` extension no longer writes to the table and produces errors only in the `postgresql.log` file. For more information, see [Modifying parameters in a DB parameter group \(p. 347\)](#).

Use the following command to check the value of the cron.log_run parameter.

```
postgres=> SHOW cron.log_run;
```

For more information, see [The pg_cron parameters \(p. 1373\)](#).

Scheduling a cron job for a database other than postgres

The metadata for pg_cron is all held in the PostgreSQL default database named `postgres`. Because background workers are used for running the maintenance cron jobs, you can schedule a job in any of your databases within the PostgreSQL DB instance:

1. In the cron database, schedule the job as you normally do using the [cron.schedule \(p. 1374\)](#).

```
postgres=> SELECT cron.schedule('database1 manual vacuum', '29 03 * * *', 'vacuum freeze test_table');
```

2. As a user with the `rds_superuser` role, update the database column for the job that you just created so that it runs in another database within your PostgreSQL DB instance.

```
postgres=> UPDATE cron.job SET database = 'database1' WHERE jobid = 106;
```

3. Verify by querying the `cron.job` table.

```
postgres=> select * from cron.job;
   jobid | schedule | command | nodename | nodeport | database | username | active | jobname
-----+-----+-----+-----+-----+-----+-----+-----+
 106 | 29 03 * * * | vacuum freeze test_table | localhost | 8192 | database1 | adminuser | t | database1 manual vacuum
 1 | 59 23 * * * | vacuum freeze pgbench_accounts | localhost | 8192 | postgres | adminuser | t | manual vacuum
(2 rows)
```

Note

In some situations, you might add a cron job that you intend to run on a different database. In such cases, the job might try to run in the default database (`postgres`) before you update the correct database column. If the user name has permissions, the job successfully runs in the default database.

pg_cron reference

You can use the following parameters, functions, and tables with the pg_cron extension. For more information, see [What is pg_cron?](#) in the pg_cron documentation.

Topics

- [The pg_cron parameters \(p. 1373\)](#)
- [The cron.schedule\(\) function \(p. 1374\)](#)
- [The cron.unschedule\(\) function \(p. 1375\)](#)
- [The pg_cron tables \(p. 1375\)](#)

The pg_cron parameters

Following is a list of parameters that control the pg_cron extension behavior.

Parameter	Description
cron.database_name	The database in which pg_cron metadata is kept.
cron.host	The hostname to connect to PostgreSQL. You can't modify this value.
cron.log_run	Log all the jobs that run into the job_run_details table. Values are on or off. For more information, see The pg_cron tables (p. 1375) .
cron.log_statement	Log all cron statements before running them. Values are on or off.
cron.max_running_jobs	The maximum number of jobs that can run concurrently.
cron.use_background_workers	Use background workers instead of client sessions. You can't modify this value.

Use the following SQL command to display these parameters and their values.

```
postgres=> SELECT name, setting, short_desc FROM pg_settings WHERE name LIKE 'cron.%' ORDER BY name;
```

The cron.schedule() function

This function schedules a cron job. The job is initially scheduled in the default postgres database. The function returns a bigint value representing the job identifier. To schedule jobs to run in other databases within your PostgreSQL DB instance, see the example in [Scheduling a cron job for a database other than postgres \(p. 1373\)](#).

The function has two syntax formats.

Syntax

```
cron.schedule (job_name,
    schedule,
    command
);
cron.schedule (schedule,
    command
);
```

Parameters

Parameter	Description
job_name	The name of the cron job.
schedule	Text indicating the schedule for the cron job. The format is the standard cron format.
command	Text of the command to run.

Examples

```
postgres=> SELECT cron.schedule ('test','0 10 * * *', 'VACUUM pgbench_history');
schedule
-----
145
(1 row)

postgres=> SELECT cron.schedule ('0 15 * * *', 'VACUUM pgbench_accounts');
schedule
-----
146
(1 row)
```

The cron.unschedule() function

This function deletes a cron job. You can either pass in the job_name or the job_id. A policy makes sure that you are the owner to remove the schedule for the job. The function returns a Boolean indicating success or failure.

The function has the following syntax formats.

Syntax

```
cron.unschedule (job_id);
cron.unschedule (job_name);
```

Parameters

Parameter	Description
job_id	A job identifier that was returned from the cron.schedule function when the cron job was scheduled.
job_name	The name of a cron job that was scheduled with the cron.schedule function.

Examples

```
postgres=> select cron.unschedule(108);
unschedule
-----
t
(1 row)

postgres=> select cron.unschedule('test');
unschedule
-----
t
(1 row)
```

The pg_cron tables

The following tables are used to schedule the cron jobs and record how the jobs completed.

Table	Description
<code>cron.job</code>	<p>Contains the metadata about each scheduled job. Most interactions with this table should be done by using the <code>cron.schedule</code> and <code>cron.unschedule</code> functions.</p> <p>Note We don't recommend giving update or insert privileges directly to this table. Doing so would allow the user to update the <code>username</code> column to run as <code>rds-superuser</code>.</p>
<code>cron.job_run_details</code>	<p>Contains historic information about past scheduled jobs that ran. This is useful to investigate the status, return messages, and start and end time from the job that ran.</p> <p>Note To prevent this table from growing indefinitely, purge it on a regular basis. For an example, see Purging the pg_cron history table (p. 1372).</p>

Tuning with wait events for Aurora PostgreSQL

Wait events are an important tuning tool for Aurora PostgreSQL. If you can find out why sessions are waiting for resources and what they are doing, you are better able to reduce bottlenecks. You can use the information in this section to find possible causes and corrective actions.

Important

The wait events in this section are specific to Aurora PostgreSQL. Use the information in this section to tune only Amazon Aurora, not RDS for PostgreSQL.

Some wait events in this section have no analogs in the open source versions of these database engines. Other wait events have the same names as events in open source engines, but behave differently. For example, Amazon Aurora storage works differently from open source storage, so storage-related wait events indicate different resource conditions.

Topics

- [Essential concepts for Aurora PostgreSQL tuning \(p. 1377\)](#)
- [Aurora PostgreSQL wait events \(p. 1380\)](#)
- [Client:ClientRead \(p. 1381\)](#)
- [Client:ClientWrite \(p. 1384\)](#)
- [CPU \(p. 1385\)](#)
- [IO:BufFileRead and IO:BufFileWrite \(p. 1389\)](#)
- [IO:DataFileRead \(p. 1395\)](#)
- [IO:XactSync \(p. 1401\)](#)
- [ipc:damrecordtxack \(p. 1403\)](#)
- [Lock:advisory \(p. 1403\)](#)
- [Lock:extend \(p. 1405\)](#)
- [Lock:Relation \(p. 1407\)](#)
- [Lock:transactionid \(p. 1410\)](#)

- [Lock:tuple \(p. 1413\)](#)
- [lwlock:buffer_content \(BufferContent\) \(p. 1415\)](#)
- [LWLock:buffer_mapping \(p. 1417\)](#)
- [LWLock:BufferIO \(p. 1418\)](#)
- [LWLock:lock_manager \(p. 1420\)](#)
- [Timeout:PgSleep \(p. 1423\)](#)

Essential concepts for Aurora PostgreSQL tuning

Before you tune your Aurora PostgreSQL database, make sure to learn what wait events are and why they occur. Also review the basic memory and disk architecture of Aurora PostgreSQL. For a helpful architecture diagram, see the [PostgreSQL](#) wikibook.

Topics

- [Aurora PostgreSQL wait events \(p. 1377\)](#)
- [Aurora PostgreSQL memory \(p. 1377\)](#)
- [Aurora PostgreSQL processes \(p. 1379\)](#)

Aurora PostgreSQL wait events

A *wait event* indicates a resource for which a session is waiting. For example, the wait event `Client:ClientRead` occurs when Aurora PostgreSQL is waiting to receive data from the client. Typical resources that a session waits for include the following:

- Single-threaded access to a buffer, for example, when a session is attempting to modify a buffer
- A row that is currently locked by another session
- A data file read
- A log file write

For example, to satisfy a query, the session might perform a full table scan. If the data isn't already in memory, the session waits for the disk I/O to complete. When the buffers are read into memory, the session might need to wait because other sessions are accessing the same buffers. The database records the waits by using a predefined wait event. These events are grouped into categories.

A wait event doesn't by itself show a performance problem. For example, if requested data isn't in memory, reading data from disk is necessary. If one session locks a row for an update, another session waits for the row to be unlocked so that it can update it. A commit requires waiting for the write to a log file to complete. Waits are integral to the normal functioning of a database.

Large numbers of wait events typically show a performance problem. In such cases, you can use wait event data to determine where sessions are spending time. For example, if a report that typically runs in minutes now runs for hours, you can identify the wait events that contribute the most to total wait time. If you can determine the causes of the top wait events, you can sometimes make changes that improve performance. For example, if your session is waiting on a row that has been locked by another session, you can end the locking session.

Aurora PostgreSQL memory

Aurora PostgreSQL memory is divided into shared and local.

Topics

- [Shared memory in Aurora PostgreSQL \(p. 1378\)](#)
- [Local memory in Aurora PostgreSQL \(p. 1378\)](#)

Shared memory in Aurora PostgreSQL

Aurora PostgreSQL allocates shared memory when the instance starts. Shared memory is divided into multiple subareas. Following, you can find a description of the most important ones.

Topics

- [Shared buffers \(p. 1378\)](#)
- [Write ahead log \(WAL\) buffers \(p. 1378\)](#)

Shared buffers

The *shared buffer pool* is an Aurora PostgreSQL memory area that holds all pages that are or were being used by application connections. A *page* is the memory version of a disk block. The shared buffer pool caches the data blocks read from disk. The pool reduces the need to reread data from disk, making the database operate more efficiently.

Every table and index is stored as an array of pages of a fixed size. Each block contains multiple tuples, which correspond to rows. A tuple can be stored in any page.

The shared buffer pool has finite memory. If a new request requires a page that isn't in memory, and no more memory exists, Aurora PostgreSQL evicts a less frequently used page to accommodate the request. The eviction policy is implemented by a clock sweep algorithm.

The `shared_buffers` parameter determines how much memory the server dedicates to caching data.

Write ahead log (WAL) buffers

A *write-ahead log (WAL) buffer* holds transaction data that Aurora PostgreSQL later writes to persistent storage. Using the WAL mechanism, Aurora PostgreSQL can do the following:

- Recover data after a failure
- Reduce disk I/O by avoiding frequent writes to disk

When a client changes data, Aurora PostgreSQL writes the changes to the WAL buffer. When the client issues a `COMMIT`, the WAL writer process writes transaction data to the WAL file.

The `wal_level` parameter determines how much information is written to the WAL.

Local memory in Aurora PostgreSQL

Every backend process allocates local memory for query processing.

Topics

- [Work memory area \(p. 1378\)](#)
- [Maintenance work memory area \(p. 1379\)](#)
- [Temporary buffer area \(p. 1379\)](#)

Work memory area

The *work memory area* holds temporary data for queries that performs sorts and hashes. For example, a query with an `ORDER BY` clause performs a sort. Queries use hash tables in hash joins and aggregations.

The `work_mem` parameter specifies the amount of memory to be used by internal sort operations and hash tables before writing to temporary disk files. The default value is 4 MB. Multiple sessions can run simultaneously, and each session can run maintenance operations in parallel. For this reason, the total work memory used can be multiples of the `work_mem` setting.

Maintenance work memory area

The *maintenance work memory area* caches data for maintenance operations. These operations include vacuuming, creating an index, and adding foreign keys.

The `maintenance_work_mem` parameter specifies the maximum amount of memory to be used by maintenance operations. The default value is 64 MB. A database session can only run one maintenance operation at a time.

Temporary buffer area

The *temporary buffer area* caches temporary tables for each database session.

Each session allocates temporary buffers as needed up to the limit you specify. When the session ends, the server clears the buffers.

The `temp_buffers` parameter sets the maximum number of temporary buffers used by each session. Before the first use of temporary tables within a session, you can change the `temp_buffers` value.

Aurora PostgreSQL processes

Aurora PostgreSQL uses multiple processes.

Topics

- [Postmaster process \(p. 1379\)](#)
- [Backend processes \(p. 1379\)](#)
- [Background processes \(p. 1379\)](#)

Postmaster process

The *postmaster process* is the first process started when you start Aurora PostgreSQL. The postmaster process has the following primary responsibilities:

- Fork and monitor background processes
- Receive authentication requests from client processes, and authenticate them before allowing the database to service requests

Backend processes

If the postmaster authenticates a client request, the postmaster forks a new backend process, also called a `postgres` process. One client process connects to exactly one backend process. The client process and the backend process communicate directly without intervention by the postmaster process.

Background processes

The postmaster process forks several processes that perform different backend tasks. Some of the more important include the following:

- WAL writer

Aurora PostgreSQL writes data in the WAL (write ahead logging) buffer to the log files. The principle of write ahead logging is that the database can't write changes to the data files until after the database writes log records describing those changes to disk. The WAL mechanism reduces disk I/O, and allows Aurora PostgreSQL to use the logs to recover the database after a failure.

- **Background writer**

This process periodically writes dirty (modified) pages from the memory buffers to the data files. A page becomes dirty when a backend process modifies it in memory.

- **Autovacuum daemon**

The daemon consists of the following:

- The autovacuum launcher
- The autovacuum worker processes

When autovacuum is turned on, it checks for tables that have had a large number of inserted, updated, or deleted tuples. The daemon has the following responsibilities:

- Recover or reuse disk space occupied by updated or deleted rows
- Update statistics used by the planner
- Protect against loss of old data because of transaction ID wraparound

The autovacuum feature automates the execution of VACUUM and ANALYZE commands. VACUUM has the following variants: standard and full. Standard vacuum runs in parallel with other database operations. VACUUM FULL requires an exclusive lock on the table it is working on. Thus, it can't run in parallel with operations that access the same table. VACUUM creates a substantial amount of I/O traffic, which can cause poor performance for other active sessions.

Aurora PostgreSQL wait events

The following table lists the wait events for Aurora PostgreSQL that most commonly indicate performance problems, and summarizes the most common causes and corrective actions. The following wait events are a subset of the list in [Amazon Aurora PostgreSQL wait events \(p. 1570\)](#).

Wait event	Definition
Client:ClientRead (p. 1381)	This event occurs when Aurora PostgreSQL is waiting to receive data from the client.
Client:ClientWrite (p. 1384)	This event occurs when Aurora PostgreSQL is waiting to write data to the client.
CPU (p. 1385)	This event occurs when a thread is active in CPU or is waiting for CPU.
IO:BufFileRead and IO:BufFileWrite (p. 1389)	These events occur when Aurora PostgreSQL creates temporary files.
IO:DataFileRead (p. 1395)	This event occurs when a connection waits on a backend process to read a required page from storage because the page isn't available in shared memory.
IO:XactSync (p. 1401)	This event occurs when the database is waiting for the Aurora storage subsystem to acknowledge the commit of a regular transaction, or the commit or rollback of a prepared transaction.

Wait event	Definition
ipc:damrecordtxack (p. 1403)	This event occurs when Aurora PostgreSQL in a session using database activity streams generates an activity stream event, then waits for that event to become durable.
Lock:advisory (p. 1403)	This event occurs when a PostgreSQL application uses a lock to coordinate activity across multiple sessions.
Lock:extend (p. 1405)	This event occurs when a backend process is waiting to lock a relation to extend it while another process has a lock on that relation for the same purpose.
Lock:Relation (p. 1407)	This event occurs when a query is waiting to acquire a lock on a table or view that's currently locked by another transaction.
Lock:transactionid (p. 1410)	This event occurs when a transaction is waiting for a row-level lock.
Lock:tuple (p. 1413)	This event occurs when a backend process is waiting to acquire a lock on a tuple.
llock:buffer_content (BufferContent) (p. 1415)	This event occurs when a session is waiting to read or write a data page in memory while another session has that page locked for writing.
LWLock:buffer_mapping (p. 1417)	This event occurs when a session is waiting to associate a data block with a buffer in the shared buffer pool.
LWLock:BufferIO (p. 1418)	This event occurs when Aurora PostgreSQL or RDS for PostgreSQL is waiting for other processes to finish their input/output (I/O) operations when concurrently trying to access a page.
LWLock:lock_manager (p. 1420)	This event occurs when the Aurora PostgreSQL engine maintains the shared lock's memory area to allocate, check, and deallocate a lock when a fast path lock isn't possible.
Timeout:PgSleep (p. 1423)	This event occurs when a server process has called the pg_sleep function and is waiting for the sleep timeout to expire.

Client:ClientRead

The Client:ClientRead event occurs when Aurora PostgreSQL is waiting to receive data from the client.

Topics

- [Supported engine versions \(p. 1382\)](#)
- [Context \(p. 1382\)](#)
- [Likely causes of increased waits \(p. 1382\)](#)
- [Actions \(p. 1382\)](#)

Supported engine versions

This wait event information is supported for Aurora PostgreSQL version 10 and higher.

Context

An Aurora PostgreSQL DB cluster is waiting to receive data from the client. The Aurora PostgreSQL DB cluster must receive the data from the client before it can send more data to the client. The time that the cluster waits before receiving data from the client is a Client:ClientRead event.

Likely causes of increased waits

Common causes for the Client:ClientRead event to appear in top waits include the following:

Increased network latency

There might be increased network latency between the Aurora PostgreSQL DB cluster and client. Higher network latency increases the time required for DB cluster to receive data from the client.

Increased load on the client

There might be CPU pressure or network saturation on the client. An increase in load on the client can delay transmission of data from the client to the Aurora PostgreSQL DB cluster.

Excessive network round trips

A large number of network round trips between the Aurora PostgreSQL DB cluster and the client can delay transmission of data from the client to the Aurora PostgreSQL DB cluster.

Large copy operation

During a copy operation, the data is transferred from the client's file system to the Aurora PostgreSQL DB cluster. Sending a large amount of data to the DB cluster can delay transmission of data from the client to the DB cluster.

Idle client connection

When a client connects to the Aurora PostgreSQL DB cluster in an `idle in transaction` state, the DB cluster might wait for the client to send more data or issue a command. A connection in this state can lead to an increase in Client:ClientRead events.

PgBouncer used for connection pooling

PgBouncer has a low-level network configuration setting called `pkt_buf`, which is set to 4,096 by default. If the workload is sending query packets larger than 4,096 bytes through PgBouncer, we recommend increasing the `pkt_buf` setting to 8,192. If the new setting doesn't decrease the number of Client:ClientRead events, we recommend increasing the `pkt_buf` setting to larger values, such as 16,384 or 32,768. If the query text is large, the larger setting can be particularly helpful.

Actions

We recommend different actions depending on the causes of your wait event.

Topics

- [Place the clients in the same Availability Zone and VPC subnet as the cluster \(p. 1383\)](#)
- [Scale your client \(p. 1383\)](#)
- [Use current generation instances \(p. 1383\)](#)
- [Increase network bandwidth \(p. 1383\)](#)

- Monitor maximums for network performance (p. 1383)
- Monitor for transactions in the "idle in transaction" state (p. 1383)

Place the clients in the same Availability Zone and VPC subnet as the cluster

To reduce network latency and increase network throughput, place clients in the same Availability Zone and virtual private cloud (VPC) subnet as the Aurora PostgreSQL DB cluster. Make sure that the clients are as geographically close to the DB cluster as possible.

Scale your client

Using Amazon CloudWatch or other host metrics, determine if your client is currently constrained by CPU or network bandwidth, or both. If the client is constrained, scale your client accordingly.

Use current generation instances

In some cases, you might not be using a DB instance class that supports jumbo frames. If you're running your application on Amazon EC2, consider using a current generation instance for the client. Also, configure the maximum transmission unit (MTU) on the client operating system. This technique might reduce the number of network round trips and increase network throughput. For more information, see [Jumbo frames \(9001 MTU\)](#) in the *Amazon EC2 User Guide for Linux Instances*.

For information about DB instance classes, see [Aurora DB instance classes \(p. 54\)](#). To determine the DB instance class that is equivalent to an Amazon EC2 instance type, place db. before the Amazon EC2 instance type name. For example, the r5.8xlarge Amazon EC2 instance is equivalent to the db.r5.8xlarge DB instance class.

Increase network bandwidth

Use NetworkReceiveThroughput and NetworkTransmitThroughput Amazon CloudWatch metrics to monitor incoming and outgoing network traffic on the DB cluster. These metrics can help you to determine if network bandwidth is sufficient for your workload.

If your network bandwidth isn't enough, increase it. If the AWS client or your DB instance is reaching the network bandwidth limits, the only way to increase the bandwidth is to increase your DB instance size.

For more information about CloudWatch metrics, see [Amazon CloudWatch metrics for Amazon Aurora \(p. 633\)](#).

Monitor maximums for network performance

If you are using Amazon EC2 clients, Amazon EC2 provides maximums for network performance metrics, including aggregate inbound and outbound network bandwidth. It also provides connection tracking to ensure that packets are returned as expected and link-local services access for services such as the Domain Name System (DNS). To monitor these maximums, use a current enhanced networking driver and monitor network performance for your client.

For more information, see [Monitor network performance for your Amazon EC2 instance](#) in the *Amazon EC2 User Guide for Linux Instances* and [Monitor network performance for your Amazon EC2 instance](#) in the *Amazon EC2 User Guide for Windows Instances*.

Monitor for transactions in the "idle in transaction" state

Check whether you have an increasing number of idle in transaction connections. To do this, monitor the state column in the pg_stat_activity table. You might be able to identify the connection source by running a query similar to the following.

```
select client_addr, state, count(1) from pg_stat_activity
```

```
where state like 'idle in transaction'  
group by 1,2  
order by 3 desc
```

Client:ClientWrite

The Client:ClientWrite event occurs when Aurora PostgreSQL is waiting to write data to the client.

Topics

- [Supported engine versions \(p. 1384\)](#)
- [Context \(p. 1384\)](#)
- [Likely causes of increased waits \(p. 1384\)](#)
- [Actions \(p. 1384\)](#)

Supported engine versions

This wait event information is supported for Aurora PostgreSQL version 10 and higher.

Context

A client process must read all of the data received from an Aurora PostgreSQL DB cluster before the cluster can send more data. The time that the cluster waits before sending more data to the client is a Client:ClientWrite event.

Reduced network throughput between the Aurora PostgreSQL DB cluster and the client can cause this event. CPU pressure and network saturation on the client can also cause this event. *CPU pressure* is when the CPU is fully utilized and there are tasks waiting for CPU time. *Network saturation* is when the network between the database and client is carrying more data than it can handle.

Likely causes of increased waits

Common causes for the Client:ClientWrite event to appear in top waits include the following:

Increased network latency

There might be increased network latency between the Aurora PostgreSQL DB cluster and client. Higher network latency increases the time required for the client to receive the data.

Increased load on the client

There might be CPU pressure or network saturation on the client. An increase in load on the client delays the reception of data from the Aurora PostgreSQL DB cluster.

Large volume of data sent to the client

The Aurora PostgreSQL DB cluster might be sending a large amount of data to the client. A client might not be able to receive the data as fast as the cluster is sending it. Activities such as a copy of a large table can result in an increase in Client:ClientWrite events.

Actions

We recommend different actions depending on the causes of your wait event.

Topics

- [Place the clients in the same Availability Zone and VPC subnet as the cluster \(p. 1385\)](#)

- [Use current generation instances \(p. 1385\)](#)
- [Reduce the amount of data sent to the client \(p. 1385\)](#)
- [Scale your client \(p. 1385\)](#)

Place the clients in the same Availability Zone and VPC subnet as the cluster

To reduce network latency and increase network throughput, place clients in the same Availability Zone and virtual private cloud (VPC) subnet as the Aurora PostgreSQL DB cluster.

Use current generation instances

In some cases, you might not be using a DB instance class that supports jumbo frames. If you're running your application on Amazon EC2, consider using a current generation instance for the client. Also, configure the maximum transmission unit (MTU) on the client operating system. This technique might reduce the number of network round trips and increase network throughput. For more information, see [Jumbo frames \(9001 MTU\)](#) in the *Amazon EC2 User Guide for Linux Instances*.

For information about DB instance classes, see [Aurora DB instance classes \(p. 54\)](#). To determine the DB instance class that is equivalent to an Amazon EC2 instance type, place db. before the Amazon EC2 instance type name. For example, the r5.8xlarge Amazon EC2 instance is equivalent to the db.r5.8xlarge DB instance class.

Reduce the amount of data sent to the client

When possible, adjust your application to reduce the amount of data that the Aurora PostgreSQL DB cluster sends to the client. Making such adjustments relieves CPU and network contention on the client.

Scale your client

Using Amazon CloudWatch or other host metrics, determine if your client is currently constrained by CPU or network bandwidth, or both. If the client is constrained, scale your client accordingly.

CPU

This event occurs when a thread is active in CPU or is waiting for CPU.

Topics

- [Supported engine versions \(p. 1385\)](#)
- [Context \(p. 1385\)](#)
- [Likely causes of increased waits \(p. 1387\)](#)
- [Actions \(p. 1387\)](#)

Supported engine versions

This wait event information is relevant for Aurora PostgreSQL version 9.6 and higher.

Context

The *central processing unit (CPU)* is the component of a computer that runs instructions. For example, CPU instructions perform arithmetic operations and exchange data in memory. If a query increases the number of instructions that it performs through the database engine, the time spent running the query increases. *CPU scheduling* is giving CPU time to a process. Scheduling is orchestrated by the kernel of the operating system.

Topics

- [How to tell when this wait occurs \(p. 1386\)](#)
- [DBLoadCPU metric \(p. 1386\)](#)
- [os.cpuUtilization metrics \(p. 1386\)](#)
- [Likely cause of CPU scheduling \(p. 1386\)](#)

How to tell when this wait occurs

This CPU wait event indicates that a backend process is active in CPU or is waiting for CPU. You know that it's occurring when a query shows the following information:

- The `pg_stat_activity.state` column has the value `active`.
- The `wait_event_type` and `wait_event` columns in `pg_stat_activity` are both `null`.

To see the backend processes that are using or waiting on CPU, run the following query.

```
SELECT *
FROM   pg_stat_activity
WHERE  state = 'active'
AND    wait_event_type IS NULL
AND    wait_event IS NULL;
```

DBLoadCPU metric

The Performance Insights metric for CPU is DBLoadCPU. The value for DBLoadCPU can differ from the value for the Amazon CloudWatch metric CPUUtilization. The latter metric is collected from the HyperVisor for a database instance.

os.cpuUtilization metrics

Performance Insights operating-system metrics provide detailed information about CPU utilization. For example, you can display the following metrics:

- `os.cpuUtilization.nice.avg`
- `os.cpuUtilization.total.avg`
- `os.cpuUtilization.wait.avg`
- `os.cpuUtilization.idle.avg`

Performance Insights reports the CPU usage by the database engine as `os.cpuUtilization.nice.avg`.

Likely cause of CPU scheduling

From an operating system perspective, the CPU is active when it isn't running the idle thread. The CPU is active while it performs a computation, but it's also active when it waits on memory I/O. This type of I/O dominates a typical database workload.

Processes are likely to wait to get scheduled on a CPU when the following conditions are met:

- The CloudWatch CPUUtilization metric is near 100 percent.
- The average load is greater than the number of vCPUs, indicating a heavy load. You can find the `loadAverageMinute` metric in the OS metrics section in Performance Insights.

Likely causes of increased waits

When the CPU wait event occurs more than normal, possibly indicating a performance problem, typical causes include the following.

Topics

- [Likely causes of sudden spikes \(p. 1387\)](#)
- [Likely causes of long-term high frequency \(p. 1387\)](#)
- [Corner cases \(p. 1387\)](#)

Likely causes of sudden spikes

The most likely causes of sudden spikes are as follows:

- Your application has opened too many simultaneous connections to the database. This scenario is known as a "connection storm."
- Your application workload changed in any of the following ways:
 - New queries
 - An increase in the size of your dataset
 - Index maintenance or creation
 - New functions
 - New operators
 - An increase in parallel query execution
- Your query execution plans have changed. In some cases, a change can cause an increase in buffers. For example, the query is now using a sequential scan when it previously used an index. In this case, the queries need more CPU to accomplish the same goal.

Likely causes of long-term high frequency

The most likely causes of events that recur over a long period:

- Too many backend processes are running concurrently on CPU. These processes can be parallel workers.
- Queries are performing suboptimally because they need a large number of buffers.

Corner cases

If none of the likely causes turn out to be actual causes, the following situations might be occurring:

- The CPU is swapping processes in and out.
- CPU context switching has increased.
- Aurora PostgreSQL code is missing wait events.

Actions

If the CPU wait event dominates database activity, it doesn't necessarily indicate a performance problem. Respond to this event only when performance degrades.

Topics

- [Investigate whether the database is causing the CPU increase \(p. 1388\)](#)

- Determine whether the number of connections increased (p. 1388)
- Respond to workload changes (p. 1389)

Investigate whether the database is causing the CPU increase

Examine the `os.cpuUtilization.nice.avg` metric in Performance Insights. If this value is far less than the CPU usage, nondatabase processes are the main contributor to CPU.

Determine whether the number of connections increased

Examine the `DatabaseConnections` metric in Amazon CloudWatch. Your action depends on whether the number increased or decreased during the period of increased CPU wait events.

The connections increased

If the number of connections went up, compare the number of backend processes consuming CPU to the number of vCPUs. The following scenarios are possible:

- The number of backend processes consuming CPU is less than the number of vCPUs.

In this case, the number of connections isn't an issue. However, you might still try to reduce CPU utilization.

- The number of backend processes consuming CPU is greater than the number of vCPUs.

In this case, consider the following options:

- Decrease the number of backend processes connected to your database. For example, implement a connection pooling solution such as RDS Proxy. To learn more, see [Using Amazon RDS Proxy \(p. 288\)](#).
- Upgrade your instance size to get a higher number of vCPUs.
- Redirect some read-only workloads to reader nodes, if applicable.

The connections didn't increase

Examine the `blks_hit` metrics in Performance Insights. Look for a correlation between an increase in `blks_hit` and CPU usage. The following scenarios are possible:

- CPU usage and `blks_hit` are correlated.

In this case, find the top SQL statements that are linked to the CPU usage, and look for plan changes. You can use either of the following techniques:

- Explain the plans manually and compare them to the expected execution plan.
- Look for an increase in block hits per second and local block hits per second. In the **Top SQL** section of Performance Insights dashboard, choose **Preferences**.
- CPU usage and `blks_hit` aren't correlated.

In this case, determine whether any of the following occurs:

- The application is rapidly connecting to and disconnecting from the database.

Diagnose this behavior by turning on `log_connections` and `log_disconnections`, then analyzing the PostgreSQL logs. Consider using the `pgbadger` log analyzer. For more information, see <https://github.com/darold/pgbadger>.

- The OS is overloaded.

In this case, Performance Insights shows that backend processes are consuming CPU for a longer time than usual. Look for evidence in the Performance Insights `os.cpuUtilization` metrics or the CloudWatch `CPUUtilization` metric. If the operating system is overloaded, look at Enhanced

Monitoring metrics to diagnose further. Specifically, look at the process list and the percentage of CPU consumed by each process.

- Top SQL statements are consuming too much CPU.

Examine statements that are linked to the CPU usage to see whether they can use less CPU. Run an EXPLAIN command, and focus on the plan nodes that have the most impact. Consider using a PostgreSQL execution plan visualizer. To try out this tool, see <http://explain.dalibo.com/>.

Respond to workload changes

If your workload has changed, look for the following types of changes:

New queries

Check whether the new queries are expected. If so, ensure that their execution plans and the number of executions per second are expected.

An increase in the size of the data set

Determine whether partitioning, if it's not already implemented, might help. This strategy might reduce the number of pages that a query needs to retrieve.

Index maintenance or creation

Check whether the schedule for the maintenance is expected. A best practice is to schedule maintenance activities outside of peak activities.

New functions

Check whether these functions perform as expected during testing. Specifically, check whether the number of executions per second is expected.

New operators

Check whether they perform as expected during the testing.

An increase in running parallel queries

Determine whether any of the following situations has occurred:

- The relations or indexes involved have suddenly grown in size so that they differ significantly from `min_parallel_table_scan_size` or `min_parallel_index_scan_size`.
- Recent changes have been made to `parallel_setup_cost` or `parallel_tuple_cost`.
- Recent changes have been made to `max_parallel_workers` or `max_parallel_workers_per_gather`.

IO:BufFileRead and IO:BufFileWrite

The `IO:BufFileRead` and `IO:BufFileWrite` events occur when Aurora PostgreSQL creates temporary files. When operations require more memory than the working memory parameters currently define, they write temporary data to persistent storage. This operation is sometimes called "spilling to disk."

Topics

- [Supported engine versions \(p. 1390\)](#)
- [Context \(p. 1390\)](#)
- [Likely causes of increased waits \(p. 1390\)](#)
- [Actions \(p. 1390\)](#)

Supported engine versions

This wait event information is supported for all versions of Aurora PostgreSQL.

Context

`IO:BuffFileRead` and `IO:BuffFileWrite` relate to the work memory area and maintenance work memory area. For more information about these local memory areas, see [Work memory area \(p. 1378\)](#) and [Maintenance work memory area \(p. 1379\)](#).

The default value for `work_mem` is 4 MB. If one session performs operations in parallel, each worker handling the parallelism uses 4 MB of memory. For this reason, set `work_mem` carefully. If you increase the value too much, a database running many sessions might consume too much memory. If you set the value too low, Aurora PostgreSQL creates temporary files in local storage. The disk I/O for these temporary files can reduce performance.

If you observe the following sequence of events, your database might be generating temporary files:

1. Sudden and sharp decreases in availability
2. Fast recovery for the free space

You might also see a "chainsaw" pattern. This pattern can indicate that your database is creating small files constantly.

Likely causes of increased waits

In general, these wait events are caused by operations that consume more memory than the `work_mem` or `maintenance_work_mem` parameters allocate. To compensate, the operations write to temporary files. Common causes for the `IO:BuffFileRead` and `IO:BuffFileWrite` events include the following:

Queries that need more memory than exists in the work memory area

Queries with the following characteristics use the work memory area:

- Hash joins
- `ORDER BY` clause
- `GROUP BY` clause
- `DISTINCT`
- Window functions
- `CREATE TABLE AS SELECT`
- Materialized view refresh

Statements that need more memory than exists in the maintenance work memory area

The following statements use the maintenance work memory area:

- `CREATE INDEX`
- `CLUSTER`

Actions

We recommend different actions depending on the causes of your wait event.

Topics

- [Identify the problem \(p. 1391\)](#)

- [Examine your join queries \(p. 1391\)](#)
- [Examine your ORDER BY and GROUP BY queries \(p. 1392\)](#)
- [Avoid using the DISTINCT operation \(p. 1392\)](#)
- [Consider using window functions instead of GROUP BY functions \(p. 1393\)](#)
- [Investigate materialized views and CTAS statements \(p. 1393\)](#)
- [Use pg_repack when you create indexes \(p. 1393\)](#)
- [Increase maintenance_work_mem when you cluster tables \(p. 1394\)](#)
- [Tune memory to prevent IO:BuffFileRead and IO:BuffFileWrite \(p. 1394\)](#)

Identify the problem

Assume a situation in which Performance Insights isn't turned on and you suspect that `IO:BuffFileRead` and `IO:BuffFileWrite` are occurring more frequently than is normal. Do the following:

1. Examine the `FreeLocalStorage` metric in Amazon CloudWatch.
2. Look for a chainsaw pattern, which is a series of jagged spikes.

A chainsaw pattern indicates a quick consumption and release of storage, often associated with temporary files. If you notice this pattern, turn on Performance Insights. When using Performance Insights, you can identify when the wait events occur and which queries are associated with them. Your solution depends on the specific query causing the events.

Or set the parameter `log_temp_files`. This parameter logs all queries generating more than threshold KB of temporary files. If the value is 0, Aurora PostgreSQL logs all temporary files. If the value is 1024, Aurora PostgreSQL logs all queries that produce temporary files larger than 1 MB. For more information about `log_temp_files`, see [Error Reporting and Logging](#) in the PostgreSQL documentation.

Examine your join queries

Your application probably use joins. For example, the following query joins four tables.

```
SELECT *
  FROM order
 INNER JOIN order_item
    ON (order.id = order_item.order_id)
 INNER JOIN customer
    ON (customer.id = order.customer_id)
 INNER JOIN customer_address
    ON (customer_address.customer_id = customer.id AND
        order.customer_address_id = customer_address.id)
 WHERE customer.id = 1234567890;
```

A possible cause of spikes in temporary file usage is a problem in the query itself. For example, a broken clause might not filter the joins properly. Consider the second inner join in the following example.

```
SELECT *
  FROM order
 INNER JOIN order_item
    ON (order.id = order_item.order_id)
 INNER JOIN customer
    ON (customer.id = customer.id)
 INNER JOIN customer_address
    ON (customer_address.customer_id = customer.id AND
        order.customer_address_id = customer_address.id)
 WHERE customer.id = 1234567890;
```

The preceding query mistakenly joins `customer.id` to `customer.id`, generating a Cartesian product between every customer and every order. This type of accidental join generates large temporary files. Depending on the size of the tables, a Cartesian query can even fill up storage. Your application might have Cartesian joins when the following conditions are met:

- You see large, sharp decreases in storage availability, followed by fast recovery.
- No indexes are being created.
- No `CREATE TABLE FROM SELECT` statements are being issued.
- No materialized views are being refreshed.

To see whether the tables are being joined using the proper keys, inspect your query and object-relational mapping directives. Bear in mind that certain queries of your application are not called all the time, and some queries are dynamically generated.

Examine your ORDER BY and GROUP BY queries

In some cases, an `ORDER BY` clause can result in excessive temporary files. Consider the following guidelines:

- Only include columns in an `ORDER BY` clause when they need to be ordered. This guideline is especially important for queries that return thousands of rows and specify many columns in the `ORDER BY` clause.
- Considering creating indexes to accelerate `ORDER BY` clauses when they match columns that have the same ascending or descending order. Partial indexes are preferable because they are smaller. Smaller indexes are read and traversed more quickly.
- If you create indexes for columns that can accept null values, consider whether you want the null values stored at the end or at the beginning of the indexes.

If possible, reduce the number of rows that need to be ordered by filtering the result set. If you use `WITH` clause statements or subqueries, remember that an inner query generates a result set and passes it to the outside query. The more rows that a query can filter out, the less ordering the query needs to do.

- If you don't need to obtain the full result set, use the `LIMIT` clause. For example, if you only want the top five rows, a query using the `LIMIT` clause doesn't keep generating results. In this way, the query requires less memory and temporary files.

A query that uses a `GROUP BY` clause can also require temporary files. `GROUP BY` queries summarize values by using functions such as the following:

- `COUNT`
- `AVG`
- `MIN`
- `MAX`
- `SUM`
- `STDDEV`

To tune `GROUP BY` queries, follow the recommendations for `ORDER BY` queries.

Avoid using the DISTINCT operation

If possible, avoid using the `DISTINCT` operation to remove duplicated rows. The more unnecessary and duplicated rows that your query returns, the more expensive the `DISTINCT` operation becomes. If possible, add filters in the `WHERE` clause even if you use the same filters for different tables. Filtering

the query and joining correctly improves your performance and reduces resource use. It also prevents incorrect reports and results.

If you need to use `DISTINCT` for multiple rows of a same table, consider creating a composite index. Grouping multiple columns in an index can improve the time to evaluate distinct rows. Also, if you use Amazon Aurora PostgreSQL version 10 or higher, you can correlate statistics among multiple columns by using the `CREATE STATISTICS` command.

Consider using window functions instead of GROUP BY functions

Using `GROUP BY`, you change the result set, and then retrieve the aggregated result. Using window functions, you aggregate data without changing the result set. A window function uses the `OVER` clause to perform calculations across the sets defined by the query, correlating one row with another. You can use all the `GROUP BY` functions in window functions, but also use functions such as the following:

- `RANK`
- `ARRAY_AGG`
- `ROW_NUMBER`
- `LAG`
- `LEAD`

To minimize the number of temporary files generated by a window function, remove duplications for the same result set when you need two distinct aggregations. Consider the following query.

```
SELECT sum(salary) OVER (PARTITION BY dept ORDER BY salary DESC) as sum_salary
      , avg(salary) OVER (PARTITION BY dept ORDER BY salary ASC) as avg_salary
   FROM empsalary;
```

You can rewrite the query with the `WINDOW` clause as follows.

```
SELECT sum(salary) OVER w as sum_salary
      , avg(salary) OVER w as avg_salary
   FROM empsalary
 WINDOW w AS (PARTITION BY dept ORDER BY salary DESC);
```

By default, the Aurora PostgreSQL execution planner consolidates similar nodes so that it doesn't duplicate operations. However, by using an explicit declaration for the window block, you can maintain the query more easily. You might also improve performance by preventing duplication.

Investigate materialized views and CTAS statements

When a materialized view refreshes, it runs a query. This query can contain an operation such as `GROUP BY`, `ORDER BY`, or `DISTINCT`. During a refresh, you might observe large numbers of temporary files and the wait events `IO:BufFileWrite` and `IO:BufFileRead`. Similarly, when you create a table based on a `SELECT` statement, the `CREATE TABLE` statement runs a query. To reduce the temporary files needed, optimize the query.

Use pg_repack when you create indexes

When you create an index, the engine orders the result set. As tables grow in size, and as values in the indexed column become more diverse, the temporary files require more space. In most cases, you can't prevent the creation of temporary files for large tables without modifying the maintenance work memory area. For more information, see [Maintenance work memory area \(p. 1379\)](#).

A possible workaround when recreating a large index is to use the `pg_repack` tool. For more information, see [Reorganize tables in PostgreSQL databases with minimal locks](#) in the `pg_repack` documentation.

Increase maintenance_work_mem when you cluster tables

The CLUSTER command clusters the table specified by *table_name* based on an existing index specified by *index_name*. Aurora PostgreSQL physically recreates the table to match the order of a given index.

When magnetic storage was prevalent, clustering was common because storage throughput was limited. Now that SSD-based storage is common, clustering is less popular. However, if you cluster tables, you can still increase performance slightly depending on the table size, index, query, and so on.

If you run the CLUSTER command and observe the wait events IO:BuffFileWrite and IO:BuffFileRead, tune `maintenance_work_mem`. Increase the memory size to a fairly large amount. A high value means that the engine can use more memory for the clustering operation.

Tune memory to prevent IO:BuffFileRead and IO:BuffFileWrite

In some situation, you need to tune memory. Your goal is to balance the following requirements:

- The `work_mem` value (see [Work memory area \(p. 1378\)](#))
- The memory remaining after discounting the `shared_buffers` value (see [Buffer pool \(p. 839\)](#))
- The maximum connections opened and in use, which is limited by `max_connections`

Increase the size of the work memory area

In some situations, your only option is to increase the memory used by your session. If your queries are correctly written and are using the correct keys for joins, consider increasing the `work_mem` value. For more information, see [Work memory area \(p. 1378\)](#).

To find out how many temporary files a query generates, set `log_temp_files` to 0. If you increase the `work_mem` value to the maximum value identified in the logs, you prevent the query from generating temporary files. However, `work_mem` sets the maximum per plan node for each connection or parallel worker. If the database has 5,000 connections, and if each one uses 256 MiB memory, the engine needs 1.2 TiB of RAM. Thus, your instance might run out of memory.

Reserve sufficient memory for the shared buffer pool

Your database uses memory areas such as the shared buffer pool, not just the work memory area. Consider the requirements of these additional memory areas before you increase `work_mem`. For more information about the buffer pool, see [Buffer pool \(p. 839\)](#).

For example, assume that your Aurora PostgreSQL instance class is db.r5.2xlarge. This class has 64 GiB of memory. By default, 75 percent of the memory is reserved for the shared buffer pool. After you subtract the amount allocated to the shared memory area, 16,384 MB remains. Don't allocate the remaining memory exclusively to the work memory area because the operating system and the engine also require memory.

The memory that you can allocate to `work_mem` depends on the instance class. If you use a larger instance class, more memory is available. However, in the preceding example, you can't use more than 16 GiB. Otherwise, your instance becomes unavailable when it runs out of memory. To recover the instance from the unavailable state, the Aurora PostgreSQL automation services automatically restart.

Manage the number of connections

Suppose that your database instance has 5,000 simultaneous connections. Each connection uses at least 4 MiB of `work_mem`. The high memory consumption of the connections is likely to degrade performance. In response, you have the following options:

- Upgrade to a larger instance class.

- Decrease the number of simultaneous database connections by using a connection proxy or pooler.

For proxies, consider Amazon RDS Proxy, pgBouncer, or a connection pooler based on your application. This solution alleviates the CPU load. It also reduces the risk when all connections require the work memory area. When fewer database connections exist, you can increase the value of `work_mem`. In this way, you reduce the occurrence of the `IO:BufFileRead` and `IO:BufFileWrite` wait events. Also, the queries waiting for the work memory area speed up significantly.

IO:DataFileRead

The `IO:DataFileRead` event occurs when a connection waits on a backend process to read a required page from storage because the page isn't available in shared memory.

Topics

- [Supported engine versions \(p. 1395\)](#)
- [Context \(p. 1395\)](#)
- [Likely causes of increased waits \(p. 1395\)](#)
- [Actions \(p. 1396\)](#)

Supported engine versions

This wait event information is supported for all versions of Aurora PostgreSQL.

Context

All queries and data manipulation (DML) operations access pages in the buffer pool. Statements that can induce reads include `SELECT`, `UPDATE`, and `DELETE`. For example, an `UPDATE` can read pages from tables or indexes. If the page being requested or updated isn't in the shared buffer pool, this read can lead to the `IO:DataFileRead` event.

Because the shared buffer pool is finite, it can fill up. In this case, requests for pages that aren't in memory force the database to read blocks from disk. If the `IO:DataFileRead` event occurs frequently, your shared buffer pool might be too small to accommodate your workload. This problem is acute for `SELECT` queries that read a large number of rows that don't fit in the buffer pool. For more information about the buffer pool, see [Buffer pool \(p. 839\)](#).

Likely causes of increased waits

Common causes for the `IO:DataFileRead` event include the following:

Connection spikes

You might find multiple connections generating the same number of `IO:DataFileRead` wait events. In this case, a spike (sudden and large increase) in `IO:DataFileRead` events can occur.

SELECT and DML statements performing sequential scans

Your application might be performing a new operation. Or an existing operation might change because of a new execution plan. In such cases, look for tables (particularly large tables) that have a greater `seq_scan` value. Find them by querying `pg_stat_user_tables`. To track queries that are generating more read operations, use the extension `pg_stat_statements`.

CTAS and CREATE INDEX for large data sets

A `CTAS` is a `CREATE TABLE AS SELECT` statement. If you run a `CTAS` using a large data set as a source, or create an index on a large table, the `IO:DataFileRead` event can occur. When you

create an index, the database might need to read the entire object using a sequential scan. A CTAS generates IO:DataFile reads when pages aren't in memory.

Multiple vacuum workers running at the same time

Vacuum workers can be triggered manually or automatically. We recommend adopting an aggressive vacuum strategy. However, when a table has many updated or deleted rows, the IO:DataFileRead waits increase. After space is reclaimed, the vacuum time spent on IO:DataFileRead decreases.

Ingesting large amounts of data

When your application ingests large amounts of data, ANALYZE operations might occur more often. The ANALYZE process can be triggered by an autovacuum launcher or invoked manually.

The ANALYZE operation reads a subset of the table. The number of pages that must be scanned is calculated by multiplying 30 by the default_statistics_target value. For more information, see the [PostgreSQL documentation](#). The default_statistics_target parameter accepts values between 1 and 10,000, where the default is 100.

Resource starvation

If instance network bandwidth or CPU are consumed, the IO:DataFileRead event might occur more frequently.

Actions

We recommend different actions depending on the causes of your wait event.

Topics

- [Check predicate filters for queries that generate waits \(p. 1396\)](#)
- [Minimize the effect of maintenance operations \(p. 1397\)](#)
- [Respond to high numbers of connections \(p. 1401\)](#)

Check predicate filters for queries that generate waits

Assume that you identify specific queries that are generating IO:DataFileRead wait events. You might identify them using the following techniques:

- Performance Insights
- Catalog views such as the one provided by the extension pg_stat_statements
- The catalog view pg_stat_all_tables, if it periodically shows an increased number of physical reads
- The pg_statio_all_tables view, if it shows that _read counters are increasing

We recommend that you determine which filters are used in the predicate (WHERE clause) of these queries. Follow these guidelines:

- Run the EXPLAIN command. In the output, identify which types of scans are used. A sequential scan doesn't necessarily indicate a problem. Queries that use sequential scans naturally produce more IO:DataFileRead events when compared to queries that use filters.

Find out whether the column listed in the WHERE clause is indexed. If not, consider creating an index for this column. This approach avoids the sequential scans and reduces the IO:DataFileRead events. If a query has restrictive filters and still produces sequential scans, evaluate whether the proper indexes are being used.

- Find out whether the query is accessing a very large table. In some cases, partitioning a table can improve performance, allowing the query to only read necessary partitions.

- Examine the cardinality (total number of rows) from your join operations. Note how restrictive the values are that you're passing in the filters for your `WHERE` clause. If possible, tune your query to reduce the number of rows that are passed in each step of the plan.

Minimize the effect of maintenance operations

Maintenance operations such as `VACUUM` and `ANALYZE` are important. We recommend that you don't turn them off because you find `IO:DataFileRead` wait events related to these maintenance operations. The following approaches can minimize the effect of these operations:

- Run maintenance operations manually during off-peak hours. This technique prevents the database from reaching the threshold for automatic operations.
- For very large tables, consider partitioning the table. This technique reduces the overhead of maintenance operations. The database only accesses the partitions that require maintenance.
- When you ingest large amounts of data, consider disabling the autoanalyze feature.

The autovacuum feature is automatically triggered for a table when the following formula is true.

```
pg_stat_user_tables.n_dead_tup > (pg_class.reltuples * autovacuum_vacuum_scale_factor) +
autovacuum_vacuum_threshold
```

The view `pg_stat_user_tables` and catalog `pg_class` have multiple rows. One row can correspond to one row in your table. This formula assumes that the `reltuples` are for a specific table. The parameters `autovacuum_vacuum_scale_factor` (0.20 by default) and `autovacuum_vacuum_threshold` (50 tuples by default) are usually set globally for the whole instance. However, you can set different values for a specific table.

Topics

- [Find tables consuming unnecessary space \(p. 1397\)](#)
- [Find indexes consuming unnecessary space \(p. 1398\)](#)
- [Find tables that are eligible to be autovacuumed \(p. 1400\)](#)

Find tables consuming unnecessary space

To find tables consuming unnecessary space, run the following query.

```
/*
* WARNING: Run with a nonsuperuser role, the query inspects only tables
* that you have the permission to read.
* This query is compatible with PostgreSQL 9.0 and later.
*/

SELECT current_database(), schemaname, tblname, bs*tblpages AS real_size,
       (tblpages-est_tblpages)*bs AS extra_size,
       CASE WHEN tblpages - est_tblpages > 0
             THEN 100 * (tblpages - est_tblpages)/tblpages::float
             ELSE 0
       END AS extra_ratio, fillfactor, (tblpages-est_tblpages_ff)*bs AS bloat_size,
       CASE WHEN tblpages - est_tblpages_ff > 0
             THEN 100 * (tblpages - est_tblpages_ff)/tblpages::float
             ELSE 0
       END AS bloat_ratio, is_na
       -- , (pst).free_percent + (pst).dead_tuple_percent AS real_frag
FROM (
  SELECT
    ceil( reltuples / ( (bs-page_hdr)/tpl_size ) ) + ceil( toasttuples / 4 )
    AS est_tblpages,
```

```

    ceil( reltuples / ( (bs-page_hdr)*fillfactor/(tpl_size*100) ) ) + ceil( toasttuples /
4 )
    AS est_tblpages_ff,
tblpages, fillfactor, bs, tblid, schemaname, tblname, heappages, toastpages, is_na
-- ,stattuple.pgstattuple(tblid) AS pst
FROM (
SELECT
( 4 + tpl_hdr_size + tpl_data_size + (2*ma)
- CASE WHEN tpl_hdr_size%ma = 0 THEN ma ELSE tpl_hdr_size%ma END
- CASE WHEN ceil(tpl_data_size)::int%ma = 0 THEN ma ELSE ceil(tpl_data_size)::int
%ma END
) AS tpl_size, bs - page_hdr AS size_per_block, (heappages + toastpages) AS tblpages,
heappages,
toastpages, reltuples, toasttuples, bs, page_hdr, tblid, schemaname, tblname,
fillfactor, is_na
FROM (
SELECT
tbl.oid AS tblid, ns.nspname AS schemaname, tbl.relname AS tblname, tbl.reltuples,
tbl.relpages AS heappages, coalesce(toast.relpages, 0) AS toastpages,
coalesce(toast.reltuples, 0) AS toasttuples,
coalesce(substring(
array_to_string(tbl.reloptions, ' ')
FROM 'fillfactor=(0-9]+)::smallint, 100) AS fillfactor,
current_setting('block_size')::numeric AS bs,
CASE WHEN version()~'mingw32' OR version()~'64-bit|x86_64|ppc64|ia64|amd64'
THEN 8
ELSE 4
END AS ma,
24 AS page_hdr,
23 + CASE WHEN MAX(coalesce(null_frac,0)) > 0 THEN ( 7 + count(*) ) / 8 ELSE 0::int
END
+ CASE WHEN tbl.relhasoids THEN 4 ELSE 0 END AS tpl_hdr_size,
sum( (1-coalesce(s.null_frac, 0)) * coalesce(s.avg_width, 1024) ) AS tpl_data_size,
bool_or(att.atttypid = 'pg_catalog.name'::regtype)
OR count(att.attname) <> count(s.attname) AS is_na
FROM pg_attribute AS att
JOIN pg_class AS tbl ON att.attrelid = tbl.oid
JOIN pg_namespace AS ns ON ns.oid = tbl.relnamespace
LEFT JOIN pg_stats AS s ON s.schemaname=ns.nspname
AND s.tablename = tbl.relname AND s.inherited=false AND s.attname=att.attname
LEFT JOIN pg_class AS toast ON tbl.reltoastrelid = toast.oid
WHERE att.attnum > 0 AND NOT att.attisdropped
AND tbl.relkind = 'r'
GROUP BY 1,2,3,4,5,6,7,8,9,10, tbl.relhasoids
ORDER BY 2,3
) AS s
) AS s2
) AS s3 ;
-- WHERE NOT is_na
-- AND tblpages*((pst).free_percent + (pst).dead_tuple_percent)::float4/100 >= 1

```

Find indexes consuming unnecessary space

To find indexes consuming unnecessary space, run the following query.

```

-- WARNING: run with a nonsuperuser role, the query inspects
-- only indexes on tables you have permissions to read.
-- WARNING: rows with is_na = 't' are known to have bad statistics ("name" type is not
supported).
-- This query is compatible with PostgreSQL 8.2 and later.

SELECT current_database(), nspname AS schemaname, tblname, idxname, bs*(relopages)::bigint
AS real_size,
bs*(relopages-est_pages)::bigint AS extra_size,

```

```

100 * (relpages-est_pages)::float / relpages AS extra_ratio,
fillfactor, bs*(relpages-est_pages_ff) AS bloat_size,
100 * (relpages-est_pages_ff)::float / relpages AS bloat_ratio,
is_na
-- , 100-(sub.pst).avg_leaf_density, est_pages, index_tuple_hdr_bm,
-- maxalign, pagehdr, nulldatawidth, nulldatahdrwidth, sub.reltuples, sub.relpages
-- (DEBUG INFO)
FROM (
    SELECT coalesce(1 +
        ceil(reltuples/floor((bs-pageopqdata-pagehdr)/(4+nulldatahdrwidth)::float)), 0
        -- ItemIdData size + computed avg size of a tuple (nulldatahdrwidth)
    ) AS est_pages,
    coalesce(1 +
        ceil(reltuples/floor((bs-pageopqdata-pagehdr)*fillfactor/
(100*(4+nulldatahdrwidth)::float))), 0
        ) AS est_pages_ff,
    bs, nsdbname, table_oid, tblname, idxname, relpages, fillfactor, is_na
    -- ,stattuple.pgstatindex(quote_ident(nsdbname)||'.'||quote_ident(idxname)) AS pst,
    -- index_tuple_hdr_bm, maxalign, pagehdr, nulldatawidth, nulldatahdrwidth, reltuples
    -- (DEBUG INFO)
FROM (
    SELECT maxalign, bs, nsdbname, tblname, idxname, reltuples, relpages, relam, table_oid,
fillfactor,
    ( index_tuple_hdr_bm +
        maxalign - CASE -- Add padding to the index tuple header to align on MAXALIGN
            WHEN index_tuple_hdr_bm%maxalign = 0 THEN maxalign
            ELSE index_tuple_hdr_bm%maxalign
        END
        + nulldatawidth + maxalign - CASE -- Add padding to the data to align on MAXALIGN
            WHEN nulldatawidth = 0 THEN 0
            WHEN nulldatawidth::integer%maxalign = 0 THEN maxalign
            ELSE nulldatawidth::integer%maxalign
        END
    )::numeric AS nulldatahdrwidth, pagehdr, pageopqdata, is_na
    -- , index_tuple_hdr_bm, nulldatawidth -- (DEBUG INFO)
FROM (
    SELECT
        i.nsdbname, i.tblname, i.idxname, i.reltuples, i.relpages, i.relam, a.attrelid AS table_oid,
        current_setting('block_size')::numeric AS bs, fillfactor,
        CASE -- MAXALIGN: 4 on 32bits, 8 on 64bits (and mingw32 ?)
            WHEN version() ~ 'mingw32' OR version() ~ '64-bit|x86_64|ppc64|ia64|amd64' THEN 8
            ELSE 4
        END AS maxalign,
        /* per page header, fixed size: 20 for 7.X, 24 for others */
        24 AS pagehdr,
        /* per page btree opaque data */
        16 AS pageopqdata,
        /* per tuple header: add IndexAttributeBitMapData if some cols are null-able */
        CASE WHEN max(coalesce(s.null_frac,0)) = 0
            THEN 2 -- IndexTupleData size
            ELSE 2 + (( 32 + 8 - 1 ) / 8)
            -- IndexTupleData size + IndexAttributeBitMapData size ( max num filed per index
+ 8 - 1 /8 )
        END AS index_tuple_hdr_bm,
        /* data len: we remove null values save space using it fractionnal part from stats
*/
        sum( (1-coalesce(s.null_frac, 0)) * coalesce(s.avg_width, 1024)) AS nulldatawidth,
        max( CASE WHEN a.atttypid = 'pg_catalog.name'::regtype THEN 1 ELSE 0 END ) > 0 AS is_na
    FROM pg_attribute AS a
    JOIN (
        SELECT nsdbname, tbl.relname AS tblname, idx.relname AS idxname,
        idx.reltuples, idx.relpages, idx.relam,
        indrelid, indexrelid, indkey::smallint[] AS attnum,
        coalesce(substring(

```

```

        array_to_string(idx.reloptions, ' ')
        from 'fillfactor=([0-9]+)::smallint, 90) AS fillfactor
    FROM pg_index
        JOIN pg_class idx ON idx.oid=pg_index.indexrelid
        JOIN pg_class tbl ON tbl.oid=pg_index.indrelid
        JOIN pg_namespace ON pg_namespace.oid = idx.relnamespace
        WHERE pg_index.indisvalid AND tbl.relkind = 'r' AND idx.relpages > 0
    ) AS i ON a.attrelid = i.indexrelid
    JOIN pg_stats AS s ON s.schemaname = i.nspname
        AND ((s.tablename = i.tablename AND s.attname =
    pg_catalog.pg_get_indexdef(a.attrelid, a.attnum, TRUE))
        -- stats from tbl
        OR  (s.tablename = i.idxname AND s.attname = a.attname))
        -- stats from functionnal cols
        JOIN pg_type AS t ON a.atttypid = t.oid
        WHERE a.attnum > 0
        GROUP BY 1, 2, 3, 4, 5, 6, 7, 8, 9
    ) AS s1
    ) AS s2
        JOIN pg_am am ON s2.relam = am.oid WHERE am.amname = 'btree'
) AS sub
-- WHERE NOT is_na
ORDER BY 2,3,4;

```

Find tables that are eligible to be autovacuumed

To find tables that are eligible to be autovacuumed, run the following query.

```

--This query shows tables that need vacuuming and are eligible candidates.
--The following query lists all tables that are due to be processed by autovacuum.
-- During normal operation, this query should return very little.
WITH vbt AS (SELECT setting AS autovacuum_vacuum_threshold
    FROM pg_settings WHERE name = 'autovacuum_vacuum_threshold')
, vsf AS (SELECT setting AS autovacuum_vacuum_scale_factor
    FROM pg_settings WHERE name = 'autovacuum_vacuum_scale_factor')
, fma AS (SELECT setting AS autovacuum_freeze_max_age
    FROM pg_settings WHERE name = 'autovacuum_freeze_max_age')
, sto AS (SELECT opt_oid, split_part(setting, '=', 1) as param,
    split_part(setting, '=', 2) as value
    FROM (SELECT oid opt_oid, unnest(reloptions) setting FROM pg_class) opt)
SELECT
    ''||ns.nspname||''. ""||c.relname||''' as relation
    , pg_size_pretty(pg_table_size(c.oid)) as table_size
    , age(relfrozenxid) as xid_age
    , coalesce(cfma.value::float, autovacuum_freeze_max_age::float)
autovacuum_freeze_max_age
    , (coalesce(cvbt.value::float, autovacuum_vacuum_threshold::float) +
        coalesce(cvsf.value::float, autovacuum_vacuum_scale_factor::float) * c.reltuples)
        as autovacuum_vacuum_tuples
    , n_dead_tup as dead_tuples
FROM pg_class c
JOIN pg_namespace ns ON ns.oid = c.relnamespace
JOIN pg_stat_all_tables stat ON stat.relid = c.oid
JOIN vbt on (1=1)
JOIN vsf ON (1=1)
JOIN fma on (1=1)
LEFT JOIN sto cvbt ON cvbt.param = 'autovacuum_vacuum_threshold' AND c.oid = cvbt.opt_oid
LEFT JOIN sto cvsft ON cvsft.param = 'autovacuum_vacuum_scale_factor' AND c.oid =
    cvsft.opt_oid
LEFT JOIN sto cfma ON cfma.param = 'autovacuum_freeze_max_age' AND c.oid = cfma.opt_oid
WHERE c.relkind = 'r'
AND nspname <> 'pg_catalog'
AND (
    age(relfrozenxid) >= coalesce(cfma.value::float, autovacuum_freeze_max_age::float)

```

```
        or
        coalesce(cvbt.value::float, autovacuum_vacuum_threshold::float) +
        coalesce(cvsf.value::float, autovacuum_vacuum_scale_factor::float) * c.reltuples <=
n_dead_tup
-- or 1 = 1
)
ORDER BY age(relfrozenxid) DES;
```

Respond to high numbers of connections

When you monitor Amazon CloudWatch, you might find that the `DatabaseConnections` metric spikes. This increase indicates an increased number of connections to your database. We recommend the following approach:

- Limit the number of connections that the application can open with each instance. If your application has an embedded connection pool feature, set a reasonable number of connections. Base the number on what the vCPUs in your instance can parallelize effectively.

If your application doesn't use a connection pool feature, consider using Amazon RDS Proxy or an alternative. This approach lets your application open multiple connections with the load balancer. The balancer can then open a restricted number of connections with the database. As fewer connections are running in parallel, your DB instance performs less context switching in the kernel. Queries should progress faster, leading to fewer wait events. For more information, see [Using Amazon RDS Proxy \(p. 288\)](#).
- Whenever possible, take advantage of reader nodes for Aurora PostgreSQL and read replicas for RDS for PostgreSQL. When your application runs a read-only operation, send these requests to the reader-only endpoint. This technique spreads application requests across all reader nodes, reducing the I/O pressure on the writer node.
- Consider scaling up your DB instance. A higher-capacity instance class gives more memory, which gives Aurora PostgreSQL a larger shared buffer pool to hold pages. The larger size also gives the DB instance more vCPUs to handle connections. More vCPUs are particularly helpful when the operations that are generating `IO:DataFileRead` wait events are writes.

IO:XactSync

The `IO:XactSync` event occurs when the database is waiting for the Aurora storage subsystem to acknowledge the commit of a regular transaction, or the commit or rollback of a prepared transaction. A prepared transaction is part of PostgreSQL's support for a two-phase commit.

Topics

- [Supported engine versions \(p. 1401\)](#)
- [Context \(p. 1401\)](#)
- [Likely causes of increased waits \(p. 1402\)](#)
- [Actions \(p. 1402\)](#)

Supported engine versions

This wait event information is supported for all versions of Aurora PostgreSQL.

Context

The event `IO:XactSync` indicates that the instance is spending time waiting for the Aurora storage subsystem to confirm that transaction data was processed.

Likely causes of increased waits

When the `IO:XactSync` event appears more than normal, possibly indicating a performance problem, typical causes include the following:

Network saturation

Traffic between clients and the DB instance or traffic to the storage subsystem might be too heavy for the network bandwidth.

CPU pressure

A heavy workload might be preventing the Aurora storage daemon from getting sufficient CPU time.

Actions

We recommend different actions depending on the causes of your wait event.

Topics

- [Monitor your resources \(p. 1402\)](#)
- [Scale up the CPU \(p. 1402\)](#)
- [Increase network bandwidth \(p. 1402\)](#)
- [Reduce the number of commits \(p. 1403\)](#)

Monitor your resources

To determine the cause of the increased `IO:XactSync` events, check the following metrics:

- `WriteThroughput` and `CommitThroughput` – Changes in write throughput or commit throughput can show an increase in workload.
- `WriteLatency` and `CommitLatency` – Changes in write latency or commit latency can show that the storage subsystem is being asked to do more work.
- `CPUUtilization` – If the instance's CPU utilization is above 90 percent, the Aurora storage daemon might not be getting sufficient time on the CPU. In this case, I/O performance degrades.

For information about these metrics, see [Instance-level metrics for Amazon Aurora \(p. 639\)](#).

Scale up the CPU

To address CPU starvation issues, consider changing to an instance type with more CPU capacity. For information about CPU capacity for a DB instance class, see [Hardware specifications for DB instance classes for Aurora \(p. 62\)](#).

Increase network bandwidth

To determine whether the instance is reaching its network bandwidth limits, check for the following other wait events:

- `IO:DataFileRead`, `IO:BufferRead`, `IO:BufferWrite`, and `IO:XactWrite` – Queries using large amounts of I/O can generate more of these wait events.
- `Client:ClientRead` and `Client:ClientWrite` – Queries with large amounts of client communication can generate more of these wait events.

If network bandwidth is an issue, consider changing to an instance type with more network bandwidth. For information about network performance for a DB instance class, see [Hardware specifications for DB instance classes for Aurora \(p. 62\)](#).

Reduce the number of commits

To reduce the number of commits, combine statements into transaction blocks.

ipc:damrecordtxack

The `ipc:damrecordtxack` event occurs when Aurora PostgreSQL in a session using database activity streams generates an activity stream event, then waits for that event to become durable.

Topics

- [Relevant engine versions \(p. 1403\)](#)
- [Context \(p. 1403\)](#)
- [Causes \(p. 1403\)](#)
- [Actions \(p. 1403\)](#)

Relevant engine versions

This wait event information is relevant for all Aurora PostgreSQL 10.7 and higher 10 versions, 11.4 and higher 11 versions, and all 12 and 13 versions.

Context

In synchronous mode, durability of activity stream events is favored over database performance. While waiting for a durable write of the event, the session blocks other database activity, causing the `ipc:damrecordtxack` wait event.

Causes

The most common cause for the `ipc:damrecordtxack` event to appear in top waits is that the Database Activity Streams (DAS) feature is a holistic audit. Higher SQL activity generates activity stream events that need to be recorded.

Actions

We recommend different actions depending on the causes of your wait event:

- Reduce the number of SQL statements or turn off database activity streams. Doing this reduces the number of events that require durable writes.
- Change to asynchronous mode. Doing this helps to reduce contention on the `ipc:damrecordtxack` wait event.

However, the DAS feature can't guarantee the durability of every event in asynchronous mode.

Lock:advisory

The `Lock:advisory` event occurs when a PostgreSQL application uses a lock to coordinate activity across multiple sessions.

Topics

- [Relevant engine versions \(p. 1404\)](#)

- [Context \(p. 1404\)](#)
- [Causes \(p. 1404\)](#)
- [Actions \(p. 1404\)](#)

Relevant engine versions

This wait event information is relevant for Aurora PostgreSQL versions 9.6 and higher.

Context

PostgreSQL advisory locks are application-level, cooperative locks explicitly locked and unlocked by the user's application code. An application can use PostgreSQL advisory locks to coordinate activity across multiple sessions. Unlike regular, object- or row-level locks, the application has full control over the lifetime of the lock. For more information, see [Advisory Locks](#) in the PostgreSQL documentation.

Advisory locks can be released before a transaction ends or be held by a session across transactions. This isn't true for implicit, system-enforced locks, such as an access-exclusive lock on a table acquired by a `CREATE INDEX` statement.

For a description of the functions used to acquire (lock) and release (unlock) advisory locks, see [Advisory Lock Functions](#) in the PostgreSQL documentation.

Advisory locks are implemented on top of the regular PostgreSQL locking system and are visible in the `pg_locks` system view.

Causes

This lock type is exclusively controlled by an application explicitly using it. Advisory locks that are acquired for each row as part of a query can cause a spike in locks or a long-term buildup.

These effects happen when the query is run in a way that acquires locks on more rows than are returned by the query. The application must eventually release every lock, but if locks are acquired on rows that aren't returned, the application can't find all of the locks.

The following example is from [Advisory Locks](#) in the PostgreSQL documentation.

```
SELECT pg_advisory_lock(id) FROM foo WHERE id > 12345 LIMIT 100;
```

In this example, the `LIMIT` clause can only stop the query's output after the rows have already been internally selected and their ID values locked. This can happen suddenly when a growing data volume causes the planner to choose a different execution plan that wasn't tested during development. The buildup in this case happens because the application explicitly calls `pg_advisory_unlock` for every ID value that was locked. However, in this case it can't find the set of locks acquired on rows that weren't returned. Because the locks are acquired on the session level, they aren't released automatically at the end of the transaction.

Another possible cause for spikes in blocked lock attempts is unintended conflicts. In these conflicts, unrelated parts of the application share the same lock ID space by mistake.

Actions

Review application usage of advisory locks and detail where and when in the application flow each type of advisory lock is acquired and released.

Determine whether a session is acquiring too many locks or a long-running session isn't releasing locks early enough, leading to a slow buildup of locks. You can correct a slow buildup of session-level locks by ending the session using `pg_terminate_backend(pid)`.

A client waiting for an advisory lock appears in `pg_stat_activity` with `wait_event_type=Lock` and `wait_event=advisory`. You can obtain specific lock values by querying the `pg_locks` system view for the same pid, looking for `locktype=advisory` and `granted=f`.

You can then identify the blocking session by querying `pg_locks` for the same advisory lock having `granted=t`, as shown in the following example.

```

SELECT blocked_locks.pid AS blocked_pid,
       blocking_locks.pid AS blocking_pid,
       blocked_activity.username AS blocked_user,
       blocking_activity.username AS blocking_user,
       now() - blocked_activity.xact_start AS blocked_transaction_duration,
       now() - blocking_activity.xact_start AS blocking_transaction_duration,
       concat(blocked_activity.wait_event_type,':',blocked_activity.wait_event) AS
blocked_wait_event,
       concat(blocking_activity.wait_event_type,':',blocking_activity.wait_event) AS
blocking_wait_event,
       blocked_activity.state AS blocked_state,
       blocking_activity.state AS blocking_state,
       blocked_locks.locktype AS blocked_locktype,
       blocking_locks.locktype AS blocking_locktype,
       blocked_activity.query AS blocked_statement,
       blocking_activity.query AS blocking_statement
  FROM pg_catalog.pg_locks blocked_locks
 JOIN pg_catalog.pg_stat_activity blocked_activity ON blocked_activity.pid =
blocked_locks.pid
 JOIN pg_catalog.pg_locks blocking_locks
   ON blocking_locks.locktype = blocked_locks.locktype
  AND blocking_locks.DATABASE IS NOT DISTINCT FROM blocked_locks.DATABASE
  AND blocking_locks.relation IS NOT DISTINCT FROM blocked_locks.relation
  AND blocking_locks.page IS NOT DISTINCT FROM blocked_locks.page
  AND blocking_locks.tuple IS NOT DISTINCT FROM blocked_locks.tuple
  AND blocking_locks.virtualxid IS NOT DISTINCT FROM blocked_locks.virtualxid
  AND blocking_locks.transactionid IS NOT DISTINCT FROM blocked_locks.transactionid
  AND blocking_locks.classid IS NOT DISTINCT FROM blocked_locks.classid
  AND blocking_locks.objid IS NOT DISTINCT FROM blocked_locks.objid
  AND blocking_locks.objsubid IS NOT DISTINCT FROM blocked_locks.objsubid
  AND blocking_locks.pid != blocked_locks.pid
 JOIN pg_catalog.pg_stat_activity blocking_activity ON blocking_activity.pid =
blocking_locks.pid
 WHERE NOT blocked_locks.GRANTED;

```

All of the advisory lock API functions have two sets of arguments, either one `bigint` argument or two `integer` arguments:

- For the API functions with one `bigint` argument, the upper 32 bits are in `pg_locks.classid` and the lower 32 bits are in `pg_locks.objid`.
- For the API functions with two `integer` arguments, the first argument is `pg_locks.classid` and the second argument is `pg_locks.objid`.

The `pg_locks.objsubid` value indicates which API form was used: 1 means one `bigint` argument; 2 means two `integer` arguments.

Lock:extend

The `Lock:extend` event occurs when a backend process is waiting to lock a relation to extend it while another process has a lock on that relation for the same purpose.

Topics

- [Supported engine versions \(p. 1406\)](#)

- [Context \(p. 1406\)](#)
- [Likely causes of increased waits \(p. 1406\)](#)
- [Actions \(p. 1406\)](#)

Supported engine versions

This wait event information is supported for all versions of Aurora PostgreSQL.

Context

The event `Lock:extend` indicates that a backend process is waiting to extend a relation that another backend process holds a lock on while it's extending that relation. Because only one process at a time can extend a relation, the system generates a `Lock:extend` wait event. `INSERT`, `COPY`, and `UPDATE` operations can generate this event.

Likely causes of increased waits

When the `Lock:extend` event appears more than normal, possibly indicating a performance problem, typical causes include the following:

Surge in concurrent inserts or updates to the same table

There might be an increase in the number of concurrent sessions with queries that insert into or update the same table.

Insufficient network bandwidth

The network bandwidth on the DB instance might be insufficient for the storage communication needs of the current workload. This can contribute to storage latency that causes an increase in `Lock:extend` events.

Actions

We recommend different actions depending on the causes of your wait event.

Topics

- [Reduce concurrent inserts and updates to the same relation \(p. 1406\)](#)
- [Increase network bandwidth \(p. 1407\)](#)

Reduce concurrent inserts and updates to the same relation

First, determine whether there's an increase in `tup_inserted` and `tup_updated` metrics and an accompanying increase in this wait event. If so, check which relations are in high contention for insert and update operations. To determine this, query the `pg_stat_all_tables` view for the values in `n_tup_ins` and `n_tup_upd` fields. For information about the `pg_stat_all_tables` view, see [pg_stat_all_tables](#) in the PostgreSQL documentation.

To get more information about blocking and blocked queries, query `pg_stat_activity` as in the following example:

```
SELECT
    blocked.pid,
    blocked.usename,
```

```

blocked.query,
blocking.pid AS blocking_id,
blocking.query AS blocking_query,
blocking.wait_event AS blocking_wait_event,
blocking.wait_event_type AS blocking_wait_event_type
FROM pg_stat_activity AS blocked
JOIN pg_stat_activity AS blocking ON blocking.pid = ANY(pg_blocking_pids(blocked.pid))
where
blocked.wait_event = 'extend'
and blocked.wait_event_type = 'Lock';

pid | username | query | blocking_id | blocking_wait_event | blocking_wait_event_type
-----+-----+-----+-----+-----+
+-----+
+-----+
7143 | myuser | insert into tab1 values (1); | 4600 | INSERT INTO tab1 (a)
SELECT s FROM generate_series(1,1000000) s; | DataFileExtend | IO

```

After you identify relations that contribute to increase Lock:extend events, use the following techniques to reduce the contention:

- Find out whether you can use partitioning to reduce contention for the same table. Separating inserted or updated tuples into different partitions can reduce contention. For information about partitioning, see [Managing PostgreSQL partitions with the pg_partman extension \(p. 1530\)](#).
- If the wait event is mainly due to update activity, consider reducing the relation's fillfactor value. This can reduce requests for new blocks during the update. The fillfactor is a storage parameter for a table that determines the maximum amount of space for packing a table page. It's expressed as a percentage of the total space for a page. For more information about the fillfactor parameter, see [CREATE TABLE](#) in the PostgreSQL documentation.

Important

We highly recommend that you test your system if you change the fillfactor because changing this value can negatively impact performance, depending on your workload.

Increase network bandwidth

To see whether there's an increase in write latency, check the `WriteLatency` metric in CloudWatch. If there is, use the `WriteThroughput` and `ReadThroughput` Amazon CloudWatch metrics to monitor the storage related traffic on the DB cluster. These metrics can help you to determine if network bandwidth is sufficient for the storage activity of your workload.

If your network bandwidth isn't enough, increase it. If your DB instance is reaching the network bandwidth limits, the only way to increase the bandwidth is to increase your DB instance size.

For more information about CloudWatch metrics, see [Amazon CloudWatch metrics for Amazon Aurora \(p. 633\)](#). For information about network performance for each DB instance class, see [Hardware specifications for DB instance classes for Aurora \(p. 62\)](#).

Lock:Relation

The Lock:Relation event occurs when a query is waiting to acquire a lock on a table or view (relation) that's currently locked by another transaction.

Topics

- [Supported engine versions \(p. 1408\)](#)
- [Context \(p. 1408\)](#)
- [Likely causes of increased waits \(p. 1408\)](#)

- [Actions \(p. 1409\)](#)

Supported engine versions

This wait event information is supported for all versions of Aurora PostgreSQL.

Context

Most PostgreSQL commands implicitly use locks to control concurrent access to data in tables. You can also use these locks explicitly in your application code with the `LOCK` command. Many lock modes aren't compatible with each other, and they can block transactions when they're trying to access the same object. When this happens, Aurora PostgreSQL generates a `Lock:Relation` event. Some common examples are the following:

- Exclusive locks such as `ACCESS EXCLUSIVE` can block all concurrent access. Data definition language (DDL) operations such as `DROP TABLE`, `TRUNCATE`, `VACUUM FULL`, and `CLUSTER` acquire `ACCESS EXCLUSIVE` locks implicitly. `ACCESS EXCLUSIVE` is also the default lock mode for `LOCK TABLE` statements that don't specify a mode explicitly.
- Using `CREATE INDEX` (without `CONCURRENT`) on a table conflicts with data manipulation language (DML) statements `UPDATE`, `DELETE`, and `INSERT`, which acquire `ROW EXCLUSIVE` locks.

For more information about table-level locks and conflicting lock modes, see [Explicit Locking](#) in the PostgreSQL documentation.

Blocking queries and transactions typically unblock in one of the following ways:

- Blocking query – The application can cancel the query or the user can end the process. The engine can also force the query to end because of a session's statement-timeout or a deadlock detection mechanism.
- Blocking transaction – A transaction stops blocking when it runs a `ROLLBACK` or `COMMIT` statement. Rollbacks also happen automatically when sessions are disconnected by a client or by network issues, or are ended. Sessions can be ended when the database engine is shut down, when the system is out of memory, and so forth.

Likely causes of increased waits

When the `Lock:Relation` event appears more than normal, possibly indicating a performance problem, typical causes include the following:

Increased concurrent sessions with conflicting table locks

There might be an increase in the number of concurrent sessions with queries that lock the same table with conflicting locking modes.

Maintenance operations

Health maintenance operations such as `VACUUM` and `ANALYZE` can significantly increase the number of conflicting locks. `VACUUM FULL` acquires an `ACCESS EXCLUSIVE` lock, and `ANALYZE` acquires a `SHARE UPDATE EXCLUSIVE` lock. Both types of locks can cause a `Lock:Relation` wait event. Application data maintenance operations such as refreshing a materialized view can also increase blocked queries and transactions.

Locks on reader instances

There might be an increase in locks acquired on reader instances. From the standpoint of locking, Aurora PostgreSQL treats the writer and reader instances as a single unit. Thus, locks acquired on

a reader instance can block queries on the writer. Similarly, locks on the writer can block reader queries.

Actions

We recommend different actions depending on the causes of your wait event.

Topics

- [Reduce the impact of blocking SQL statements \(p. 1409\)](#)
- [Minimize the effect of maintenance operations \(p. 1409\)](#)
- [Check for reader locks \(p. 1409\)](#)

Reduce the impact of blocking SQL statements

To reduce the impact of blocking SQL statements, modify your application code where possible. Following are two common techniques for reducing blocks:

- Use the `NOWAIT` option – Some SQL commands, such as `SELECT` and `LOCK` statements, support this option. The `NOWAIT` directive cancels the lock-requesting query if the lock can't be acquired immediately. This technique can help prevent a blocking session from causing a pile-up of blocked sessions behind it.

For example: Assume that transaction A is waiting on a lock held by transaction B. Now, if B requests a lock on a table that's locked by transaction C, transaction A might be blocked until transaction C completes. But if transaction B uses a `NOWAIT` when it requests the lock on C, it can fail fast and ensure that transaction A doesn't have to wait indefinitely.

- Use `SET lock_timeout` – Set a `lock_timeout` value to limit the time a SQL statement waits to acquire a lock on a relation. If the lock isn't acquired within the timeout specified, the transaction requesting the lock is cancelled. Set this value at the session level.

Minimize the effect of maintenance operations

Maintenance operations such as `VACUUM` and `ANALYZE` are important. We recommend that you don't turn them off because you find `Lock:Relation` wait events related to these maintenance operations. The following approaches can minimize the effect of these operations:

- Run maintenance operations manually during off-peak hours.
- To reduce `Lock:Relation` waits caused by autovacuum tasks, perform any needed autovacuum tuning. For information about tuning autovacuum, see [Working with PostgreSQL autovacuum on Amazon RDS](#) in the *Amazon RDS User Guide*.

Check for reader locks

You can see how concurrent sessions on a writer and readers might be holding locks that block each other. One way to do this is to run a query that returns the lock type and relation, as in the following example.

Writer session	Reader session	Description
<pre>export WRITER=aurorapg1.12345678910.us-west-1.rds.amazonaws.com</pre>	<pre>export READER=aurorapg2.12345678910.us-west-1.rds.amazonaws.com</pre>	This example shows two concurrent sessions. The first column shows the writer session.

Writer session	Reader session	Description
<pre>psql -h \$WRITER psql (15devel, server 10.14) Type "help" for help.</pre>	<pre>psql -h \$READER psql (15devel, server 10.14) Type "help" for help.</pre>	The second column shows the reader session.
<pre>postgres=> CREATE TABLE t1(b integer); CREATE TABLE</pre>		The writer session creates table t1 on the writer instance.
	<pre>postgres=> SET lock_timeout=100; SET</pre>	The reader session sets a lock timeout interval of 100 milliseconds.
	<pre>postgres=> SELECT * FROM t1; b --- (0 rows)</pre>	The reader session tries to read data from table t1 on the reader instance.
<pre>postgres=> BEGIN; BEGIN postgres=> DROP TABLE t1; DROP TABLE postgres=></pre>		The writer session drops t1.
	<pre>postgres=> SELECT * FROM t1; ERROR: canceling statement due to lock timeout LINE 1: SELECT * FROM t1; ^</pre>	The query times out and is canceled on the reader.
	<pre>postgres=> SELECT locktype, relation, mode, backend_type postgres-> FROM pg_locks l, pg_stat_activity t1 postgres-> WHERE l.pid=t1.pid AND relation = 't1'::regclass::oid; locktype relation mode backend_type -----+-----+ +-----+ +-----+ relation 68628525 AccessExclusiveLock aurora wal replay (1 row)</pre>	The reader session queries pg_locks and pg_stat_activity to determine the cause of the error. The result indicates that the aurora wal replay process is holding an ACCESS EXCLUSIVE lock on table t1.

Lock:transactionid

The Lock:transactionid event occurs when a transaction is waiting for a row-level lock.

Topics

- [Supported engine versions \(p. 1411\)](#)
- [Context \(p. 1411\)](#)
- [Likely causes of increased waits \(p. 1411\)](#)
- [Actions \(p. 1412\)](#)

Supported engine versions

This wait event information is supported for all versions of Aurora PostgreSQL.

Context

The event `Lock:transactionid` occurs when a transaction is trying to acquire a row-level lock that has already been granted to a transaction that is running at the same time. The session that shows the `Lock:transactionid` wait event is blocked because of this lock. After the blocking transaction ends in either a `COMMIT` or `ROLLBACK` statement, the blocked transaction can proceed.

The multiversion concurrency control semantics of Aurora PostgreSQL guarantee that readers don't block writers and writers don't block readers. For row-level conflicts to occur, blocking and blocked transactions must issue conflicting statements of the following types:

- `UPDATE`
- `SELECT ... FOR UPDATE`
- `SELECT ... FOR KEY SHARE`

The statement `SELECT ... FOR KEY SHARE` is a special case. The database uses the clause `FOR KEY SHARE` to optimize the performance of referential integrity. A row-level lock on a row can block `INSERT`, `UPDATE`, and `DELETE` commands on other tables that reference the row.

Likely causes of increased waits

When this event appears more than normal, the cause is typically `UPDATE`, `SELECT ... FOR UPDATE`, or `SELECT ... FOR KEY SHARE` statements combined with the following conditions.

Topics

- [High concurrency \(p. 1411\)](#)
- [Idle in transaction \(p. 1411\)](#)
- [Long-running transactions \(p. 1412\)](#)

High concurrency

Aurora PostgreSQL can use granular row-level locking semantics. The probability of row-level conflicts increases when the following conditions are met:

- A highly concurrent workload contends for the same rows.
- Concurrency increases.

Idle in transaction

Sometimes the `pg_stat_activity.state` column shows the value `idle in transaction`. This value appears for sessions that have started a transaction, but haven't yet issued a `COMMIT` or `ROLLBACK`. If the `pg_stat_activity.state` value isn't `active`, the query shown in `pg_stat_activity` is the

most recent one to finish running. The blocking session isn't actively processing a query because an open transaction is holding a lock.

If an idle transaction acquired a row-level lock, it might be preventing other sessions from acquiring it. This condition leads to frequent occurrence of the wait event `Lock:transactionid`. To diagnose the issue, examine the output from `pg_stat_activity` and `pg_locks`.

Long-running transactions

Transactions that run for a long time get locks for a long time. These long-held locks can block other transactions from running.

Actions

Row-locking is a conflict among `UPDATE`, `SELECT ... FOR UPDATE`, or `SELECT ... FOR KEY SHARE` statements. Before attempting a solution, find out when these statements are running on the same row. Use this information to choose a strategy described in the following sections.

Topics

- [Respond to high concurrency \(p. 1412\)](#)
- [Respond to idle transactions \(p. 1412\)](#)
- [Respond to long-running transactions \(p. 1412\)](#)

Respond to high concurrency

If concurrency is the issue, try one of the following techniques:

- Lower the concurrency in the application. For example, decrease the number of active sessions.
- Implement a connection pool. To learn how to pool connections with RDS Proxy, see [Using Amazon RDS Proxy \(p. 288\)](#).
- Design the application or data model to avoid contending `UPDATE` and `SELECT ... FOR UPDATE` statements. You can also decrease the number of foreign keys accessed by `SELECT ... FOR KEY SHARE` statements.

Respond to idle transactions

If `pg_stat_activity.state` shows `idle` in `transaction`, use the following strategies:

- Turn on autocommit wherever possible. This approach prevents transactions from blocking other transactions while waiting for a `COMMIT` or `ROLLBACK`.
- Search for code paths that are missing `COMMIT`, `ROLLBACK`, or `END`.
- Make sure that the exception handling logic in your application always has a path to a valid `end_of_transaction`.
- Make sure that your application processes query results after ending the transaction with `COMMIT` or `ROLLBACK`.

Respond to long-running transactions

If long-running transactions are causing the frequent occurrence of `Lock:transactionid`, try the following strategies:

- Keep row locks out of long-running transactions.
- Limit the length of queries by implementing autocommit whenever possible.

Lock:tuple

The `Lock:tuple` event occurs when a backend process is waiting to acquire a lock on a tuple.

Topics

- [Supported engine versions \(p. 1413\)](#)
- [Context \(p. 1413\)](#)
- [Likely causes of increased waits \(p. 1413\)](#)
- [Actions \(p. 1414\)](#)

Supported engine versions

This wait event information is supported for all versions of Aurora PostgreSQL.

Context

The event `Lock:tuple` indicates that a backend is waiting to acquire a lock on a tuple while another backend holds a conflicting lock on the same tuple. The following table illustrates a scenario in which sessions generate the `Lock:tuple` event.

Time	Session 1	Session 2	Session 3
t1	Starts a transaction.		
t2	Updates row 1.		
t3		Updates row 1. The session acquires an exclusive lock on the tuple and then waits for session 1 to release the lock by committing or rolling back.	
t4			Updates row 1. The session waits for session 2 to release the exclusive lock on the tuple.

Or you can simulate this wait event by using the benchmarking tool `pgbench`. Configure a high number of concurrent sessions to update the same row in a table with a custom SQL file.

To learn more about conflicting lock modes, see [Explicit Locking](#) in the PostgreSQL documentation. To learn more about `pgbench`, see [pgbench](#) in the PostgreSQL documentation.

Likely causes of increased waits

When this event appears more than normal, possibly indicating a performance problem, typical causes include the following:

- A high number of concurrent sessions are trying to acquire a conflicting lock for the same tuple by running `UPDATE` or `DELETE` statements.
- Highly concurrent sessions are running a `SELECT` statement using the `FOR UPDATE` or `FOR NO KEY UPDATE` lock modes.
- Various factors drive application or connection pools to open more sessions to execute the same operations. As new sessions are trying to modify the same rows, DB load can spike, and `Lock:tuple` can appear.

For more information, see [Row-Level Locks](#) in the PostgreSQL documentation.

Actions

We recommend different actions depending on the causes of your wait event.

Topics

- [Investigate your application logic \(p. 1414\)](#)
- [Find the blocker session \(p. 1414\)](#)
- [Reduce concurrency when it is high \(p. 1415\)](#)
- [Troubleshoot bottlenecks \(p. 1415\)](#)

Investigate your application logic

Find out whether a blocker session has been in the `idle in transaction` state for long time. If so, consider ending the blocker session as a short-term solution. You can use the `pg_terminate_backend` function. For more information about this function, see [Server Signaling Functions](#) in the PostgreSQL documentation.

For a long-term solution, do the following:

- Adjust the application logic.
- Use the `idle_in_transaction_session_timeout` parameter. This parameter ends any session with an open transaction that has been idle for longer than the specified amount of time. For more information, see [Client Connection Defaults](#) in the PostgreSQL documentation.
- Use autocommit as much as possible. For more information, see [SET AUTOCOMMIT](#) in the PostgreSQL documentation.

Find the blocker session

While the `Lock:tuple` wait event is occurring, identify the blocker and blocked session by finding out which locks depend on one another. For more information, see [Lock dependency information](#) in the PostgreSQL wiki. To analyze past `Lock:tuple` events, use the Aurora function `aurora_stat_backend_waits`.

The following example queries all sessions, filtering on `tuple` and ordering by `wait_time`.

```
--AURORA_STAT_BACKEND_WAITS
SELECT a.pid,
       a.username,
       a.app_name,
       a.current_query,
       a.current_wait_type,
       a.current_wait_event,
       a.current_state,
       wt.type_name AS wait_type,
       we.event_name AS wait_event,
       a.waits,
       a.wait_time
FROM (SELECT pid,
             username,
             left(application_name,16) AS app_name,
             coalesce(wait_event_type,'CPU') AS current_wait_type,
             coalesce(wait_event,'CPU') AS current_wait_event,
             state AS current_state,
             left(query,80) as current_query,
             (aurora_stat_backend_waits(pid)).*
      FROM pg_stat_activity
```

```

        WHERE pid <> pg_backend_pid()
          AND usename<>'rdsadmin') a
NATURAL JOIN aurora_stat_wait_type() wt
NATURAL JOIN aurora_stat_wait_event() we
WHERE we.event_name = 'tuple'
    ORDER BY a.wait_time;

pid | usename | app_name | current_query | current_wait_type | current_wait_event | current_state | wait_type | wait_event | waits | wait_time
-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
32136 | sys     | psql     | /*session3*/ update tab set col=1 where col=1; | Lock
      | tuple   |           | active       | Lock            | tuple          | 1           |          |
11999 | sys     | psql     | /*session4*/ update tab set col=1 where col=1; | Lock
      | tuple   |           | active       | Lock            | tuple          | 1           |          |

```

Reduce concurrency when it is high

The `Lock:tuple` event might occur constantly, especially in a busy workload time. In this situation, consider reducing the high concurrency for very busy rows. Often, just a few rows control a queue or the Boolean logic, which makes these rows very busy.

You can reduce concurrency by using different approaches based in the business requirement, application logic, and workload type. For example, you can do the following:

- Redesign your table and data logic to reduce high concurrency.
- Change the application logic to reduce high concurrency at the row level.
- Leverage and redesign queries with row-level locks.
- Use the `NOWAIT` clause with retry operations.
- Consider using optimistic and hybrid-locking logic concurrency control.
- Consider changing the database isolation level.

Troubleshoot bottlenecks

The `Lock:tuple` can occur with bottlenecks such as CPU starvation or maximum usage of Amazon EBS bandwidth. To reduce bottlenecks, consider the following approaches:

- Scale up your instance class type.
- Optimize resource-intensive queries.
- Change the application logic.
- Archive data that is rarely accessed.

lwlock:buffer_content (BufferContent)

The `lwlock:buffer_content` event occurs when a session is waiting to read or write a data page in memory while another session has that page locked for writing. In Aurora PostgreSQL 13 and higher, this wait event is called `BufferContent`.

Topics

- [Supported engine versions \(p. 1416\)](#)
- [Context \(p. 1416\)](#)
- [Likely causes of increased waits \(p. 1416\)](#)
- [Actions \(p. 1416\)](#)

Supported engine versions

This wait event information is supported for all versions of Aurora PostgreSQL.

Context

To read or manipulate data, PostgreSQL accesses it through shared memory buffers. To read from the buffer, a process gets a lightweight lock (LWLock) on the buffer content in shared mode. To write to the buffer, it gets that lock in exclusive mode. Shared locks allow other processes to concurrently acquire shared locks on that content. Exclusive locks prevent other processes from getting any type of lock on it.

The `lwlock:buffer_content (BufferContent)` event indicates that multiple processes are attempting to get a lock on contents of a specific buffer.

Likely causes of increased waits

When the `lwlock:buffer_content (BufferContent)` event appears more than normal, possibly indicating a performance problem, typical causes include the following:

Increased concurrent updates to the same data

There might be an increase in the number of concurrent sessions with queries that update the same buffer content. This contention can be more pronounced on tables with a lot of indexes.

Workload data is not in memory

When data that the active workload is processing is not in memory, these wait events can increase. This effect is because processes holding locks can keep them longer while they perform disk I/O operations.

Excessive use of foreign key constraints

Foreign key constraints can increase the amount of time a process holds onto a buffer content lock. This effect is because read operations require a shared buffer content lock on the referenced key while that key is being updated.

Actions

We recommend different actions depending on the causes of your wait event. You might identify `lwlock:buffer_content (BufferContent)` events by using Amazon RDS Performance Insights or by querying the view `pg_stat_activity`.

Topics

- [Improve in-memory efficiency \(p. 1416\)](#)
- [Reduce usage of foreign key constraints \(p. 1416\)](#)
- [Remove unused indexes \(p. 1417\)](#)

Improve in-memory efficiency

To increase the chance that active workload data is in memory, partition tables or scale up your instance class. For information about DB instance classes, see [Aurora DB instance classes \(p. 54\)](#).

Reduce usage of foreign key constraints

Investigate workloads experiencing high numbers of `lwlock:buffer_content (BufferContent)` wait events for usage of foreign key constraints. Remove unnecessary foreign key constraints.

Remove unused indexes

For workloads experiencing high numbers of `lwllock:buffer_content (BufferContent)` wait events, identify unused indexes and remove them.

LWLock:buffer_mapping

This event occurs when a session is waiting to associate a data block with a buffer in the shared buffer pool.

Note

This event appears as `LWLock:buffer_mapping` in Aurora PostgreSQL version 12 and lower, and `LWLock:BufferMapping` in version 13 and higher.

Topics

- [Supported engine versions \(p. 1417\)](#)
- [Context \(p. 1417\)](#)
- [Causes \(p. 1417\)](#)
- [Actions \(p. 1417\)](#)

Supported engine versions

This wait event information is relevant for Aurora PostgreSQL version 9.6 and higher.

Context

The *shared buffer pool* is an Aurora PostgreSQL memory area that holds all pages that are or were being used by processes. When a process needs a page, it reads the page into the shared buffer pool. The `shared_buffers` parameter sets the shared buffer size and reserves a memory area to store the table and index pages. If you change this parameter, make sure to restart the database. For more information, see [Shared buffers \(p. 1378\)](#).

The `LWLock:buffer_mapping` wait event occurs in the following scenarios:

- A process searches the buffer table for a page and acquires a shared buffer mapping lock.
- A process loads a page into the buffer pool and acquires an exclusive buffer mapping lock.
- A process removes a page from the pool and acquires an exclusive buffer mapping lock.

Causes

When this event appears more than normal, possibly indicating a performance problem, the database is paging in and out of the shared buffer pool. Typical causes include the following:

- Large queries
- Bloated indexes and tables
- Full table scans
- A shared pool size that is smaller than the working set

Actions

We recommend different actions depending on the causes of your wait event.

Topics

- [Monitor buffer-related metrics \(p. 1418\)](#)
- [Assess your indexing strategy \(p. 1418\)](#)
- [Reduce the number of buffers that must be allocated quickly \(p. 1418\)](#)

Monitor buffer-related metrics

When `LWLock:buffer_mapping` waits spike, investigate the buffer hit ratio. You can use these metrics to get a better understanding of what is happening in the buffer cache. Examine the following metrics:

`BufferCacheHitRatio`

This Amazon CloudWatch metric measures the percentage of requests that are served by the buffer cache of a DB instance in your DB cluster. You might see this metric decrease in the lead-up to the `LWLock:buffer_mapping` wait event.

`blks_hit`

This Performance Insights counter metric indicates the number of blocks that were retrieved from the shared buffer pool. After the `LWLock:buffer_mapping` wait event appears, you might observe a spike in `blks_hit`.

`blks_read`

This Performance Insights counter metric indicates the number of blocks that required I/O to be read into the shared buffer pool. You might observe a spike in `blks_read` in the lead-up to the `LWLock:buffer_mapping` wait event.

Assess your indexing strategy

To confirm that your indexing strategy is not degrading performance, check the following:

Index bloat

Ensure that index and table bloat aren't leading to unnecessary pages being read into the shared buffer. If your tables contain unused rows, consider archiving the data and removing the rows from the tables. You can then rebuild the indexes for the resized tables.

Indexes for frequently used queries

To determine whether you have the optimal indexes, monitor DB engine metrics in Performance Insights. The `tup_returned` metric shows the number of rows read. The `tup_fetched` metric shows the number of rows returned to the client. If `tup_returned` is significantly larger than `tup_fetched`, the data might not be properly indexed. Also, your table statistics might not be current.

Reduce the number of buffers that must be allocated quickly

To reduce the `LWLock:buffer_mapping` wait events, try to reduce the number of buffers that must be allocated quickly. One strategy is to perform smaller batch operations. You might be able to achieve smaller batches by partitioning your tables.

LWLock:BufferIO

The `LWLock:BufferIO` event occurs when Aurora PostgreSQL or RDS for PostgreSQL is waiting for other processes to finish their input/output (I/O) operations when concurrently trying to access a page. Its purpose is for the same page to be read into the shared buffer.

Topics

- [Relevant engine versions \(p. 1419\)](#)
- [Context \(p. 1419\)](#)
- [Causes \(p. 1419\)](#)
- [Actions \(p. 1419\)](#)

Relevant engine versions

This wait event information is relevant for all Aurora PostgreSQL 13 versions.

Context

Each shared buffer has an I/O lock that is associated with the `LWLock:BufferIO` wait event, each time a block (or a page) has to be retrieved outside the shared buffer pool.

This lock is used to handle multiple sessions that all require access to the same block. This block has to be read from outside the shared buffer pool, defined by the `shared_buffers` parameter.

As soon as the page is read inside the shared buffer pool, the `LWLock:BufferIO` lock is released.

Note

The `LWLock:BufferIO` wait event precedes the [IO:DataFileRead \(p. 1395\)](#) wait event. The `IO:DataFileRead` wait event occurs while data is being read from storage.

For more information on lightweight locks, see [Locking Overview](#).

Causes

Common causes for the `LWLock:BufferIO` event to appear in top waits include the following:

- Multiple backends or connections trying to access the same page that's also pending an I/O operation
- The ratio between the size of the shared buffer pool (defined by the `shared_buffers` parameter) and the number of buffers needed by the current workload
- The size of the shared buffer pool not being well balanced with the number of pages being evicted by other operations
- Large or bloated indexes that require the engine to read more pages than necessary into the shared buffer pool
- Lack of indexes that forces the DB engine to read more pages from the tables than necessary
- Checkpoints occurring too frequently or needing to flush too many modified pages
- Sudden spikes for database connections trying to perform operations on the same page

Actions

We recommend different actions depending on the causes of your wait event:

- Observe Amazon CloudWatch metrics for correlation between sharp decreases in the `BufferCacheHitRatio` and `LWLock:BufferIO` wait events. This effect can mean that you have a small `shared_buffers` setting. You might need to increase it or scale up your DB instance class. You can split your workload into more reader nodes.
- Tune `max_wal_size` and `checkpoint_timeout` based on your workload peak time if you see `LWLock:BufferIO` coinciding with `BufferCacheHitRatio` metric dips. Then identify which query might be causing it.
- Verify whether you have unused indexes, then remove them.

- Use partitioned tables (which also have partitioned indexes). Doing this helps to keep index reordering low and reduces its impact.
- Avoid indexing columns unnecessarily.
- Prevent sudden database connection spikes by using a connection pool.
- Restrict the maximum number of connections to the database as a best practice.

LWLock:lock_manager

This event occurs when the Aurora PostgreSQL engine maintains the shared lock's memory area to allocate, check, and deallocate a lock when a fast path lock isn't possible.

Topics

- [Supported engine versions \(p. 1420\)](#)
- [Context \(p. 1420\)](#)
- [Likely causes of increased waits \(p. 1421\)](#)
- [Actions \(p. 1421\)](#)

Supported engine versions

This wait event information is relevant for Aurora PostgreSQL version 9.6 and higher.

Context

When you issue a SQL statement, Aurora PostgreSQL records locks to protect the structure, data, and integrity of your database during concurrent operations. The engine can achieve this goal using a fast path lock or a path lock that isn't fast. A path lock that isn't fast is more expensive and creates more overhead than a fast path lock.

Fast path locking

To reduce the overhead of locks that are taken and released frequently, but that rarely conflict, backend processes can use fast path locking. The database uses this mechanism for locks that meet the following criteria:

- They use the DEFAULT lock method.
- They represent a lock on a database relation rather than a shared relation.
- They are weak locks that are unlikely to conflict.
- The engine can quickly verify that no conflicting locks can possibly exist.

The engine can't use fast path locking when either of the following conditions is true:

- The lock doesn't meet the preceding criteria.
- No more slots are available for the backend process.

For more information about fast path locking, see [fast path](#) in the PostgreSQL lock manager README and [pg-locks](#) in the PostgreSQL documentation.

Example of a scaling problem for the lock manager

In this example, a table named `purchases` stores five years of data, partitioned by day. Each partition has two indexes. The following sequence of events occurs:

1. You query many days worth of data, which requires the database to read many partitions.
2. The database creates a lock entry for each partition. If partition indexes are part of the optimizer access path, the database creates a lock entry for them, too.
3. When the number of requested locks entries for the same backend process is higher than 16, which is the value of `FP_LOCK_SLOTS_PER_BACKEND`, the lock manager uses the non-fast path lock method.

Modern applications might have hundreds of sessions. If concurrent sessions are querying the parent without proper partition pruning, the database might create hundreds or even thousands of non-fast path locks. Typically, when this concurrency is higher than the number of vCPUs, the `LWLock:lock_manager` wait event appears.

Note

The `LWLock:lock_manager` wait event isn't related to the number of partitions or indexes in a database schema. Instead, it's related to the number of non-fast path locks that the database must control.

Likely causes of increased waits

When the `LWLock:lock_manager` wait event occurs more than normal, possibly indicating a performance problem, the most likely causes of sudden spikes are as follows:

- Concurrent active sessions are running queries that don't use fast path locks. These sessions also exceed the maximum vCPU.
- A large number of concurrent active sessions are accessing a heavily partitioned table. Each partition has multiple indexes.
- The database is experiencing a connection storm. By default, some applications and connection pool software create more connections when the database is slow. This practice makes the problem worse. Tune your connection pool software so that connection storms don't occur.
- A large number of sessions query a parent table without pruning partitions.
- A data definition language (DDL), data manipulation language (DML), or a maintenance command exclusively locks either a busy relation or tuples that are frequently accessed or modified.

Actions

If the `CPU` wait event occurs, it doesn't necessarily indicate a performance problem. Respond to this event only when performance degrades and this wait event is dominating DB load.

Topics

- [Use partition pruning \(p. 1421\)](#)
- [Remove unnecessary indexes \(p. 1422\)](#)
- [Tune your queries for fast path locking \(p. 1422\)](#)
- [Tune for other wait events \(p. 1422\)](#)
- [Reduce hardware bottlenecks \(p. 1422\)](#)
- [Use a connection pooler \(p. 1422\)](#)
- [Upgrade your Aurora PostgreSQL version \(p. 1423\)](#)

Use partition pruning

Partition pruning is a query optimization strategy that excludes unneeded partitions from table scans, thereby improving performance. Partition pruning is turned on by default. If it is turned off, turn it on as follows.

```
SET enable_partition_pruning = on;
```

Queries can take advantage of partition pruning when their `WHERE` clause contains the column used for the partitioning. For more information, see [Partition Pruning](#) in the PostgreSQL documentation.

Remove unnecessary indexes

Your database might contain unused or rarely used indexes. If so, consider deleting them. Do either of the following:

- Learn how to find unnecessary indexes by reading [Unused Indexes](#) in the PostgreSQL wiki.
- Run PG Collector. This SQL script gathers database information and presents it in a consolidated HTML report. Check the "Unused indexes" section. For more information, see [pg-collector](#) in the AWS Labs GitHub repository.

Tune your queries for fast path locking

To find out whether your queries use fast path locking, query the `fastpath` column in the `pg_locks` table. If your queries aren't using fast path locking, try to reduce number of relations per query to fewer than 16.

Tune for other wait events

If `LWLock:lock_manager` is first or second in the list of top waits, check whether the following wait events also appear in the list:

- `Lock:Relation`
- `Lock:transactionid`
- `Lock:tuple`

If the preceding events appear high in the list, consider tuning these wait events first. These events can be a driver for `LWLock:lock_manager`.

Reduce hardware bottlenecks

You might have a hardware bottleneck, such as CPU starvation or maximum usage of your Amazon EBS bandwidth. In these cases, consider reducing the hardware bottlenecks. Consider the following actions:

- Scale up your instance class.
- Optimize queries that consume large amounts of CPU and memory.
- Change your application logic.
- Archive your data.

For more information about CPU, memory, and EBS network bandwidth, see [Amazon RDS Instance Types](#).

Use a connection pooler

If your total number of active connections exceeds the maximum vCPU, more OS processes require CPU than your instance type can support. In this case, consider using or tuning a connection pool. For more information about the vCPUs for your instance type, see [Amazon RDS Instance Types](#).

For more information about connection pooling, see the following resources:

- [Using Amazon RDS Proxy \(p. 288\)](#)

- pgbouncer
- [Connection Pools and Data Sources](#) in the *PostgreSQL Documentation*

Upgrade your Aurora PostgreSQL version

If your current version of Aurora PostgreSQL is lower than 12, upgrade to version 12 or higher. PostgreSQL versions 12 and 13 have an improved partition mechanism. For more information about version 12, see [PostgreSQL 12.0 Release Notes](#). For more information about upgrading Aurora PostgreSQL, see [Amazon Aurora PostgreSQL updates \(p. 1597\)](#).

Timeout:PgSleep

The `Timeout:PgSleep` event occurs when a server process has called the `pg_sleep` function and is waiting for the sleep timeout to expire.

Topics

- [Supported engine versions \(p. 1423\)](#)
- [Likely causes of increased waits \(p. 1423\)](#)
- [Actions \(p. 1423\)](#)

Supported engine versions

This wait event information is supported for all versions of Aurora PostgreSQL.

Likely causes of increased waits

This wait event occurs when an application, stored function, or user issues a SQL statement that calls one of the following functions:

- `pg_sleep`
- `pg_sleep_for`
- `pg_sleep_until`

The preceding functions delay execution until the specified number of seconds have elapsed. For example, `SELECT pg_sleep(1)` pauses for 1 second. For more information, see [Delaying Execution](#) in the PostgreSQL documentation.

Actions

Identify the statement that was running the `pg_sleep` function. Determine if the use of the function is appropriate.

Best practices with Amazon Aurora PostgreSQL

This topic includes information on best practices and options for using or migrating data to an Amazon Aurora PostgreSQL DB cluster.

Fast failover with Amazon Aurora PostgreSQL

There are several things you can do to make a failover perform faster with Aurora PostgreSQL. This section discusses each of the following ways:

- Aggressively set TCP keepalives to ensure that longer running queries that are waiting for a server response will be stopped before the read timeout expires in the event of a failure.
- Set the Java DNS caching timeouts aggressively to ensure the Aurora read-only endpoint can properly cycle through read-only nodes on subsequent connection attempts.
- Set the timeout variables used in the JDBC connection string as low as possible. Use separate connection objects for short and long running queries.
- Use the provided read and write Aurora endpoints to establish a connection to the cluster.
- Use RDS APIs to test application response on server side failures and use a packet dropping tool to test application response for client-side failures.
- Use the AWS JDBC Driver for PostgreSQL (preview) to take full advantage of the failover capabilities of Aurora PostgreSQL. For more information about the AWS JDBC Driver for PostgreSQL and complete instructions for using it, see the [AWS JDBC Driver for PostgreSQL GitHub repository](#).

Contents

- [Setting TCP keepalives parameters \(p. 1424\)](#)
- [Configuring your application for fast failover \(p. 1425\)](#)
 - [Reducing DNS cache timeouts \(p. 1425\)](#)
 - [Setting an Aurora PostgreSQL connection string for fast failover \(p. 1425\)](#)
 - [Other options for obtaining the host string \(p. 1427\)](#)
 - [Java example to list instances using the DescribeDBClusters API \(p. 1428\)](#)
- [Testing failover \(p. 1428\)](#)
- [Fast failover Java example \(p. 1429\)](#)

Setting TCP keepalives parameters

The TCP keepalive process is simple: when you set up a TCP connection, you associate a set of timers. When the keepalive timer reaches zero, you send a keepalive probe packet. If you receive a reply to your keepalive probe, you can assume that the connection is still up and running.

Enabling TCP keepalive parameters and setting them aggressively ensures that if your client is no longer able to connect to the database, then any active connections are quickly closed. This action allows the application to react appropriately, such as by picking a new host to connect to.

You need to set the following TCP keepalive parameters:

- `tcp_keepalive_time` controls the time, in seconds, after which a keepalive packet is sent when no data has been sent by the socket (ACKs are not considered data). We recommend the following setting:

```
tcp_keepalive_time = 1
```

- `tcp_keepalive_intvl` controls the time, in seconds, between sending subsequent keepalive packets after the initial packet is sent (set using the `tcp_keepalive_time` parameter). We recommend the following setting:

```
tcp_keepalive_intvl = 1
```

- `tcp_keepalive_probes` is the number of unacknowledged keepalive probes that occur before the application is notified. We recommend the following setting:

```
tcp_keepalive_probes = 5
```

These settings should notify the application within five seconds when the database stops responding. A higher `tcp_keepalive_probes` value can be set if keepalive packets are often dropped within the

application's network. This subsequently increases the time it takes to detect an actual failure, but allows for more buffer in less reliable networks.

Setting TCP keepalive parameters on Linux

- When testing how to configure the TCP keepalive parameters, we recommend doing so via the command line with the following commands: This suggested configuration is system wide, meaning that it affects all other applications that create sockets with the SO_KEEPALIVE option on.

```
sudo sysctl net.ipv4.tcp_keepalive_time=1
sudo sysctl net.ipv4.tcp_keepalive_intvl=1
sudo sysctl net.ipv4.tcp_keepalive_probes=5
```

- After you've found a configuration that works for your application, persist these settings by adding the following lines to `/etc/sysctl.conf`, including any changes you made:

```
tcp_keepalive_time = 1
tcp_keepalive_intvl = 1
tcp_keepalive_probes = 5
```

For information on setting TCP keepalive parameters on Windows, see [Things you May want to know about TCP keepalive](#).

Configuring your application for fast failover

This section discusses several Aurora PostgreSQL specific configuration changes you can make. To learn more about PostgreSQL JDBC driver setup and configuration, see the [PostgreSQL JDBC Driver](#) documentation.

Topics

- [Reducing DNS cache timeouts \(p. 1425\)](#)
- [Setting an Aurora PostgreSQL connection string for fast failover \(p. 1425\)](#)
- [Other options for obtaining the host string \(p. 1427\)](#)

Reducing DNS cache timeouts

When your application tries to establish a connection after a failover, the new Aurora PostgreSQL writer will be a previous reader, which can be found using the Aurora **read only** endpoint before DNS updates have fully propagated. Setting the java DNS TTL to a low value helps cycle between reader nodes on subsequent connection attempts.

```
// Sets internal TTL to match the Aurora RO Endpoint TTL
java.security.Security.setProperty("networkaddress.cache.ttl" , "1");
// If the lookup fails, default to something like small to retry
java.security.Security.setProperty("networkaddress.cache.negative.ttl" , "3");
```

Setting an Aurora PostgreSQL connection string for fast failover

To make use of Aurora PostgreSQL fast failover, your application's connection string should have a list of hosts (highlighted in bold in the following example) instead of just a single host. Here is an example connection string you could use to connect to an Aurora PostgreSQL cluster:

```
jdbc:postgresql://myauroracluster.cluster-c9bfei4hj1rd.us-east-1-beta.rds.amazonaws.com:5432
myauroracluster.cluster-ro-c9bfei4hj1rd.us-east-1-beta.rds.amazonaws.com:5432
```

```
/postgres?user=<primaryuser>&password=<primarypw>&loginTimeout=2  
&connectTimeout=2&cancelSignalTimeout=2&socketTimeout=60  
&tcpKeepAlive=true&targetServerType=primary
```

For more information about PostgreSQL JDBC driver parameters, see [Connecting to the Database](#).

For best availability and to avoid a dependency on the RDS API, the best option for connecting is to maintain a file with a host string that your application reads from when you establish a connection to the database. This host string would have all the Aurora endpoints available for the cluster. For more information about Aurora endpoints, see [Amazon Aurora connection management \(p. 32\)](#). For example, you could store the endpoints in a file locally like the following:

```
myauroracluster.cluster-c9bfei4hj1rd.us-east-1-beta.rds.amazonaws.com:5432,  
myauroracluster.cluster-ro-c9bfei4hj1rd.us-east-1-beta.rds.amazonaws.com:5432
```

Your application would read from this file to populate the host section of the JDBC connection string. Renaming the DB cluster causes these endpoints to change; ensure that your application handles that event should it occur.

Another option is to use a list of DB instance nodes:

```
my-node1.cksc6xlmwcyw.us-east-1-beta.rds.amazonaws.com:5432,  
my-node2.cksc6xlmwcyw.us-east-1-beta.rds.amazonaws.com:5432,  
my-node3.cksc6xlmwcyw.us-east-1-beta.rds.amazonaws.com:5432,  
my-node4.cksc6xlmwcyw.us-east-1-beta.rds.amazonaws.com:5432
```

The benefit of this approach is that the PostgreSQL JDBC connection driver will loop through all nodes on this list to find a valid connection, whereas when using the Aurora endpoints only two nodes will be tried per connection attempt. The downside of using DB instance nodes is that if you add or remove nodes from your cluster and the list of instance endpoints becomes stale, the connection driver may never find the correct host to connect to.

Set the following parameters aggressively to help ensure that your application doesn't wait too long to connect to any one host.

- **targetServerType** – Use this parameter to control whether the driver connects to a write or read node. To ensure your applications will reconnect only to a write node, set the **targetServerType** value to **primary**.

Values for the **targetServerType** parameter include **primary**, **secondary**, **any**, and **preferSecondary**. The **preferSecondary** value attempts to establish a connection to a reader first but connects to the writer if no reader connection can be established.

- **loginTimeout** – Controls how long your application waits to login to the database *after* a socket connection has been established.
- **connectTimeout** – Controls how long the socket waits to establish a connection to the database.

You can modify other application parameters to speed up the connection process, depending on how aggressive you want your application to be.

- **cancelSignalTimeout** – In some applications, you may want to send a "best effort" cancel signal on a query that has timed out. If this cancel signal is in your failover path, you should consider setting it aggressively to avoid sending this signal to a dead host.
- **socketTimeout** – This parameter controls how long the socket waits for read operations. This parameter can be used as a global "query timeout" to ensure no query waits longer than this value. A good practice is to have one connection handler that runs short lived queries and sets this value lower, and to have another connection handler for long running queries with this value set much higher. Then, you can rely on TCP keepalive parameters to stop long running queries if the server goes down.

- `tcpKeepAlive` – Enable this parameter to ensure the TCP keepalive parameters that you set are respected.
- `loadBalanceHosts` – When set to `true`, this parameter has the application connect to a random host chosen from a list of candidate hosts.

Other options for obtaining the host string

You can get the host string from several sources, including the `aurora_replica_status` function and by using the Amazon RDS API.

Your application can connect to any DB instance in the DB cluster and query the `aurora_replica_status` function to determine who the writer of the cluster is, or to find any other reader nodes in the cluster. You can use this function to reduce the amount of time it takes to find a host to connect to, though in certain scenarios the `aurora_replica_status` function may show out of date or incomplete information in certain network failure scenarios.

A good way to ensure your application can find a node to connect to is to attempt to connect to the `cluster writerendpoint` and then the `cluster readerendpoint` until you can establish a readable connection. These endpoints do not change unless you rename your DB cluster, and thus can generally be left as static members of your application or stored in a resource file that your application reads from.

After you establish a connection using one of these endpoints, you can call the `aurora_replica_status` function to get information about the rest of the cluster. For example, the following command retrieves information with the `aurora_replica_status` function.

```
postgres=> SELECT server_id, session_id, highest_lsn_rcvd, cur_replay_latency_in_usec,
    now(), last_update_timestamp
  FROM aurora_replica_status();

server_id | session_id | highest_lsn_rcvd | cur_replay_latency_in_usec | now | last_update_timestamp
-----+-----+-----+-----+-----+-----+
mynode-1 | 3e3c5044-02e2-11e7-b70d-95172646d6ca | 594221001 | 201421 | 2017-03-07
19:50:24.695322+00 | 2017-03-07 19:50:23+00
mynode-2 | 1efdd188-02e4-11e7-becd-f12d7c88a28a | 594221001 | 201350 | 2017-03-07
19:50:24.695322+00 | 2017-03-07 19:50:23+00
mynode-3 | MASTER_SESSION_ID | | 2017-03-07 19:50:24.695322+00 | 2017-03-07 19:50:23+00
(3 rows)
```

So for example, the hosts section of your connection string could start with both the writer and reader cluster endpoints:

```
myauroracluster.cluster-c9bfei4hj1rd.us-east-1-beta.rds.amazonaws.com:5432,
myauroracluster.cluster-ro-c9bfei4hj1rd.us-east-1-beta.rds.amazonaws.com:5432
```

In this scenario, your application would attempt to establish a connection to any node type, primary or secondary. When your application is connected, a good practice is to first examine the read/write status of the node by querying for the result of the command `SHOW transaction_read_only`.

If the return value of the query is `OFF`, then you've successfully connected to the primary node. If the return value is `ON`, and your application requires a read/write connection, you can then call the `aurora_replica_status` function to determine the `server_id` that has `session_id='MASTER_SESSION_ID'`. This function gives you the name of the primary node. You can use this in conjunction with the 'endpointPostfix' described below.

One thing to be aware of is when you connect to a replica that has stale data. When this happens, the `aurora_replica_status` function might show out-of-date information. A threshold for staleness

can be set at the application level and examined by looking at the difference between the server time and the `last_update_timestamp`. In general, your application should avoid flipping between two hosts due to conflicting information returned by the `aurora_replica_status` function. Your application should try all known hosts first instead of blindly following the data returned by the `aurora_replica_status` function.

Java example to list instances using the `DescribeDBClusters` API

You can programmatically find the list of instances by using the [AWS SDK for Java](#), specifically the `DescribeDBClusters` API. Here's a small example of how you might do this in java 8:

```
AmazonRDS client = AmazonRDSClientBuilder.defaultClient();
DescribeDBClustersRequest request = new DescribeDBClustersRequest()
    .withDBClusterIdentifier(clusterName);
DescribeDBClustersResult result =
rdsClient.describeDBClusters(request);

DBCluster singleClusterResult = result.getDBClusters().get(0);

String pgJDBCEndpointStr =
singleClusterResult.getDBClusterMembers().stream()
    .sorted(Comparator.comparing(DBClusterMember::getIsClusterWriter)
    .reversed()) // This puts the writer at the front of the list
    .map(m -> m.getDBInstanceIdentifier() + endpointPostfix + ":" +
singleClusterResult.getPort())
    .collect(Collectors.joining(", "));
```

`pgJDBCEndpointStr` will contain a formatted list of endpoints. For example:

```
my-node1.cksc6xlmwcyw.us-east-1-beta.rds.amazonaws.com:5432,
my-node2.cksc6xlmwcyw.us-east-1-beta.rds.amazonaws.com:5432
```

The variable `endpointPostfix` can be a constant that your application sets, or can be obtained by querying the `DescribeDBInstances` API for a single instance in your cluster. This value remains constant within a region and for an individual customer, so it would save an API call to simply keep this constant in a resource file that your application reads from. In the example above, it would be set to:

```
.cksc6xlmwcyw.us-east-1-beta.rds.amazonaws.com
```

For availability purposes, a good practice is to default to using the Aurora endpoints of your DB cluster if the API is not responding, or is taking too long to respond. The endpoints are guaranteed to be up to date within the time it takes to update the DNS record. This is typically less than 30 seconds. You can store this in a resource file that your application consumes.

Testing failover

In all cases you must have a DB cluster with two or more DB instances in it.

From the server side, certain APIs can cause an outage that can be used to test how your applications responds:

- [FailoverDBCluster](#) - Will attempt to promote a new DB instance in your DB cluster to writer.

The following code sample shows how you can use `failoverDBCluster` to cause an outage. For more details about setting up an Amazon RDS client, see [Using the AWS SDK for Java](#).

```
public void causeFailover() {
```

```
final AmazonRDS rdsClient = AmazonRDSClientBuilder.defaultClient();

FailoverDBClusterRequest request = new FailoverDBClusterRequest();
request.setDBClusterIdentifier("cluster-identifier");

rdsClient.failoverDBCluster(request);
}
```

- [RebootDBInstance](#) – Failover is not guaranteed in this API. It will shutdown the database on the writer, though, and can be used to test how your application responds to connections dropping (note that the **ForceFailover** parameter is not applicable for Aurora engines and instead should use the [FailoverDBCluster API](#)).
- [ModifyDBCluster](#) – Modifying the **Port** will cause an outage when the nodes in the cluster begin listening on a new port. In general your application can respond to this failure by ensuring that only your application controls port changes and can appropriately update the endpoints it depends on, by having someone manually update the port when they make modifications at the API level, or by querying the RDS API in your application to determine if the port has changed.
- [ModifyDBInstance](#) – Modifying the **DBInstanceClass** will cause an outage.
- [DeleteDBInstance](#) – Deleting the primary/writer will cause a new DB instance to be promoted to writer in your DB cluster.

From the application/client side, if using Linux, you can test how the application responds to sudden packet drops based on port, host, or if tcp keepalive packets are not sent or received by using iptables.

Fast failover Java example

The following code sample shows how an application might set up an Aurora PostgreSQL driver manager. The application would call `getConnection()` when it needs a connection. A call to this function can fail to find a valid host, such as when no writer is found but the `targetServerType` parameter was set to `primary`. The calling application should simply retry calling the function. This can easily be wrapped into a connection pooler to avoid pushing the retry behavior onto the application. Most connection poolers allow you to specify a JDBC connection string, so your application could call into `getJdbcConnectionString()` and pass that to the connection pooler to make use of faster failover on Aurora PostgreSQL.

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.ArrayList;
import java.util.List;
import java.util.stream.Collectors;
import java.util.stream.IntStream;

import org.joda.time.Duration;

public class FastFailoverDriverManager {
    private static Duration LOGIN_TIMEOUT = Duration.standardSeconds(2);
    private static Duration CONNECT_TIMEOUT = Duration.standardSeconds(2);
    private static Duration CANCEL_SIGNAL_TIMEOUT = Duration.standardSeconds(1);
    private static Duration DEFAULT_SOCKET_TIMEOUT = Duration.standardSeconds(5);

    public FastFailoverDriverManager() {
        try {
            Class.forName("org.postgresql.Driver");
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        }
    }
}
```

```

/*
 * RO endpoint has a TTL of 1s, we should honor that here. Setting this
aggressively makes sure that when
    * the PG JDBC driver creates a new connection, it will resolve a new different RO
endpoint on subsequent attempts
        * (assuming there is > 1 read node in your cluster)
    */
java.security.Security.setProperty("networkaddress.cache.ttl" , "1");
// If the lookup fails, default to something like small to retry
java.security.Security.setProperty("networkaddress.cache.negative.ttl" , "3");
}

public Connection getConnection(String targetServerType) throws SQLException {
    return getConnection(targetServerType, DEFAULT_SOCKET_TIMEOUT);
}

public Connection getConnection(String targetServerType, Duration queryTimeout) throws
SQLException {
    Connection conn =
DriverManager.getConnection(getJdbcConnectionString(targetServerType, queryTimeout));

/*
 * A good practice is to set socket and statement timeout to be the same thing
since both
    * the client AND server will stop the query at the same time, leaving no running
queries
    * on the backend
*/
Statement st = conn.createStatement();
st.execute("set statement_timeout to " + queryTimeout.getMillis());
st.close();

    return conn;
}

private static String urlFormat = "jdbc:postgresql://%" +
+ "/postgres"
+ "?user=%s"
+ "&password=%s"
+ "&loginTimeout=%d"
+ "&connectTimeout=%d"
+ "&cancelSignalTimeout=%d"
+ "&socketTimeout=%d"
+ "&targetServerType=%s"
+ "&tcpKeepAlive=true"
+ "&ssl=true"
+ "&loadBalanceHosts=true";
public String getJdbcConnectionString(String targetServerType, Duration queryTimeout) {
    return String.format(urlFormat,
        getFormattedEndpointList(getLocalEndpointList()),
        CredentialManager.getUsername(),
        CredentialManager.getPassword(),
        LOGIN_TIMEOUT.getStandardSeconds(),
        CONNECT_TIMEOUT.getStandardSeconds(),
        CANCEL_SIGNAL_TIMEOUT.getStandardSeconds(),
        queryTimeout.getStandardSeconds(),
        targetServerType
    );
}

private List<String> getLocalEndpointList() {
/*
 * As mentioned in the best practices doc, a good idea is to read a local resource
file and parse the cluster endpoints.
 * For illustration purposes, the endpoint list is hardcoded here

```

```
        */
    List<String> newEndpointList = new ArrayList<>();
    newEndpointList.add("myauroracluster.cluster-c9bfei4hjlr.us-east-1-
beta.rds.amazonaws.com:5432");
    newEndpointList.add("myauroracluster.cluster-ro-c9bfei4hjlr.us-east-1-
beta.rds.amazonaws.com:5432");

    return newEndpointList;
}

private static String getFormattedEndpointList(List<String> endpoints) {
    return IntStream.range(0, endpoints.size())
        .mapToObj(i -> endpoints.get(i).toString())
        .collect(Collectors.joining(","));
}
```

Troubleshooting storage issues

If the amount of memory required by a sort or index creation operation exceeds the amount of memory available, Aurora PostgreSQL writes the excess data to storage. When it writes the data it uses the same storage space it uses for storing error and message logs. If your sorts or index creation functions exceed the memory available, you could develop a local storage shortage. If you experience issues with Aurora PostgreSQL running out of storage space, you can either reconfigure your data sorts to use more memory, or reduce the data retention period for your PostgreSQL log files. For more information about changing the log retention period see, [PostgreSQL database log files \(p. 706\)](#).

If your Aurora cluster is larger than 40 TB, don't use db.t2, db.t3, or db.t4g instance classes.

Replication with Amazon Aurora PostgreSQL

Following, you can find a description of replication with Amazon Aurora PostgreSQL, including how to monitor replication.

Topics

- [Using Aurora Replicas \(p. 1431\)](#)
- [Monitoring Aurora PostgreSQL replication \(p. 1432\)](#)
- [Using PostgreSQL logical replication with Aurora \(p. 1432\)](#)

Using Aurora Replicas

An *Aurora Replica* is an independent endpoint in an Aurora DB cluster, best used for scaling read operations and increasing availability. An Aurora DB cluster can include up to 15 Aurora Replicas located throughout the Availability Zones of the Aurora DB cluster's AWS Region.

The DB cluster volume is made up of multiple copies of the data for the DB cluster. However, the data in the cluster volume is represented as a single, logical volume to the primary writer DB instance and to Aurora Replicas in the DB cluster. For more information about Aurora Replicas, see [Aurora Replicas \(p. 70\)](#).

Aurora Replicas work well for read scaling because they're fully dedicated to read operations on your cluster volume. The writer DB instance manages write operations. The cluster volume is shared among all instances in your Aurora PostgreSQL DB cluster. Thus, no extra work is needed to replicate a copy of the data for each Aurora Replica.

With Aurora PostgreSQL, when an Aurora Replica is deleted, its instance endpoint is removed immediately, and the Aurora Replica is removed from the reader endpoint. If there are statements running on the Aurora Replica that is being deleted, there is a three minute grace period. Existing statements can finish gracefully during the grace period. When the grace period ends, the Aurora Replica is shut down and deleted.

Aurora PostgreSQL DB clusters don't support Aurora Replicas in different AWS Regions, so you can't use Aurora Replicas for cross-Region replication.

Note

Rebooting the writer DB instance of an Amazon Aurora DB cluster also automatically reboots the Aurora Replicas for that DB cluster. The automatic reboot re-establishes an entry point that guarantees read/write consistency across the DB cluster.

Monitoring Aurora PostgreSQL replication

Read scaling and high availability depend on minimal lag time. You can monitor how far an Aurora Replica is lagging behind the writer DB instance of your Aurora PostgreSQL DB cluster by monitoring the Amazon CloudWatch `ReplicaLag` metric. Because Aurora Replicas read from the same cluster volume as the writer DB instance, the `ReplicaLag` metric has a different meaning for an Aurora PostgreSQL DB cluster. The `ReplicaLag` metric for an Aurora Replica indicates the lag for the page cache of the Aurora Replica compared to that of the writer DB instance.

For more information on monitoring RDS instances and CloudWatch metrics, see [Monitoring metrics in an Amazon Aurora cluster \(p. 541\)](#).

Using PostgreSQL logical replication with Aurora

PostgreSQL logical replication provides fine-grained control over replicating and synchronizing parts of a database. For example, you can use logical replication to replicate an individual table of a database.

Following, you can find information about how to work with PostgreSQL logical replication and Amazon Aurora. For more detailed information about the PostgreSQL implementation of logical replication, see [Logical replication](#) and [Logical decoding concepts](#) in the PostgreSQL documentation.

Note

Logical replication is available with Aurora PostgreSQL version 2.2.0 (compatible with PostgreSQL 10.6) and later.

Following, you can find information about how to work with PostgreSQL logical replication and Amazon Aurora.

Topics

- [Configuring logical replication \(p. 1432\)](#)
- [Example of logical replication of a database table \(p. 1434\)](#)
- [Logical replication using the AWS Database Migration Service \(p. 1435\)](#)
- [Stopping logical replication \(p. 1437\)](#)

Configuring logical replication

To use logical replication, you first set the `rds.logical_replication` parameter for a cluster parameter group. You then set up the publisher and subscriber.

Logical replication uses a publish and subscribe model. *Publishers* and *subscribers* are the nodes. A *publication* is a set of changes generated from one or more database tables. You specify a publication on

a publisher. A *subscription* defines the connection to another database and one or more publications to which it subscribes. You specify a subscription on a subscriber. The publication and subscription make the connection between the publisher and subscriber.

Note

Following are requirements for logical replication:

- To perform logical replication for a PostgreSQL database, your AWS user account needs the `rds_superuser` role.
- The RDS for PostgreSQL DB instance that you use as the source must have automated backups enabled. For instructions on how to enable automated backups for an RDS for PostgreSQL DB instance, see [Enabling automated backups](#) in the *Amazon RDS User Guide*.

To enable PostgreSQL logical replication with Aurora

1. Create a new DB cluster parameter group to use for logical replication, as described in [Creating a DB cluster parameter group \(p. 343\)](#). Use the following settings:
 - For **Parameter group family**, choose your version of Aurora PostgreSQL, such as `aurora-postgresql12`.
 - For **Type**, choose **DB Cluster Parameter Group**.
2. Modify the DB cluster parameter group, as described in [Modifying parameters in a DB cluster parameter group \(p. 349\)](#). Set the `rds.logical_replication` static parameter to 1.

Enabling the `rds.logical_replication` parameter affects the DB cluster's performance.

3. Review the `max_replication_slots`, `max_wal_senders`, `max_logical_replication_workers`, and `max_worker_processes` parameters in your DB cluster parameter group based on your expected usage. If necessary, modify the DB cluster parameter group to change the settings for these parameters, as described in [Modifying parameters in a DB cluster parameter group \(p. 349\)](#).

Follow these guidelines for setting the parameters:

- `max_replication_slots` – Ensure that `max_replication_slots` is at least as high as the combined number of logical replication publications and subscriptions you plan to create. If you are using AWS DMS, make sure `max_replication_slots` is at least as high as the number of AWS DMS tasks you plan to use for change data capture from this DB cluster, plus any logical replication publications and subscriptions.
- `max_wal_senders` and `max_logical_replication_workers` – Ensure that `max_wal_senders` and `max_logical_replication_workers` are each set at least as high as the number of logical replication slots that you intend to be active, or the number of active AWS DMS tasks for change data capture. Leaving a logical replication slot inactive prevents vacuum from removing obsolete tuples from tables, so we recommend that you don't keep inactive replication slots for long periods of time.
- `max_worker_processes` – Ensure that `max_worker_processes` is at least as high as the combined values of `max_logical_replication_workers`, `autovacuum_max_workers`, and `max_parallel_workers`. Having a high number of background worker processes might affect application workloads on small DB instance classes, so monitor the performance of your database if you set `max_worker_processes` higher than the default value.

To configure a publisher for logical replication

1. Set the publisher's cluster parameter group:
 - To use an existing Aurora PostgreSQL DB cluster for the publisher, the engine version must be 10.6 or later. Do the following:

1. Modify the DB cluster parameter group to set it to the group that you created when you enabled logical replication. For details about modifying an Aurora PostgreSQL DB cluster, see [Modifying an Amazon Aurora DB cluster \(p. 372\)](#).
 2. Restart the DB cluster for static parameter changes to take effect. The DB cluster parameter group includes a change to the static parameter `rds.logical_replication`.
- To use a new Aurora PostgreSQL DB cluster for the publisher, create the DB cluster using the following settings. For details about creating an Aurora PostgreSQL DB cluster, see [Creating a DB cluster \(p. 126\)](#).
 1. Choose the **Amazon Aurora** engine and choose the **PostgreSQL-compatible** edition.
 2. For **Engine version**, choose an Aurora PostgreSQL engine that is compatible with PostgreSQL 10.6 or greater.
 3. For **DB cluster parameter group**, choose the group that you created when you enabled logical replication.
2. Modify the inbound rules of the security group for the publisher to allow the subscriber to connect. Usually, you do this by including the IP address of the subscriber in the security group. For details about modifying a security group, see [Security groups for your VPC](#) in the *Amazon Virtual Private Cloud User Guide*.

Example of logical replication of a database table

To implement logical replication, use the PostgreSQL commands `CREATE PUBLICATION` and `CREATE SUBSCRIPTION`.

For this example, table data is replicated from an Aurora PostgreSQL database as the publisher to a PostgreSQL database as the subscriber. Note that a subscriber database can be an RDS PostgreSQL database or an Aurora PostgreSQL database. A subscriber can also be an application that uses PostgreSQL logical replication. After the logical replication mechanism is set up, changes on the publisher are continually sent to the subscriber as they occur.

To set up logical replication for this example, do the following:

1. Configure an Aurora PostgreSQL DB cluster as the publisher. To do so, create a new Aurora PostgreSQL DB cluster, as described when configuring the publisher in [Configuring logical replication \(p. 1432\)](#).
2. Set up the publisher database.

For example, create a table using the following SQL statement on the publisher database.

```
CREATE TABLE LogicalReplicationTest (a int PRIMARY KEY);
```

3. Insert data into the publisher database by using the following SQL statement.

```
INSERT INTO LogicalReplicationTest VALUES (generate_series(1,10000));
```

4. Create a publication on the publisher by using the following SQL statement.

```
CREATE PUBLICATION testpub FOR TABLE LogicalReplicationTest;
```

5. Create your subscriber. A subscriber database can be either of the following:
 - Aurora PostgreSQL database version 2.2.0 (compatible with PostgreSQL 10.6) or later.
 - Amazon RDS for PostgreSQL database with the PostgreSQL DB engine version 10.4 or later.

For this example, we create an Amazon RDS for PostgreSQL database as the subscriber. For details on creating a DB instance, see [Creating a DB instance](#) in the *Amazon RDS User Guide*.

6. Set up the subscriber database.

For this example, create a table like the one created for the publisher by using the following SQL statement.

```
CREATE TABLE LogicalReplicationTest (a int PRIMARY KEY);
```

7. Verify that there is data in the table at the publisher but no data yet at the subscriber by using the following SQL statement on both databases.

```
SELECT count(*) FROM LogicalReplicationTest;
```

8. Create a subscription on the subscriber.

Use the following SQL statement on the subscriber database and the following settings from the publisher cluster:

- **host** – The publisher cluster's writer DB instance.
- **port** – The port on which the writer DB instance is listening. The default for PostgreSQL is 5432.
- **dbname** – The DB name of the publisher cluster.

```
CREATE SUBSCRIPTION testsub CONNECTION
  'host=publisher-cluster-writer-endpoint port=5432 dbname=db-name user=user
  password=password'
  PUBLICATION testpub;
```

After the subscription is created, a logical replication slot is created at the publisher.

9. To verify for this example that the initial data is replicated on the subscriber, use the following SQL statement on the subscriber database.

```
SELECT count(*) FROM LogicalReplicationTest;
```

Any further changes on the publisher are replicated to the subscriber.

Logical replication using the AWS Database Migration Service

You can use the AWS Database Migration Service (AWS DMS) to replicate a database or a portion of a database. Use AWS DMS to migrate your data from an Aurora PostgreSQL database to another open source or commercial database. For more information about AWS DMS, see the [AWS Database Migration Service User Guide](#).

The following example shows how to set up logical replication from an Aurora PostgreSQL database as the publisher and then use AWS DMS for migration. This example uses the same publisher and subscriber that were created in [Example of logical replication of a database table \(p. 1434\)](#).

To set up logical replication with AWS DMS, you need details about your publisher and subscriber from Amazon RDS. In particular, you need details about the publisher's writer DB instance and the subscriber's DB instance.

Get the following information for the publisher's writer DB instance:

- The virtual private cloud (VPC) identifier
- The subnet group
- The Availability Zone (AZ)
- The VPC security group

- The DB instance ID

Get the following information for the subscriber's DB instance:

- The DB instance ID
- The source engine

To use AWS DMS for logical replication with Aurora PostgreSQL

1. Prepare the publisher database to work with AWS DMS.

To do this, PostgreSQL 10.x and later databases require that you apply AWS DMS wrapper functions to the publisher database. For details on this and later steps, see the instructions in [Using PostgreSQL version 10.x and later as a source for AWS DMS](#) in the *AWS Database Migration Service User Guide*.

2. Sign in to the AWS Management Console and open the AWS DMS console at <https://console.aws.amazon.com/dms/v2>. At top right, choose the same AWS Region in which the publisher and subscriber are located.
3. Create an AWS DMS replication instance.

Choose values that are the same as for your publisher's writer DB instance. These include the following settings:

- For **VPC**, choose the same VPC as for the writer DB instance.
- For **Replication Subnet Group**, choose a subnet group with the same values as the writer DB instance. Create a new one if necessary.
- For **Availability zone**, choose the same zone as for the writer DB instance.
- For **VPC Security Group**, choose the same group as for the writer DB instance.

4. Create an AWS DMS endpoint for the source.

Specify the publisher as the source endpoint by using the following settings:

- For **Endpoint type**, choose **Source endpoint**.
- Choose **Select RDS DB Instance**.
- For **RDS Instance**, choose the DB identifier of the publisher's writer DB instance.
- For **Source engine**, choose **postgres**.

5. Create an AWS DMS endpoint for the target.

Specify the subscriber as the target endpoint by using the following settings:

- For **Endpoint type**, choose **Target endpoint**.
- Choose **Select RDS DB Instance**.
- For **RDS Instance**, choose the DB identifier of the subscriber DB instance.
- Choose a value for **Source engine**. For example, if the subscriber is an RDS PostgreSQL database, choose **postgres**. If the subscriber is an Aurora PostgreSQL database, choose **aurora-postgresql**.

6. Create an AWS DMS database migration task.

You use a database migration task to specify what database tables to migrate, to map data using the target schema, and to create new tables on the target database. At a minimum, use the following settings for **Task configuration**:

- For **Replication instance**, choose the replication instance that you created in an earlier step.
- For **Source database endpoint**, choose the publisher source that you created in an earlier step.

- For **Target database endpoint**, choose the subscriber target that you created in an earlier step.

The rest of the task details depend on your migration project. For more information about specifying all the details for DMS tasks, see [Working with AWS DMS tasks](#) in the *AWS Database Migration Service User Guide*.

After AWS DMS creates the task, it begins migrating data from the publisher to the subscriber.

Stopping logical replication

You can stop using logical replication.

To stop using logical replication

1. Drop all replication slots.

To drop all of the replication slots, connect to the publisher and run the following SQL command

```
SELECT pg_drop_replication_slot(slot_name) FROM pg_replication_slots
WHERE slot_name IN (SELECT slot_name FROM pg_replication_slots);
```

The replication slots can't be active when you run this command.

2. Modify the DB cluster parameter group associated with the publisher, as described in [Modifying parameters in a DB cluster parameter group \(p. 349\)](#). Set the `rds.logical_replication` static parameter to 0.
3. Restart the publisher DB cluster for the change to the `rds.logical_replication` static parameter to take effect.

Integrating Amazon Aurora PostgreSQL with other AWS services

Amazon Aurora integrates with other AWS services so that you can extend your Aurora PostgreSQL DB cluster to use additional capabilities in the AWS Cloud. Your Aurora PostgreSQL DB cluster can use AWS services to do the following:

- Quickly collect, view, and assess performance for your Aurora PostgreSQL DB instances with Amazon RDS Performance Insights. Performance Insights expands on existing Amazon RDS monitoring features to illustrate your database's performance and help you analyze any issues that affect it. With the Performance Insights dashboard, you can visualize the database load and filter the load by waits, SQL statements, hosts, or users. For more information about Performance Insights, see [Monitoring DB load with Performance Insights on Amazon Aurora \(p. 573\)](#).
- Automatically add or remove Aurora Replicas with Aurora Auto Scaling. For more information, see [Using Amazon Aurora Auto Scaling with Aurora replicas \(p. 427\)](#).
- Configure your Aurora PostgreSQL DB cluster to publish log data to Amazon CloudWatch Logs. CloudWatch Logs provide highly durable storage for your log records. With CloudWatch Logs, you can perform real-time analysis of the log data, and use CloudWatch to create alarms and view metrics. For more information, see [Publishing Aurora PostgreSQL logs to Amazon CloudWatch Logs \(p. 1487\)](#).
- Import data from an Amazon S3 bucket to an Aurora PostgreSQL DB cluster, or export data from an Aurora PostgreSQL DB cluster to an Amazon S3 bucket. For more information, see [Importing Amazon S3 data into an Aurora PostgreSQL DB cluster \(p. 1438\)](#) and [Exporting data from an Aurora PostgreSQL DB cluster to Amazon S3 \(p. 1450\)](#).

- Add machine learning-based predictions to database applications using the SQL language. Aurora machine learning uses a highly optimized integration between the Aurora database and the AWS machine learning (ML) services SageMaker and Amazon Comprehend. For more information, see [Using machine learning \(ML\) with Aurora PostgreSQL \(p. 1494\)](#).
- Invoke AWS Lambda functions from an Aurora PostgreSQL DB cluster. To do this, use the `aws_lambda` PostgreSQL extension provided with Aurora PostgreSQL. For more information, see [Invoking an AWS Lambda function from an Aurora PostgreSQL DB cluster \(p. 1517\)](#).
- For easy and efficient access to Oracle databases for Aurora PostgreSQL, you can use the PostgreSQL `oracle_fdw` extension, which provides a foreign data wrapper. For more information, see [Using the oracle_fdw extension to access foreign data in Aurora PostgreSQL \(p. 1527\)](#).
- Integrate queries from Amazon Redshift and Aurora PostgreSQL. For more information, see [Getting started with using federated queries to PostgreSQL](#) in the *Amazon Redshift Database Developer Guide*.

Importing Amazon S3 data into an Aurora PostgreSQL DB cluster

You can import data from Amazon S3 into a table belonging to an Aurora PostgreSQL DB cluster. To do this, you use the `aws_s3` PostgreSQL extension that Aurora PostgreSQL provides. Your database must be running PostgreSQL version 10.7 or higher to import from Amazon S3 into Aurora PostgreSQL.

For more information on storing data with Amazon S3, see [Create a bucket](#) in the *Amazon Simple Storage Service User Guide*. For instructions on how to upload a file to an Amazon S3 bucket, see [Add an object to a bucket](#) in the *Amazon Simple Storage Service User Guide*.

Topics

- [Overview of importing Amazon S3 data \(p. 1438\)](#)
- [Setting up access to an Amazon S3 bucket \(p. 1439\)](#)
- [Using the `aws_s3.table_import_from_s3` function to import Amazon S3 data \(p. 1444\)](#)
- [Function reference \(p. 1446\)](#)

Overview of importing Amazon S3 data

To import data stored in an Amazon S3 bucket to a PostgreSQL database table, follow these steps.

To import S3 data into Aurora PostgreSQL

1. Install the required PostgreSQL extensions. These include the `aws_s3` and `aws_commons` extensions. To do so, start `psql` and use the following command.

```
psql=> CREATE EXTENSION aws_s3 CASCADE;
NOTICE: installing required extension "aws_commons"
```

The `aws_s3` extension provides the [aws_s3.table_import_from_s3 \(p. 1446\)](#) function that you use to import Amazon S3 data. The `aws_commons` extension provides additional helper functions.

2. Identify the database table and Amazon S3 file to use.

The [aws_s3.table_import_from_s3 \(p. 1446\)](#) function requires the name of the PostgreSQL database table that you want to import data into. The function also requires that you identify the Amazon S3 file to import. To provide this information, take the following steps.

- a. Identify the PostgreSQL database table to put the data in. For example, the following is a sample t1 database table used in the examples for this topic.

```
psql=> CREATE TABLE t1 (col1 varchar(80), col2 varchar(80), col3 varchar(80));
```

- b. Get the following information to identify the Amazon S3 file that you want to import:

- Bucket name – A *bucket* is a container for Amazon S3 objects or files.
- File path – The file path locates the file in the Amazon S3 bucket.
- AWS Region – The AWS Region is the location of the Amazon S3 bucket. For example, if the S3 bucket is in the US East (N. Virginia) Region, use us-east-1. For a listing of AWS Region names and associated values, see [Regions and Availability Zones \(p. 11\)](#).

To find how to get this information, see [View an object](#) in the *Amazon Simple Storage Service User Guide*. You can confirm the information by using the AWS CLI command `aws s3 cp`. If the information is correct, this command downloads a copy of the Amazon S3 file.

```
aws s3 cp s3://sample_s3_bucket/sample_file_path ./
```

- c. Use the [aws_commons.create_s3_uri \(p. 1449\)](#) function to create an `aws_commons._s3_uri_1` structure to hold the Amazon S3 file information. You provide this `aws_commons._s3_uri_1` structure as a parameter in the call to the [aws_s3.table_import_from_s3 \(p. 1446\)](#) function.

For a psql example, see the following.

```
psql=> SELECT aws_commons.create_s3_uri(  
    'sample_s3_bucket',  
    'sample.csv',  
    'us-east-1'  
) AS s3_uri \gset
```

3. Provide permission to access the Amazon S3 file.

To import data from an Amazon S3 file, give the Aurora PostgreSQL DB cluster permission to access the Amazon S3 bucket the file is in. To do this, you use either an AWS Identity and Access Management (IAM) role or security credentials. For more information, see [Setting up access to an Amazon S3 bucket \(p. 1439\)](#).

4. Import the Amazon S3 data by calling the `aws_s3.table_import_from_s3` function.

After you complete the previous preparation tasks, use the [aws_s3.table_import_from_s3 \(p. 1446\)](#) function to import the Amazon S3 data. For more information, see [Using the aws_s3.table_import_from_s3 function to import Amazon S3 data \(p. 1444\)](#).

Setting up access to an Amazon S3 bucket

To import data from an Amazon S3 file, give the Aurora PostgreSQL DB cluster permission to access the Amazon S3 bucket containing the file. You provide access to an Amazon S3 bucket in one of two ways, as described in the following topics.

Topics

- [Using an IAM role to access an Amazon S3 bucket \(p. 1440\)](#)
- [Using security credentials to access an Amazon S3 bucket \(p. 1443\)](#)
- [Troubleshooting access to Amazon S3 \(p. 1444\)](#)

Using an IAM role to access an Amazon S3 bucket

Before you load data from an Amazon S3 file, give your Aurora PostgreSQL DB cluster permission to access the Amazon S3 bucket the file is in. This way, you don't have to manage additional credential information or provide it in the [aws_s3.table_import_from_s3 \(p. 1446\)](#) function call.

To do this, create an IAM policy that provides access to the Amazon S3 bucket. Create an IAM role and attach the policy to the role. Then assign the IAM role to your DB cluster.

To give an Aurora PostgreSQL DB cluster access to Amazon S3 through an IAM role

1. Create an IAM policy.

This policy provides the bucket and object permissions that allow your Aurora PostgreSQL DB cluster to access Amazon S3.

Include in the policy the following required actions to allow the transfer of files from an Amazon S3 bucket to Aurora PostgreSQL:

- s3:GetObject
- s3>ListBucket

Include in the policy the following resources to identify the Amazon S3 bucket and objects in the bucket. This shows the Amazon Resource Name (ARN) format for accessing Amazon S3.

- arn:aws:s3:::*your-s3-bucket*
- arn:aws:s3:::*your-s3-bucket*/*

For more information on creating an IAM policy for Aurora PostgreSQL, see [Creating and using an IAM policy for IAM database access \(p. 1746\)](#). See also [Tutorial: Create and attach your first customer managed policy](#) in the *IAM User Guide*.

The following AWS CLI command creates an IAM policy named `rds-s3-import-policy` with these options. It grants access to a bucket named `your-s3-bucket`.

Note

Note the Amazon Resource Name (ARN) of the policy returned by this command. You need the ARN when you attach the policy to an IAM role, in a subsequent step.

Example

For Linux, macOS, or Unix:

```
aws iam create-policy \
--policy-name rds-s3-import-policy \
--policy-document '{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "s3import",
            "Action": [
                "s3:GetObject",
                "s3>ListBucket"
            ],
            "Effect": "Allow",
            "Resource": [
                "arn:aws:s3:::your-s3-bucket",
                "arn:aws:s3:::your-s3-bucket/*"
            ]
        }
    ]
}'
```

```
        }
    ]'
}'
```

For Windows:

```
aws iam create-policy ^
--policy-name rds-s3-import-policy ^
--policy-document '{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "s3import",
            "Action": [
                "s3:GetObject",
                "s3>ListBucket"
            ],
            "Effect": "Allow",
            "Resource": [
                "arn:aws:s3:::your-s3-bucket",
                "arn:aws:s3:::your-s3-bucket/*"
            ]
        }
    ]
}'
```

2. Create an IAM role.

You do this so Aurora PostgreSQL can assume this IAM role to access your Amazon S3 buckets. For more information, see [Creating a role to delegate permissions to an IAM user](#) in the *IAM User Guide*.

We recommend using the `aws:SourceArn` and `aws:SourceAccount` global condition context keys in resource-based policies to limit the service's permissions to a specific resource. This is the most effective way to protect against the [confused deputy problem](#).

If you use both global condition context keys and the `aws:SourceArn` value contains the account ID, the `aws:SourceAccount` value and the account in the `aws:SourceArn` value must use the same account ID when used in the same policy statement.

- Use `aws:SourceArn` if you want cross-service access for a single resource.
- Use `aws:SourceAccount` if you want to allow any resource in that account to be associated with the cross-service use.

In the policy, be sure to use the `aws:SourceArn` global condition context key with the full ARN of the resource. The following example shows how to do so using the AWS CLI command to create a role named `rds-s3-import-role`.

Example

For Linux, macOS, or Unix:

```
aws iam create-role \
--role-name rds-s3-import-role \
--assume-role-policy-document '{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "rds.amazonaws.com"
            }
        }
    ]
}'
```

```
        },
        "Action": "sts:AssumeRole",
        "Condition": {
            "StringEquals": {
                "aws:SourceAccount": "111122223333",
                "aws:SourceArn": "arn:aws:rds:us-east-1:111122223333:db:dbname"
            }
        }
    ]
}'
```

For Windows:

```
aws iam create-role ^
--role-name rds-s3-import-role ^
--assume-role-policy-document '{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "rds.amazonaws.com"
            },
            "Action": "sts:AssumeRole",
            "Condition": {
                "StringEquals": {
                    "aws:SourceAccount": "111122223333",
                    "aws:SourceArn": "arn:aws:rds:us-east-1:111122223333:db:dbname"
                }
            }
        ]
}'
```

3. Attach the IAM policy that you created to the IAM role that you created.

The following AWS CLI command attaches the policy created in the previous step to the role named `rds-s3-import-role`. Replace `your-policy-arn` with the policy ARN that you noted in an earlier step.

Example

For Linux, macOS, or Unix:

```
aws iam attach-role-policy \
--policy-arn your-policy-arn \
--role-name rds-s3-import-role
```

For Windows:

```
aws iam attach-role-policy ^
--policy-arn your-policy-arn ^
--role-name rds-s3-import-role
```

4. Add the IAM role to the DB cluster.

You do so by using the AWS Management Console or AWS CLI, as described following.

Note

You can't associate an IAM role with an Aurora Serverless DB cluster. For more information, see [Using Amazon Aurora Serverless v1 \(p. 147\)](#).

Also, be sure the database you use doesn't have any restrictions noted in [Importing Amazon S3 data into an Aurora PostgreSQL DB cluster \(p. 1438\)](#).

Console

To add an IAM role for a PostgreSQL DB cluster using the console

1. Sign in to the AWS Management Console and open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. Choose the PostgreSQL DB cluster name to display its details.
3. On the **Connectivity & security** tab, in the **Manage IAM roles** section, choose the role to add under **Add IAM roles to this cluster**.
4. Under **Feature**, choose **s3Import**.
5. Choose **Add role**.

AWS CLI

To add an IAM role for a PostgreSQL DB cluster using the CLI

- Use the following command to add the role to the PostgreSQL DB cluster named `my-db-cluster`. Replace `your-role-arn` with the role ARN that you noted in a previous step. Use `s3Import` for the value of the `--feature-name` option.

Example

For Linux, macOS, or Unix:

```
aws rds add-role-to-db-cluster \
  --db-cluster-identifier my-db-cluster \
  --feature-name s3Import \
  --role-arn your-role-arn \
  --region your-region
```

For Windows:

```
aws rds add-role-to-db-cluster ^
  --db-cluster-identifier my-db-cluster ^
  --feature-name s3Import ^
  --role-arn your-role-arn ^
  --region your-region
```

RDS API

To add an IAM role for a PostgreSQL DB cluster using the Amazon RDS API, call the [AddRoleToDBCluster](#) operation.

Using security credentials to access an Amazon S3 bucket

If you prefer, you can use security credentials to provide access to an Amazon S3 bucket instead of providing access with an IAM role. To do this, use the `credentials` parameter in the [aws_s3.table_import_from_s3 \(p. 1446\)](#) function call.

The `credentials` parameter is a structure of type `aws_commons._aws_credentials_1`, which contains AWS credentials. Use the [aws_commons.create_aws_credentials \(p. 1449\)](#) function to set the access key and secret key in an `aws_commons._aws_credentials_1` structure, as shown following.

```
psql=> SELECT aws_commons.create_aws_credentials(
    'sample_access_key', 'sample_secret_key', '')
AS creds \gset
```

After creating the `aws_commons._aws_credentials_1` structure, use the [aws_s3.table_import_from_s3 \(p. 1446\)](#) function with the `credentials` parameter to import the data, as shown following.

```
psql=> SELECT aws_s3.table_import_from_s3(
    't', '', '(format csv)',
    :'s3_uri',
    :'creds'
);
```

Or you can include the [aws_commons.create_aws_credentials \(p. 1449\)](#) function call inline within the `aws_s3.table_import_from_s3` function call.

```
psql=> SELECT aws_s3.table_import_from_s3(
    't', '', '(format csv)',
    :'s3_uri',
    aws_commons.create_aws_credentials('sample_access_key', 'sample_secret_key', '')
);
```

Troubleshooting access to Amazon S3

If you encounter connection problems when attempting to import Amazon S3 file data, see the following for recommendations:

- [Troubleshooting Amazon Aurora identity and access \(p. 1769\)](#)
- [Troubleshooting Amazon S3 in the Amazon Simple Storage Service User Guide](#)
- [Troubleshooting Amazon S3 and IAM in the IAM User Guide](#)

Using the `aws_s3.table_import_from_s3` function to import Amazon S3 data

Import your Amazon S3 data by calling the [aws_s3.table_import_from_s3 \(p. 1446\)](#) function.

Note

The following examples use the IAM role method for providing access to the Amazon S3 bucket. Thus, the `aws_s3.table_import_from_s3` function calls don't include credential parameters.

The following shows a typical PostgreSQL example using `psql`.

```
psql=> SELECT aws_s3.table_import_from_s3(
    't1',
    '',
    '(format csv)',
    :'s3_uri'
);
```

The parameters are the following:

- `t1` – The name for the table in the PostgreSQL DB cluster to copy the data into.
- `''` – An optional list of columns in the database table. You can use this parameter to indicate which columns of the S3 data go in which table columns. If no columns are specified, all the columns are copied to the table. For an example of using a column list, see [Importing an Amazon S3 file that uses a custom delimiter \(p. 1445\)](#).
- `(format csv)` – PostgreSQL COPY arguments. The copy process uses the arguments and format of the [PostgreSQL COPY](#) command. In the preceding example, the COPY command uses the comma-separated value (CSV) file format to copy the data.
- `s3_uri` – A structure that contains the information identifying the Amazon S3 file. For an example of using the `aws_commons.create_s3_uri (p. 1449)` function to create an `s3_uri` structure, see [Overview of importing Amazon S3 data \(p. 1438\)](#).

For more information about this function, see [aws_s3.table_import_from_s3 \(p. 1446\)](#).

The `aws_s3.table_import_from_s3` function returns text. To specify other kinds of files for import from an Amazon S3 bucket, see one of the following examples.

Topics

- [Importing an Amazon S3 file that uses a custom delimiter \(p. 1445\)](#)
- [Importing an Amazon S3 compressed \(gzip\) file \(p. 1446\)](#)
- [Importing an encoded Amazon S3 file \(p. 1446\)](#)

Importing an Amazon S3 file that uses a custom delimiter

The following example shows how to import a file that uses a custom delimiter. It also shows how to control where to put the data in the database table using the `column_list` parameter of the [aws_s3.table_import_from_s3 \(p. 1446\)](#) function.

For this example, assume that the following information is organized into pipe-delimited columns in the Amazon S3 file.

```
1|foo1|bar1|elephant1
2|foo2|bar2|elephant2
3|foo3|bar3|elephant3
4|foo4|bar4|elephant4
...
```

To import a file that uses a custom delimiter

1. Create a table in the database for the imported data.

```
psql=> CREATE TABLE test (a text, b text, c text, d text, e text);
```

2. Use the following form of the [aws_s3.table_import_from_s3 \(p. 1446\)](#) function to import data from the Amazon S3 file.

You can include the `aws_commons.create_s3_uri (p. 1449)` function call inline within the `aws_s3.table_import_from_s3` function call to specify the file.

```
psql=> SELECT aws_s3.table_import_from_s3(
    'test',
    'a,b,d,e',
    'DELIMITER ''|''',
    aws_commons.create_s3_uri('sampleBucket', 'pipeDelimitedSampleFile', 'us-east-2')
);
```

The data is now in the table in the following columns.

```
psql=> SELECT * FROM test;
a | b | c | d | e
---+---+---+---+---+
1 | foo1 | bar1 | elephant1
2 | foo2 | bar2 | elephant2
3 | foo3 | bar3 | elephant3
4 | foo4 | bar4 | elephant4
```

Importing an Amazon S3 compressed (gzip) file

The following example shows how to import a file from Amazon S3 that is compressed with gzip. The file that you import needs to have the following Amazon S3 metadata:

- Key: Content-Encoding
- Value: gzip

If you upload the file using the AWS Management Console, the metadata is typically applied by the system. For information about uploading files to Amazon S3 using the AWS Management Console, the AWS CLI, or the API, see [Uploading objects in the Amazon Simple Storage Service User Guide](#).

For more information about Amazon S3 metadata and details about system-provided metadata, see [Editing object metadata in the Amazon S3 console](#) in the *Amazon Simple Storage Service User Guide*.

Import the gzip file into your Aurora PostgreSQL DB cluster as shown following.

```
psql=> CREATE TABLE test_gzip(id int, a text, b text, c text, d text);
psql=> SELECT aws_s3.table_import_from_s3(
  'test_gzip', '', '(format csv)',
  'myS3Bucket', 'test-data.gz', 'us-east-2'
);
```

Importing an encoded Amazon S3 file

The following example shows how to import a file from Amazon S3 that has Windows-1252 encoding.

```
psql=> SELECT aws_s3.table_import_from_s3(
  'test_table', '', 'encoding ''WIN1252'''',
  aws_commons.create_s3_uri('sampleBucket', 'SampleFile', 'us-east-2')
);
```

Function reference

Functions

- [aws_s3.table_import_from_s3 \(p. 1446\)](#)
- [aws_commons.create_s3_uri \(p. 1449\)](#)
- [aws_commons.create_aws_credentials \(p. 1449\)](#)

[aws_s3.table_import_from_s3](#)

Imports Amazon S3 data into an Aurora PostgreSQL table. The `aws_s3` extension provides the `aws_s3.table_import_from_s3` function. The return value is text.

Syntax

The required parameters are `table_name`, `column_list` and `options`. These identify the database table and specify how the data is copied into the table.

You can also use the following parameters:

- The `s3_info` parameter specifies the Amazon S3 file to import. When you use this parameter, access to Amazon S3 is provided by an IAM role for the PostgreSQL DB cluster.

```
aws_s3.table_import_from_s3 (
    table_name text,
    column_list text,
    options text,
    s3_info aws_commons._s3_uri_1
)
```

- The `credentials` parameter specifies the credentials to access Amazon S3. When you use this parameter, you don't use an IAM role.

```
aws_s3.table_import_from_s3 (
    table_name text,
    column_list text,
    options text,
    s3_info aws_commons._s3_uri_1,
    credentials aws_commons._aws_credentials_1
)
```

Parameters

table_name

A required text string containing the name of the PostgreSQL database table to import the data into.

column_list

A required text string containing an optional list of the PostgreSQL database table columns in which to copy the data. If the string is empty, all columns of the table are used. For an example, see [Importing an Amazon S3 file that uses a custom delimiter \(p. 1445\)](#).

options

A required text string containing arguments for the PostgreSQL `COPY` command. These arguments specify how the data is to be copied into the PostgreSQL table. For more details, see the [PostgreSQL COPY documentation](#).

s3_info

An `aws_commons._s3_uri_1` composite type containing the following information about the S3 object:

- `bucket` – The name of the Amazon S3 bucket containing the file.
- `file_path` – The Amazon S3 file name including the path of the file.
- `region` – The AWS Region that the file is in. For a listing of AWS Region names and associated values, see [Regions and Availability Zones \(p. 11\)](#).

credentials

An `aws_commons._aws_credentials_1` composite type containing the following credentials to use for the import operation:

- Access key

- Secret key
- Session token

For information about creating an `aws_commons._aws_credentials_1` composite structure, see [aws_commons.create_aws_credentials \(p. 1449\)](#).

Alternate syntax

To help with testing, you can use an expanded set of parameters instead of the `s3_info` and `credentials` parameters. Following are additional syntax variations for the `aws_s3.table_import_from_s3` function:

- Instead of using the `s3_info` parameter to identify an Amazon S3 file, use the combination of the `bucket`, `file_path`, and `region` parameters. With this form of the function, access to Amazon S3 is provided by an IAM role on the PostgreSQL DB instance.

```
aws_s3.table_import_from_s3 (
    table_name text,
    column_list text,
    options text,
    bucket text,
    file_path text,
    region text
)
```

- Instead of using the `credentials` parameter to specify Amazon S3 access, use the combination of the `access_key`, `session_key`, and `session_token` parameters.

```
aws_s3.table_import_from_s3 (
    table_name text,
    column_list text,
    options text,
    bucket text,
    file_path text,
    region text,
    access_key text,
    secret_key text,
    session_token text
)
```

Alternate parameters

bucket

A text string containing the name of the Amazon S3 bucket that contains the file.

file_path

A text string containing the Amazon S3 file name including the path of the file.

region

A text string containing the AWS Region that the file is in. For a listing of AWS Region names and associated values, see [Regions and Availability Zones \(p. 11\)](#).

access_key

A text string containing the access key to use for the import operation. The default is NULL.

secret_key

A text string containing the secret key to use for the import operation. The default is NULL.

session_token

(Optional) A text string containing the session key to use for the import operation. The default is NULL.

aws_commons.create_s3_uri

Creates an `aws_commons._s3_uri_1` structure to hold Amazon S3 file information. Use the results of the `aws_commons.create_s3_uri` function in the `s3_info` parameter of the [aws_s3.table_import_from_s3 \(p. 1446\)](#) function.

Syntax

```
aws_commons.create_s3_uri(  
    bucket text,  
    file_path text,  
    region text  
)
```

Parameters

bucket

A required text string containing the Amazon S3 bucket name for the file.

file_path

A required text string containing the Amazon S3 file name including the path of the file.

region

A required text string containing the AWS Region that the file is in. For a listing of AWS Region names and associated values, see [Regions and Availability Zones \(p. 11\)](#).

aws_commons.create_aws_credentials

Sets an access key and secret key in an `aws_commons._aws_credentials_1` structure. Use the results of the `aws_commons.create_aws_credentials` function in the `credentials` parameter of the [aws_s3.table_import_from_s3 \(p. 1446\)](#) function.

Syntax

```
aws_commons.create_aws_credentials(  
    access_key text,  
    secret_key text,  
    session_token text  
)
```

Parameters

access_key

A required text string containing the access key to use for importing an Amazon S3 file. The default is NULL.

secret_key

A required text string containing the secret key to use for importing an Amazon S3 file. The default is NULL.

session_token

An optional text string containing the session token to use for importing an Amazon S3 file. The default is NULL. If you provide an optional *session_token*, you can use temporary credentials.

Exporting data from an Aurora PostgreSQL DB cluster to Amazon S3

You can query data from an Aurora PostgreSQL DB cluster and export it directly into files stored in an Amazon S3 bucket. To do this, you use the `aws_s3` PostgreSQL extension that Aurora PostgreSQL provides.

For more information on storing data with Amazon S3, see [Create a bucket](#) in the *Amazon Simple Storage Service User Guide*.

The upload to Amazon S3 uses server-side encryption by default. If you are using encryption, the Amazon S3 bucket must be encrypted with an AWS managed key. Currently, you can't export data to a bucket that is encrypted with a customer managed key.

Note

You can save DB and DB cluster snapshot data to Amazon S3 using the AWS Management Console, AWS CLI, or Amazon RDS API. For more information, see [Exporting DB snapshot data to Amazon S3 \(p. 518\)](#).

Topics

- [Overview of exporting data to Amazon S3 \(p. 1450\)](#)
- [Verify that your Aurora PostgreSQL version supports exports \(p. 1451\)](#)
- [Specifying the Amazon S3 file path to export to \(p. 1451\)](#)
- [Setting up access to an Amazon S3 bucket \(p. 1452\)](#)
- [Exporting query data using the `aws_s3.query_export_to_s3` function \(p. 1455\)](#)
- [Troubleshooting access to Amazon S3 \(p. 1457\)](#)
- [Function reference \(p. 1457\)](#)

Overview of exporting data to Amazon S3

To export data stored in an Aurora PostgreSQL database to an Amazon S3 bucket, use the following procedure.

To export Aurora PostgreSQL data to S3

1. Install the required PostgreSQL extensions. These include the `aws_s3` and `aws_commons` extensions. To do so, start `psql` and use the following commands.

```
CREATE EXTENSION IF NOT EXISTS aws_s3 CASCADE;
```

The `aws_s3` extension provides the [aws_s3.query_export_to_s3 \(p. 1458\)](#) function that you use to export data to Amazon S3. The `aws_commons` extension is included to provide additional helper functions.

2. Identify an Amazon S3 file path to use for exporting data. For details about this process, see [Specifying the Amazon S3 file path to export to \(p. 1451\)](#).
3. Provide permission to access the Amazon S3 bucket.

To export data to an Amazon S3 file, give the Aurora PostgreSQL DB cluster permission to access the Amazon S3 bucket that the export will use for storage. Doing this includes the following steps:

1. Create an IAM policy that provides access to an Amazon S3 bucket that you want to export to.
2. Create an IAM role.
3. Attach the policy you created to the role you created.
4. Add this IAM role to your DB cluster .

For details about this process, see [Setting up access to an Amazon S3 bucket \(p. 1452\)](#).

4. Identify a database query to get the data. Export the query data by calling the `aws_s3.query_export_to_s3` function.

After you complete the preceding preparation tasks, use the [aws_s3.query_export_to_s3 \(p. 1458\)](#) function to export query results to Amazon S3. For details about this process, see [Exporting query data using the aws_s3.query_export_to_s3 function \(p. 1455\)](#).

Verify that your Aurora PostgreSQL version supports exports

Currently, Amazon S3 exports are supported for the following versions of Aurora PostgreSQL:

- 10.11 and higher 10 versions
- 11.6 and higher 11 versions
- 12.4 and higher 12 versions
- 13.3 and higher 13 versions

You can also verify support by using the `describe-db-engine-versions` command. The following example verify support for version 10.14.

```
aws rds describe-db-engine-versions --region us-east-1 \
--engine aurora-postgresql --engine-version 10.14 | grep s3Export
```

If the output includes the string "s3Export", then the engine supports Amazon S3 exports. Otherwise, the engine doesn't support them.

Specifying the Amazon S3 file path to export to

Specify the following information to identify the location in Amazon S3 where you want to export data to:

- Bucket name – A *bucket* is a container for Amazon S3 objects or files.

For more information on storing data with Amazon S3, see [Create a bucket](#) and [View an object](#) in the [Amazon Simple Storage Service User Guide](#).

- File path – The file path identifies where the export is stored in the Amazon S3 bucket. The file path consists of the following:
 - An optional path prefix that identifies a virtual folder path.
 - A file prefix that identifies one or more files to be stored. Larger exports are stored in multiple files, each with a maximum size of approximately 6 GB. The additional file names have the same file prefix but with `_partXX` appended. The `XX` represents 2, then 3, and so on.

For example, a file path with an `exports` folder and a `query-1-export` file prefix is `/exports/query-1-export`.

- AWS Region (optional) – The AWS Region where the Amazon S3 bucket is located. If you don't specify an AWS Region value, then Aurora saves your files into Amazon S3 in the same AWS Region as the exporting DB cluster.

Note

Currently, the AWS Region must be the same as the region of the exporting DB cluster.

For a listing of AWS Region names and associated values, see [Regions and Availability Zones \(p. 11\)](#).

To hold the Amazon S3 file information about where the export is to be stored, you can use the [aws_commons.create_s3_uri \(p. 1460\)](#) function to create an `aws_commons._s3_uri_1` composite structure as follows.

```
psql=> SELECT aws_commons.create_s3_uri(
    'sample-bucket',
    'samplefilepath',
    'us-west-2'
) AS s3_uri_1 \gset
```

You later provide this `s3_uri_1` value as a parameter in the call to the [aws_s3.query_export_to_s3 \(p. 1458\)](#) function. For examples, see [Exporting query data using the aws_s3.query_export_to_s3 function \(p. 1455\)](#).

Setting up access to an Amazon S3 bucket

To export data to Amazon S3, give your PostgreSQL DB cluster permission to access the Amazon S3 bucket that the files are to go in.

To do this, use the following procedure.

To give a PostgreSQL DB cluster access to Amazon S3 through an IAM role

1. Create an IAM policy.

This policy provides the bucket and object permissions that allow your PostgreSQL DB cluster to access Amazon S3.

As part of creating this policy, take the following steps:

- a. Include in the policy the following required actions to allow the transfer of files from your PostgreSQL DB cluster to an Amazon S3 bucket:
 - `s3:PutObject`
 - `s3:AbortMultipartUpload`
- b. Include the Amazon Resource Name (ARN) that identifies the Amazon S3 bucket and objects in the bucket. The ARN format for accessing Amazon S3 is: `arn:aws:s3:::your-s3-bucket/*`

For more information on creating an IAM policy for Aurora PostgreSQL, see [Creating and using an IAM policy for IAM database access \(p. 1746\)](#). See also [Tutorial: Create and attach your first customer managed policy](#) in the *IAM User Guide*.

The following AWS CLI command creates an IAM policy named `rds-s3-export-policy` with these options. It grants access to a bucket named `your-s3-bucket`.

Warning

We recommend that you set up your database within a private VPC that has endpoint policies configured for accessing specific buckets. For more information, see [Using endpoint policies for Amazon S3](#) in the Amazon VPC User Guide.

We strongly recommend that you do not create a policy with all-resource access. This access can pose a threat for data security. If you create a policy that gives S3:PutObject access to all resources using "Resource" : "*", then a user with export privileges can export data to all buckets in your account. In addition, the user can export data to *any publicly writable bucket within your AWS Region*.

After you create the policy, note the Amazon Resource Name (ARN) of the policy. You need the ARN for a subsequent step when you attach the policy to an IAM role.

```
aws iam create-policy --policy-name rds-s3-export-policy --policy-document '{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "s3export",  
            "Action": [  
                "S3:PutObject"  
            ],  
            "Effect": "Allow",  
            "Resource": [  
                "arn:aws:s3:::your-s3-bucket/*"  
            ]  
        }  
    ]  
}'
```

2. Create an IAM role.

You do this so Aurora PostgreSQL can assume this IAM role on your behalf to access your Amazon S3 buckets. For more information, see [Creating a role to delegate permissions to an IAM user](#) in the *IAM User Guide*.

We recommend using the `aws:SourceArn` and `aws:SourceAccount` global condition context keys in resource-based policies to limit the service's permissions to a specific resource. This is the most effective way to protect against the [confused deputy problem](#).

If you use both global condition context keys and the `aws:SourceArn` value contains the account ID, the `aws:SourceAccount` value and the account in the `aws:SourceArn` value must use the same account ID when used in the same policy statement.

- Use `aws:SourceArn` if you want cross-service access for a single resource.
- Use `aws:SourceAccount` if you want to allow any resource in that account to be associated with the cross-service use.

In the policy, be sure to use the `aws:SourceArn` global condition context key with the full ARN of the resource. The following example shows how to do so using the AWS CLI command to create a role named `rds-s3-export-role`.

Example

For Linux, macOS, or Unix:

```
aws iam create-role \  
    --role-name rds-s3-export-role \  
    --assume-role-policy-document '{
```

```
"Version": "2012-10-17",
"Statement": [
    {
        "Effect": "Allow",
        "Principal": {
            "Service": "rds.amazonaws.com"
        },
        "Action": "sts:AssumeRole",
        "Condition": {
            "StringEquals": {
                "aws:SourceAccount": "111122223333",
                "aws:SourceArn": "arn:aws:rds:us-east-1:111122223333:db:dbname"
            }
        }
    }
]
```

For Windows:

```
aws iam create-role ^
--role-name rds-s3-export-role ^
--assume-role-policy-document '{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "rds.amazonaws.com"
            },
            "Action": "sts:AssumeRole",
            "Condition": {
                "StringEquals": {
                    "aws:SourceAccount": "111122223333",
                    "aws:SourceArn": "arn:aws:rds:us-east-1:111122223333:db:dbname"
                }
            }
        }
    ]
}'
```

3. Attach the IAM policy that you created to the IAM role that you created.

The following AWS CLI command attaches the policy created earlier to the role named `rds-s3-export-role`. Replace `your-policy-arn` with the policy ARN that you noted in an earlier step.

```
aws iam attach-role-policy --policy-arn your-policy-arn --role-name rds-s3-export-role
```

4. Add the IAM role to the DB cluster. You do so by using the AWS Management Console or AWS CLI, as described following.

Console

To add an IAM role for a PostgreSQL DB cluster using the console

1. Sign in to the AWS Management Console and open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. Choose the PostgreSQL DB cluster name to display its details.

3. On the **Connectivity & security** tab, in the **Manage IAM roles** section, choose the role to add under **Add IAM roles to this instance**.
4. Under **Feature**, choose **s3Export**.
5. Choose **Add role**.

AWS CLI

To add an IAM role for a PostgreSQL DB cluster using the CLI

- Use the following command to add the role to the PostgreSQL DB cluster named `my-db-cluster`. Replace `your-role-arn` with the role ARN that you noted in a previous step. Use `s3Export` for the value of the `--feature-name` option.

Example

For Linux, macOS, or Unix:

```
aws rds add-role-to-db-cluster \
--db-cluster-identifier my-db-cluster \
--feature-name s3Export \
--role-arn your-role-arn \
--region your-region
```

For Windows:

```
aws rds add-role-to-db-cluster ^
--db-cluster-identifier my-db-cluster ^
--feature-name s3Export ^
--role-arn your-role-arn ^
--region your-region
```

Exporting query data using the `aws_s3.query_export_to_s3` function

Export your PostgreSQL data to Amazon S3 by calling the [aws_s3.query_export_to_s3](#) (p. 1458) function.

Topics

- [Prerequisites](#) (p. 1455)
- [Calling aws_s3.query_export_to_s3](#) (p. 1456)
- [Exporting to a CSV file that uses a custom delimiter](#) (p. 1457)
- [Exporting to a binary file with encoding](#) (p. 1457)

Prerequisites

Before you use the `aws_s3.query_export_to_s3` function, be sure to complete the following prerequisites:

- Install the required PostgreSQL extensions as described in [Overview of exporting data to Amazon S3](#) (p. 1450).

- Determine where to export your data to Amazon S3 as described in [Specifying the Amazon S3 file path to export to \(p. 1451\)](#).
- Make sure that the DB cluster has export access to Amazon S3 as described in [Setting up access to an Amazon S3 bucket \(p. 1452\)](#).

The examples following use a database table called sample_table. These examples export the data into a bucket called sample-bucket. The example table and data are created with the following SQL statements in psql.

```
psql=> CREATE TABLE sample_table (bid bigint PRIMARY KEY, name varchar(80));
psql=> INSERT INTO sample_table (bid, name) VALUES (1, 'Monday'), (2, 'Tuesday'), (3, 'Wednesday');
```

Calling aws_s3.query_export_to_s3

The following shows the basic ways of calling the [aws_s3.query_export_to_s3 \(p. 1458\)](#) function.

These examples use the variable s3_uri_1 to identify a structure that contains the information identifying the Amazon S3 file. Use the [aws_commons.create_s3_uri \(p. 1460\)](#) function to create the structure.

```
psql=> SELECT aws_commons.create_s3_uri(
    'sample-bucket',
    'samplefilepath',
    'us-west-2'
) AS s3_uri_1 \gset
```

Although the parameters vary for the following two aws_s3.query_export_to_s3 function calls, the results are the same for these examples. All rows of the sample_table table are exported into a bucket called sample-bucket.

```
psql=> SELECT * FROM aws_s3.query_export_to_s3('SELECT * FROM sample_table', :'s3_uri_1');
psql=> SELECT * FROM aws_s3.query_export_to_s3('SELECT * FROM sample_table', :'s3_uri_1',
options :='format text');
```

The parameters are described as follows:

- 'SELECT * FROM sample_table' – The first parameter is a required text string containing an SQL query. The PostgreSQL engine runs this query. The results of the query are copied to the S3 bucket identified in other parameters.
- ':s3_uri_1' – This parameter is a structure that identifies the Amazon S3 file. This example uses a variable to identify the previously created structure. You can instead create the structure by including the `aws_commons.create_s3_uri` function call inline within the `aws_s3.query_export_to_s3` function call as follows.

```
SELECT * from aws_s3.query_export_to_s3('select * from sample_table',
aws_commons.create_s3_uri('sample-bucket', 'samplefilepath', 'us-west-2')
);
```

- `options :='format text'` – The `options` parameter is an optional text string containing PostgreSQL `COPY` arguments. The copy process uses the arguments and format of the [PostgreSQL COPY command](#).

If the file specified doesn't exist in the Amazon S3 bucket, it's created. If the file already exists, it's overwritten. The syntax for accessing the exported data in Amazon S3 is the following.

```
s3-region://bucket-name[/path-prefix]/file-prefix
```

Larger exports are stored in multiple files, each with a maximum size of approximately 6 GB. The additional file names have the same file prefix but with _part_{XX} appended. The _{XX} represents 2, then 3, and so on. For example, suppose that you specify the path where you store data files as the following.

```
s3-us-west-2://my-bucket/my-prefix
```

If the export has to create three data files, the Amazon S3 bucket contains the following data files.

```
s3-us-west-2://my-bucket/my-prefix
s3-us-west-2://my-bucket/my-prefix_part2
s3-us-west-2://my-bucket/my-prefix_part3
```

For the full reference for this function and additional ways to call it, see [aws_s3.query_export_to_s3 \(p. 1458\)](#). For more about accessing files in Amazon S3, see [View an object](#) in the *Amazon Simple Storage Service User Guide*.

Exporting to a CSV file that uses a custom delimiter

The following example shows how to call the [aws_s3.query_export_to_s3 \(p. 1458\)](#) function to export data to a file that uses a custom delimiter. The example uses arguments of the [PostgreSQL COPY](#) command to specify the comma-separated value (CSV) format and a colon (:) delimiter.

```
SELECT * from aws_s3.query_export_to_s3('select * from basic_test', :'s3_uri_1',
options :='format csv, delimiter $$:$>');
```

Exporting to a binary file with encoding

The following example shows how to call the [aws_s3.query_export_to_s3 \(p. 1458\)](#) function to export data to a binary file that has Windows-1253 encoding.

```
SELECT * from aws_s3.query_export_to_s3('select * from basic_test', :'s3_uri_1',
options :='format binary, encoding WIN1253');
```

Troubleshooting access to Amazon S3

If you encounter connection problems when attempting to export data to Amazon S3, first confirm that the outbound access rules for the VPC security group associated with your DB instance permit network connectivity. Specifically, they must allow access to port 443 for SSL connections. For more information, see [Provide access to the DB cluster in the VPC by creating a security group \(p. 87\)](#).

See also the following for recommendations:

- [Troubleshooting Amazon Aurora identity and access \(p. 1769\)](#)
- [Troubleshooting Amazon S3 in the *Amazon Simple Storage Service User Guide*](#)
- [Troubleshooting Amazon S3 and IAM in the *IAM User Guide*](#)

Function reference

Functions

- [aws_s3.query_export_to_s3 \(p. 1458\)](#)

- [aws_commons.create_s3_uri \(p. 1460\)](#)

aws_s3.query_export_to_s3

Exports a PostgreSQL query result to an Amazon S3 bucket. The aws_s3 extension provides the aws_s3.query_export_to_s3 function.

The two required parameters are `query` and `s3_info`. These define the query to be exported and identify the Amazon S3 bucket to export to. An optional parameter called `options` provides for defining various export parameters. For examples of using the aws_s3.query_export_to_s3 function, see [Exporting query data using the aws_s3.query_export_to_s3 function \(p. 1455\)](#).

Syntax

```
aws_s3.query_export_to_s3(  
    query text,  
    s3_info aws_commons._s3_uri_1,  
    options text  
)
```

Input parameters

query

A required text string containing an SQL query that the PostgreSQL engine runs. The results of this query are copied to an S3 bucket identified in the `s3_info` parameter.

s3_info

An `aws_commons._s3_uri_1` composite type containing the following information about the S3 object:

- `bucket` – The name of the Amazon S3 bucket to contain the file.
- `file_path` – The Amazon S3 file name and path.
- `region` – The AWS Region that the bucket is in. For a listing of AWS Region names and associated values, see [Regions and Availability Zones \(p. 11\)](#).

Currently, this value must be the same AWS Region as that of the exporting DB cluster . The default is the AWS Region of the exporting DB cluster .

To create an `aws_commons._s3_uri_1` composite structure, see the [aws_commons.create_s3_uri \(p. 1460\)](#) function.

options

An optional text string containing arguments for the PostgreSQL COPY command. These arguments specify how the data is to be copied when exported. For more details, see the [PostgreSQL COPY documentation](#).

Alternate input parameters

To help with testing, you can use an expanded set of parameters instead of the `s3_info` parameter. Following are additional syntax variations for the aws_s3.query_export_to_s3 function.

Instead of using the `s3_info` parameter to identify an Amazon S3 file, use the combination of the `bucket`, `file_path`, and `region` parameters.

```
aws_s3.query_export_to_s3(  
    query text,  
    bucket text,
```

```
    file_path text,  
    region text,  
    options text  
)
```

query

A required text string containing an SQL query that the PostgreSQL engine runs. The results of this query are copied to an S3 bucket identified in the *s3_info* parameter.

bucket

A required text string containing the name of the Amazon S3 bucket that contains the file.

file_path

A required text string containing the Amazon S3 file name including the path of the file.

region

An optional text string containing the AWS Region that the bucket is in. For a listing of AWS Region names and associated values, see [Regions and Availability Zones \(p. 11\)](#).

Currently, this value must be the same AWS Region as that of the exporting DB cluster . The default is the AWS Region of the exporting DB cluster .

options

An optional text string containing arguments for the PostgreSQL COPY command. These arguments specify how the data is to be copied when exported. For more details, see the [PostgreSQL COPY documentation](#).

Output parameters

```
aws_s3.query_export_to_s3(  
    OUT rows_uploaded bigint,  
    OUT files_uploaded bigint,  
    OUT bytes_uploaded bigint  
)
```

rows_uploaded

The number of table rows that were successfully uploaded to Amazon S3 for the given query.

files_uploaded

The number of files uploaded to Amazon S3. Files are created in sizes of approximately 6 GB. Each additional file created has *_partXX* appended to the name. The *XX* represents 2, then 3, and so on as needed.

bytes_uploaded

The total number of bytes uploaded to Amazon S3.

Examples

```
psql=> SELECT * from aws_s3.query_export_to_s3('select * from sample_table', 'sample-  
bucket', 'sample-filepath');  
psql=> SELECT * from aws_s3.query_export_to_s3('select * from sample_table', 'sample-  
bucket', 'sample-filepath','us-west-2');  
psql=> SELECT * from aws_s3.query_export_to_s3('select * from sample_table', 'sample-  
bucket', 'sample-filepath','us-west-2','format text');
```

aws_commons.create_s3_uri

Creates an `aws_commons._s3_uri_1` structure to hold Amazon S3 file information. You use the results of the `aws_commons.create_s3_uri` function in the `s3_info` parameter of the [aws_s3.query_export_to_s3 \(p. 1458\)](#) function. For an example of using the `aws_commons.create_s3_uri` function, see [Specifying the Amazon S3 file path to export to \(p. 1451\)](#).

Syntax

```
aws_commons.create_s3_uri(  
    bucket text,  
    file_path text,  
    region text  
)
```

Input parameters

bucket

A required text string containing the Amazon S3 bucket name for the file.

file_path

A required text string containing the Amazon S3 file name including the path of the file.

region

A required text string containing the AWS Region that the file is in. For a listing of AWS Region names and associated values, see [Regions and Availability Zones \(p. 11\)](#).

Managing query execution plans for Aurora PostgreSQL

With query plan management for Amazon Aurora PostgreSQL-Compatible Edition, you can control how and when query execution plans change. Query plan management has two main objectives:

- Preventing plan regressions when the database system changes
- Controlling when the query optimizer can use new plans

The quality and consistency of query optimization have a major impact on the performance and stability of any relational database management system (RDBMS). Query optimizers create a query execution plan for a SQL statement at a specific point in time. As conditions change, the optimizer might pick a different plan that makes performance better or worse. In some cases, a number of changes can all cause the query optimizer to choose a different plan and lead to performance regression. These changes include changes in statistics, constraints, environment settings, query parameter bindings, and software upgrades. Regression is a major concern for high-performance applications.

With query plan management, you can control execution plans for a set of statements that you want to manage. You can do the following:

- Improve plan stability by forcing the optimizer to choose from a small number of known, good plans.
- Optimize plans centrally and then distribute the best plans globally.
- Identify indexes that aren't used and assess the impact of creating or dropping an index.
- Automatically detect a new minimum-cost plan discovered by the optimizer.

- Try new optimizer features with less risk, because you can choose to approve only the plan changes that improve performance.

Topics

- [Enabling query plan management for Aurora PostgreSQL \(p. 1461\)](#)
- [Upgrading query plan management \(p. 1462\)](#)
- [Basics of query plan management \(p. 1462\)](#)
- [Best practices for query plan management \(p. 1465\)](#)
- [Examining plans in the `apg_plan_mgmt.dba_plans` view \(p. 1466\)](#)
- [Capturing execution plans \(p. 1469\)](#)
- [Using managed plans \(p. 1470\)](#)
- [Maintaining execution plans \(p. 1473\)](#)
- [Parameter reference for query plan management \(p. 1477\)](#)
- [Function reference for query plan management \(p. 1480\)](#)

Enabling query plan management for Aurora PostgreSQL

Query plan management is available with the following Aurora PostgreSQL versions:

- All Aurora PostgreSQL 13 versions
- Aurora PostgreSQL version 12.4 and higher
- Aurora PostgreSQL version 11.6 and higher
- Aurora PostgreSQL version 10.5 and higher

Only users with the `rds_superuser` role can complete the following procedure. The `rds_superuser` is required for creating the `apg_plan_mgmt` extension and its `apg_plan_mgmt` role. Users must be granted the `apg_plan_mgmt` role to administer the `apg_plan_mgmt` extension.

To enable query plan management

1. Open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. Create a new instance-level parameter group to use for query plan management parameters. For more information, see [Creating a DB parameter group \(p. 342\)](#). Associate the new parameter group with the DB instances in which you want to use query plan management. For more information, see [Modify a DB instance in a DB cluster \(p. 373\)](#).
3. Create a new cluster-level parameter group to use for query plan management parameters. For more information, see [Creating a DB cluster parameter group \(p. 343\)](#). Associate the new cluster-level parameter group with the DB clusters in which you want to use query plan management. For more information, see [Modifying the DB cluster by using the console, CLI, and API \(p. 372\)](#).
4. Open your cluster-level parameter group and set the `rds.enable_plan_management` parameter to 1. For more information, see [Modifying parameters in a DB cluster parameter group \(p. 349\)](#).
5. Reboot your DB instance to enable this new setting.
6. Connect to your DB instance with a SQL client such as `psql`.
7. Create the `apg_plan_mgmt` extension for your DB instance. The following shows an example.

```
psql my-database
my-database=> CREATE EXTENSION apg_plan_mgmt;
```

If you create the `apg_plan_mgmt` extension in the `template1` default database, then the query plan management extension is available in each new database that you create.

You can disable query plan management at any time by turning off the `apg_plan_mgmt.use_plan_baselines` and `apg_plan_mgmt.capture_plan_baselines`:

```
my-database=> SET apg_plan_mgmt.use_plan_baselines = off;
my-database=> SET apg_plan_mgmt.capture_plan_baselines = off;
```

Upgrading query plan management

The latest version of query plan management is 2.0. If you installed an earlier version of query plan management, we strongly recommend that you upgrade to version 2.0. For version details, see [Extension versions for Amazon Aurora PostgreSQL \(p. 1668\)](#).

To upgrade, run the following commands at the cluster or DB instance level.

```
ALTER EXTENSION apg_plan_mgmt UPDATE TO '2.0';
SELECT apg_plan_mgmt.validate_plans('update_plan_hash');
SELECT apg_plan_mgmt.reload();
```

Basics of query plan management

You can manage any SELECT, INSERT, UPDATE, or DELETE statement with query plan management, regardless of how complex the statement is. Prepared, dynamic, embedded, and immediate-mode SQL statements are all supported. All PostgreSQL language features can be used, including partitioned tables, inheritance, row-level security, and recursive common table expressions (CTEs).

Topics

- [Performing a manual plan capture \(p. 1462\)](#)
- [Viewing captured plans \(p. 1463\)](#)
- [Working with managed statements and the SQL hash \(p. 1463\)](#)
- [Working with automatic plan capture \(p. 1464\)](#)
- [Validating plans \(p. 1464\)](#)
- [Approving new plans that improve performance \(p. 1465\)](#)
- [Deleting plans \(p. 1465\)](#)

Performing a manual plan capture

To capture plans for specific statements, use the manual capture mode as in the following example.

```
/* Turn on manual capture */
SET apg_plan_mgmt.capture_plan_baselines = manual;
EXPLAIN SELECT COUNT(*) from pg_class;          -- capture the plan baseline
SET apg_plan_mgmt.capture_plan_baselines = off;    -- turn off capture
SET apg_plan_mgmt.use_plan_baselines = true;      -- turn on plan usage
```

You can either execute SELECT, INSERT, UPDATE, or DELETE statements, or you can include the EXPLAIN statement as shown above. Use EXPLAIN to capture a plan without the overhead or potential side-effects of executing the statement. For more about manual capture, see [Manually capturing plans for specific SQL statements \(p. 1469\)](#). Note that query plan management doesn't save the plans for statements that refer to system tables such as `pg_class`.

Viewing captured plans

When EXPLAIN SELECT runs in the previous example, the optimizer saves the plan. To do so, it inserts a row into the `apg_plan_mgmt.dba_plans` view and commits the plan in an autonomous transaction. You can see the contents of the `apg_plan_mgmt.dba_plans` view if you've been granted the `apg_plan_mgmt` role. The following query displays some important columns of the `dba_plans` view.

```
SELECT sql_hash, plan_hash, status, enabled, plan_outline, sql_text::varchar(40)
FROM apg_plan_mgmt.dba_plans
ORDER BY sql_text, plan_created;
```

Each row displayed represents a managed plan. The preceding example displays the following information.

- `sql_hash` – The ID of the managed statement that the plan is for.
- `plan_hash` – The ID of the managed plan.
- `status` – The status of the plan. The optimizer can run an approved plan.
- `enabled` – A value that indicates whether the plan is enabled for use or disabled and not for use.
- `plan_outline` – Details of the managed plan.

For more about the `apg_plan_mgmt.dba_plans` view, see [Examining plans in the `apg_plan_mgmt.dba_plans` view \(p. 1466\)](#).

Working with managed statements and the SQL hash

A *managed statement* is a SQL statement captured by the optimizer under query plan management. You specify which SQL statements to capture as managed statements using either manual or automatic capture:

- For manual capture, you provide the specific statements to the optimizer as shown in the previous example.
- For automatic capture, the optimizer captures plans for statements that run multiple times. Automatic capture is shown in a later example.

In the `apg_plan_mgmt.dba_plans` view, you can identify a managed statement with a SQL hash value. The SQL hash is calculated on a normalized representation of the SQL statement that removes some differences such as the literal values. Using normalization means that when multiple SQL statements differ only in their literal or parameter values, they are represented by the same SQL hash in the `apg_plan_mgmt.dba_plans` view. Therefore, there can be multiple plans for the same SQL hash where each plan is optimal under different conditions.

When the optimizer processes any SQL statement, it uses the following rules to create the normalized SQL statement:

- Removes any leading block comment
- Removes the EXPLAIN keyword and EXPLAIN options, if present
- Removes trailing spaces
- Removes all literals
- Preserves space and case for readability

For example, take the following statement.

```
/*Leading comment*/ EXPLAIN SELECT /* Query 1 */ * FROM t WHERE x > 7 AND y = 1;
```

The optimizer normalizes this statement as the following.

```
SELECT /* Query 1 */ * FROM t WHERE x > CONST AND y = CONST;
```

Working with automatic plan capture

Use automatic plan capture if you want to capture plans for all SQL statements in your application, or if you can't use manual capture. With automatic plan capture, the optimizer captures plans for statements that run at least two times. To use automatic plan capture, do the following.

1. Create a custom DB parameter group based on the default DB parameter group for the version of Aurora PostgreSQL that you're running.
2. Edit the custom DB parameter group, by changing the `apg_plan_mgmt.capture_plan_baselines` setting to `automatic`.
3. Save your customized DB parameter group.
4. Apply your custom DB parameter group to an Aurora DB instance that is already running as follows:
 - Choose your Aurora PostgreSQL DB instance from the list in the navigation pane, and then choose **Modify**.
 - In the **Additional configuration** section of the Modify DB instance page, for the **DB parameter group**, choose your custom DB parameter group.
 - Choose **Continue**. Confirm the Summary of modifications and choose **Apply immediately**.
 - Choose **Modify DB instance** to apply your custom DB parameter group.

You can also use your custom DB parameter group when you create a new Aurora PostgreSQL DB instance. For more information about parameter groups, see [Modifying parameters in a DB parameter group \(p. 347\)](#).

As your application runs, the optimizer captures plans for any statement that runs more than once. The optimizer always sets the status of a managed statement's first captured plan to `approved`. A managed statement's set of approved plans is known as its *plan baseline*.

As your application continues to run, the optimizer might find additional plans for the managed statements. The optimizer sets additional captured plans to a status of `unapproved`.

The set of all captured plans for a managed statement is known as the *plan history*. Later, you can decide if the `unapproved` plans perform well and change them to `Approved`, `Rejected`, or `Preferred` by using the `apg_plan_mgmt.evolve_plan_baselines` function or the `apg_plan_mgmt.set_plan_status` function.

To turn off automatic plan capture, set `apg_plan_mgmt.capture_plan_baselines` to `off` in the parameter group for the DB instance. Follow the same general process as outlined above, modifying your custom DB parameter group value for `apg_plan_mgmt.capture_plan_baselines` and then applying the custom DB parameter group to your Aurora DB instance.

For more about plan capture, see [Capturing execution plans \(p. 1469\)](#).

Validating plans

Managed plans can become invalid ("stale") when objects that they depend on are removed, such as an index. To find and delete all plans that are stale, use the `apg_plan_mgmt.validate_plans` function.

```
SELECT apg_plan_mgmt.validate_plans('delete');
```

For more information, see [Validating plans \(p. 1474\)](#).

Approving new plans that improve performance

While using your managed plans, you can verify whether newer, lower-cost plans discovered by the optimizer are faster than the minimum-cost plan already in the plan baseline. To do the performance comparison and optionally approve the faster plans, call the `apg_plan_mgmt.evolve_plan_baselines` function.

The following example automatically approves any unapproved plan that is enabled and faster by at least 10 percent than the minimum-cost plan in the plan baseline.

```
SELECT apg_plan_mgmt.evolve_plan_baselines(
    sql_hash,
    plan_hash,
    1.1,
    'approve'
)
FROM apg_plan_mgmt.dba_plans
WHERE status = 'Unapproved' AND enabled = true;
```

When the `apg_plan_mgmt.evolve_plan_baselines` function runs, it collects performance statistics and saves them in the `apg_plan_mgmt.dba_plans` view in the columns `planning_time_ms`, `execution_time_ms`, `cardinality_error`, `total_time_benefit_ms`, and `execution_time_benefit_ms`. The `apg_plan_mgmt.evolve_plan_baselines` function also updates the columns `last_verified` or `last_validated` timestamps, in which you can see the most recent time the performance statistics were collected.

```
SELECT sql_hash, plan_hash, status, last_verified, sql_text::varchar(40)
FROM apg_plan_mgmt.dba_plans
ORDER BY last_verified DESC; -- value updated by evolve_plan_baselines()
```

For more information about verifying plans, see [Evaluating plan performance \(p. 1473\)](#).

Deleting plans

The optimizer deletes plans automatically if they have not been executed or chosen as the minimum-cost plan for the plan retention period. By default, the plan retention period is 32 days. To change the plan retention period, set the `apg_plan_mgmt.plan_retention_period` parameter.

You can also review the contents of the `apg_plan_mgmt.dba_plans` view and delete any plans you don't want by using the `apg_plan_mgmt.delete_plan` function. For more information, see [Deleting plans \(p. 1476\)](#).

Best practices for query plan management

Consider using a plan management style that is either proactive or reactive. These plan management styles contrast in how and when new plans get approved for use.

Proactive plan management to help prevent performance regression

With proactive plan management, you manually approve new plans after you have verified that they are faster. Do this to prevent plan performance regressions. Follow these steps for proactive plan management:

1. In a development environment, identify the SQL statements that have the greatest impact on performance or system throughput. Then capture the plans for these statements as described

in [Manually capturing plans for specific SQL statements \(p. 1469\)](#) and [Automatically capturing plans \(p. 1470\)](#).

2. Export the captured plans from the development environment and import them into the production environment. For more information, see [Exporting and importing plans \(p. 1476\)](#).
3. In production, run your application and enforce the use of approved managed plans. For more information, see [Using managed plans \(p. 1470\)](#). While the application runs, also add new plans as the optimizer discovers them. For more information, see [Automatically capturing plans \(p. 1470\)](#).
4. Analyze the unapproved plans and approve those that perform well. For more information, see [Evaluating plan performance \(p. 1473\)](#).
5. While your application continues to run, the optimizer begins to use the new plans as appropriate.

Reactive plan management to detect and repair performance regression

With reactive plan management, you monitor your application as it runs to detect plans that cause performance regressions. When you detect regressions, you manually reject or fix the bad plans. Follow these steps for reactive plan management:

1. While your application runs, enforce the use of managed plans and automatically add newly discovered plans as unapproved. For more information, see [Using managed plans \(p. 1470\)](#) and [Automatically capturing plans \(p. 1470\)](#).
2. Monitor your running application for performance regressions.
3. When you discover a plan regression, set the plan's status to `rejected`. The next time the optimizer runs the SQL statement, it automatically ignores the rejected plan and uses a different approved plan instead. For more information, see [Rejecting or disabling slower plans \(p. 1474\)](#).

In some cases, you might prefer to fix a bad plan rather than reject, disable, or delete it. Use the `pg_hint_plan` extension to experiment with improving a plan. With `pg_hint_plan`, you use special comments to tell the optimizer to override how it normally creates a plan. For more information, see [Fixing plans using pg_hint_plan \(p. 1475\)](#).

Examining plans in the `apg_plan_mgmt.dba_plans` view

Query plan management provides a new SQL view for database administrators (DBAs) to use called `apg_plan_mgmt.dba_plans`. This one view contains the plan history for all of the databases in the DB instance.

This view contains the plan history for all of your managed statements. Each managed plan is identified by the combination of a SQL hash value and a plan hash value. With these identifiers, you can use tools such as Amazon RDS Performance Insights to track individual plan performance. For more information on Performance Insights, see [Using Amazon RDS performance insights](#).

Note

Access to the `apg_plan_mgmt.dba_plans` view is restricted to users that hold the `apg_plan_mgmt` role.

Listing managed plans

To list the managed plans, use a `SELECT` statement on the `apg_plan_mgmt.dba_plans` view. The following example displays some columns in the `dba_plans` view such as the `status`, which identifies the approved and unapproved plans.

```
SELECT sql_hash, plan_hash, status, enabled, stmt_name
FROM apg_plan_mgmt.dba_plans;

sql_hash | plan_hash | status | enabled | stmt_name
-----+-----+-----+-----+-----+
1984047223 | 512153379 | Approved | t | rangequery
1984047223 | 512284451 | Unapproved | t | rangequery
(2 rows)
```

Reference for the apg_plan_mgmt.dba_plans view

The columns of plan information in the `apg_plan_mgmt.dba_plans` view include the following.

dba_plans column	Description
<code>cardinality_error</code>	A measure of the error between the estimated cardinality versus the actual cardinality. <i>Cardinality</i> is the number of table rows that the plan is to process. If the cardinality error is large, then it increases the likelihood that the plan isn't optimal. This column is populated by the apg_plan_mgmt.evolve_plan_baselines (p. 1481) function.
<code>compatibility_level</code>	The feature level of the Aurora PostgreSQL optimizer.
<code>created_by</code>	The authenticated user (<code>session_user</code>) who created the plan.
<code>enabled</code>	An indicator of whether the plan is enabled or disabled. All plans are enabled by default. You can disable plans to prevent them from being used by the optimizer. To modify this value, use the apg_plan_mgmt.set_plan_enabled (p. 1484) function.
<code>environment_variables</code>	The PostgreSQL Grand Unified Configuration (GUC) parameters and values that the optimizer has overridden at the time the plan was captured.
<code>estimated_startup_cost</code>	The estimated optimizer setup cost before the optimizer delivers rows of a table.
<code>estimated_total_cost</code>	The estimated optimizer cost to deliver the final table row.
<code>execution_time_benefit</code>	The execution time benefit in milliseconds of enabling the plan. This column is populated by the apg_plan_mgmt.evolve_plan_baselines (p. 1481) function.
<code>execution_time_ms</code>	The estimated time in milliseconds that the plan would run. This column is populated by the apg_plan_mgmt.evolve_plan_baselines (p. 1481) function.
<code>has_side_effects</code>	A value that indicates that the SQL statement is a data manipulation language (DML) statement or a SELECT statement that contains a VOLATILE function.
<code>last_used</code>	This value is updated to the current date whenever the plan is either executed or when the plan is the query optimizer's minimum-cost plan. This value is stored in shared memory and periodically flushed to disk. To get the most up-to-date value, read the date from shared memory by calling the function <code>apg_plan_mgmt.plan_last_used(sql_hash, plan_hash)</code> instead of reading the <code>last_used</code> value. For additional information, see the apg_plan_mgmt.plan_retention_period (p. 1479) parameter.
<code>last_validated</code>	The most recent date and time when it was verified that the plan could be recreated by either the apg_plan_mgmt.validate_plans (p. 1486) function or the apg_plan_mgmt.evolve_plan_baselines (p. 1481) function.

dba_plans column	Description
last_verified	The most recent date and time when a plan was verified to be the best-performing plan for the specified parameters by the apg_plan_mgmt.evolve_plan_baselines (p. 1481) function.
origin	How the plan was captured with the apg_plan_mgmt.capture_plan_baselines (p. 1478) parameter. Valid values include the following: M – The plan was captured with manual plan capture. A – The plan was captured with automatic plan capture.
param_list	The parameter values that were passed to the statement if this is a prepared statement.
plan_created	The date and time the plan was created.
plan_hash	The plan identifier. The combination of <code>plan_hash</code> and <code>sql_hash</code> uniquely identifies a specific plan.
plan_outline	A representation of the plan that is used to recreate the actual execution plan, and that is database-independent. Operators in the tree correspond to operators that appear in the EXPLAIN output.
planning_time_ms	The actual time to run the planner, in milliseconds. This column is populated by the apg_plan_mgmt.evolve_plan_baselines (p. 1481) function.
queryId	A statement hash, as calculated by the <code>pg_stat_statements</code> extension. This isn't a stable or database-independent identifier because it depends on object identifiers (OIDs).
sql_hash	A hash value of the SQL statement text, normalized with literals removed.
sql_text	The full text of the SQL statement.
status	A plan's status, which determines how the optimizer uses a plan. Valid values include the following. <ul style="list-style-type: none"> Approved – A usable plan that the optimizer can choose to run. The optimizer runs the least-cost plan from a managed statement's set of approved plans (baseline). To reset a plan to approved, use the apg_plan_mgmt.evolve_plan_baselines (p. 1481) function. Unapproved – A captured plan that you have not verified for use. For more information, see Evaluating plan performance (p. 1473). Rejected – A plan that the optimizer won't use. For more information, see Rejecting or disabling slower plans (p. 1474). Preferred – A plan that you have determined is a preferred plan to use for a managed statement. If the optimizer's minimum-cost plan isn't an approved or preferred plan, you can reduce plan enforcement overhead. To do so, make a subset of the approved plans Preferred. When the optimizer's minimum cost isn't an Approved plan, a Preferred plan is chosen before an Approved plan. To reset a plan to Preferred, use the apg_plan_mgmt.set_plan_status (p. 1484) function.

dba_plans column	Description
stmt_name	The name of the SQL statement within a PREPARE statement. This value is an empty string for an unnamed prepared statement. This value is NULL for a nonprepared statement.
total_time_benefit	The total time benefit in milliseconds of enabling this plan. This value considers both planning time and execution time. If this value is negative, there is a disadvantage to enabling this plan. This column is populated by the apg_plan_mgmt.evolve_plan_baseline (p. 1481) function.

Capturing execution plans

You can capture execution plans for specific SQL statements by using manual plan capture. Alternatively, you can capture all (or the slowest) plans that are executed two or more times as your application runs by using automatic plan capture.

When capturing plans, the optimizer sets the status of a managed statement's first captured plan to approved. The optimizer sets the status of any additional plans captured for a managed statement to unapproved. However, more than one plan might occasionally be saved with the approved status. This can happen when multiple plans are created for a statement in parallel and before the first plan for the statement is committed.

To control the maximum number of plans that can be captured and stored in the dba_plans view, set the `apg_plan_mgmt.max_plans` parameter in your DB instance-level parameter group. A change to the `apg_plan_mgmt.max_plans` parameter requires a DB instance reboot for a new value to take effect. For more information, see the [apg_plan_mgmt.max_plans \(p. 1478\)](#) parameter.

Topics

- [Manually capturing plans for specific SQL statements \(p. 1469\)](#)
- [Automatically capturing plans \(p. 1470\)](#)

Manually capturing plans for specific SQL statements

If you have a known set of SQL statements to manage, put the statements into a SQL script file and then manually capture plans. The following shows a psql example of how to capture query plans manually for a set of SQL statements.

```
psql> SET apg_plan_mgmt.capture_plan_baseline = manual;
psql> \i my-statements.sql
psql> SET apg_plan_mgmt.capture_plan_baseline = off;
```

After capturing a plan for each SQL statement, the optimizer adds a new row to the `apg_plan_mgmt.dba_plans` view.

We recommend that you use either EXPLAIN or EXPLAIN EXECUTE statements in the SQL script file. Make sure that you include enough variations in parameter values to capture all the plans of interest.

If you know of a better plan than the optimizer's minimum cost plan, you might be able to force the optimizer to use the better plan. To do so, specify one or more optimizer hints. For more information, see [Fixing plans using pg_hint_plan \(p. 1475\)](#). To compare the performance of the unapproved and approved plans and approve, reject, or delete them, see [Evaluating plan performance \(p. 1473\)](#).

Automatically capturing plans

Use automatic plan capture for situations such as the following:

- You don't know the specific SQL statements that you want to manage.
- You have hundreds or thousands of SQL statements to manage.
- Your application uses a client API. For example, JDBC uses unnamed prepared statements or bulk-mode statements that can't be expressed in psql.

To capture plans automatically

1. Turn on automatic plan capture by setting `apg_plan_mgmt.capture_plan_baselines` to `automatic` in the DB instance-level parameter group. For more information, see [Modifying parameters in a DB parameter group \(p. 347\)](#).
2. Reboot your DB instance.
3. As the application runs, the optimizer captures plans for each SQL statement that runs at least twice.

As the application runs with default query plan management parameter settings, the optimizer captures plans for each SQL statement that runs at least twice. Capturing all plans while using the defaults has very little run-time overhead and can be enabled in production.

To turn off automatic plan capture

- Set the `apg_plan_mgmt.capture_plan_baselines` parameter to `off` from the DB instance-level parameter group.

To measure the performance of the unapproved plans and approve, reject, or delete them, see [Evaluating plan performance \(p. 1473\)](#).

Using managed plans

To get the optimizer to use captured plans for your managed statements, set the parameter `apg_plan_mgmt.use_plan_baselines` to `true`. The following is a local instance example.

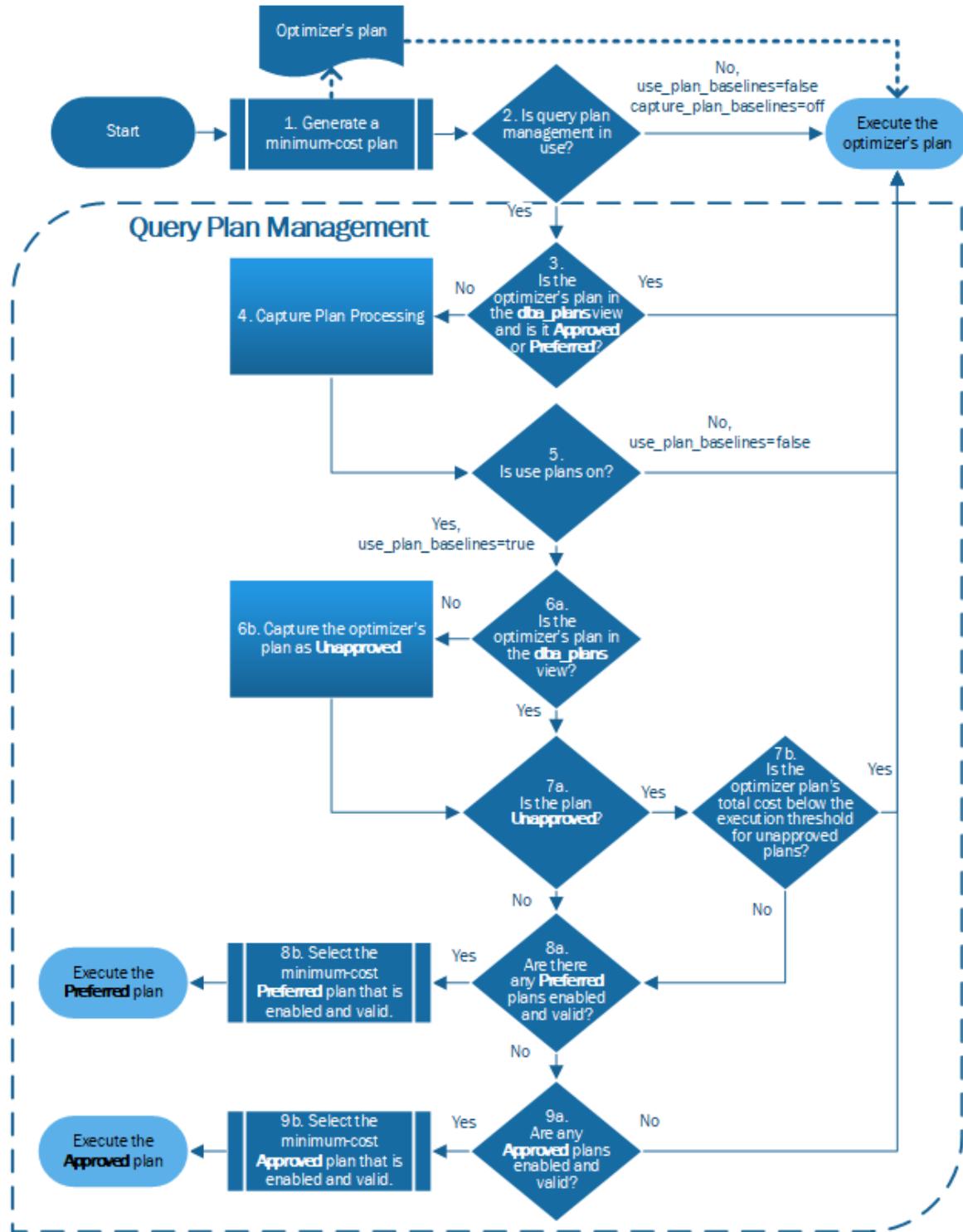
```
SET apg_plan_mgmt.use_plan_baselines = true;
```

While the application runs, this setting causes the optimizer to use the minimum-cost, preferred, or approved plan that is valid and enabled, for each managed statement.

How the optimizer chooses which plan to run

The cost of an execution plan is an estimate that the optimizer makes to compare different plans. Optimizer cost is a function of several factors that include the CPU and I/O operations that the plan uses. For more information about PostgreSQL query planner costs, see the [PostgreSQL documentation on query planning](#).

The following flowchart shows how the query plan management optimizer chooses which plan to run.



The flow is as follows:

1. When the optimizer processes every SQL statement, it generates a minimum-cost plan.
2. Without query plan management, the optimizer simply runs its generated plan. The optimizer uses query plan management if you set one or both of the following parameter settings:

- `apg_plan_mgmt.capture_plan_baselines` to manual or automatic
 - `apg_plan_mgmt.use_plan_baselines` to true
3. The optimizer immediately runs the generated plan if the following are both true:
 - The optimizer's plan is already in the `apg_plan_mgmt.dba_plans` view for the SQL statement.
 - The plan's status is either approved or preferred.
 4. The optimizer goes through the capture plan processing if the parameter `apg_plan_mgmt.capture_plan_baselines` is manual or automatic.

For details on how the optimizer captures plans, see [Capturing execution plans \(p. 1469\)](#).
 5. The optimizer runs the generated plan if `apg_plan_mgmt.use_plan_baselines` is false.
 6. If the optimizer's plan isn't in the `apg_plan_mgmt.dba_plans` view, the optimizer captures the plan as a new unapproved plan.
 7. The optimizer runs the generated plan if the following are both true:
 - The optimizer's plan isn't a rejected or disabled plan.
 - The plan's total cost is less than the unapproved execution plan threshold.

The optimizer doesn't run disabled plans or any plans that have the rejected status. In most cases, the optimizer doesn't execute unapproved plans. However, the optimizer runs an unapproved plan if you set a value for the parameter `apg_plan_mgmt.unapproved_plan_execution_threshold` and the plan's total cost is less than the threshold. For more information, see the [apg_plan_mgmt.unapproved_plan_execution_threshold \(p. 1479\)](#) parameter.
 8. If the managed statement has any enabled and valid preferred plans, the optimizer runs the minimum-cost one.

A valid plan is one that the optimizer can run. Managed plans can become invalid for various reasons. For example, plans become invalid when objects that they depend on are removed, such as an index or a partition of a partitioned table.
 9. The optimizer determines the minimum-cost plan from the managed statement's approved plans that are both enabled and valid. The optimizer then runs the minimum-cost approved plan.

Analyzing which plan the optimizer will use

When the `apg_plan_mgmt.use_plan_baselines` parameter is set to `true`, you can use EXPLAIN ANALYZE SQL statements to cause the optimizer to show the plan it would use if it were to run the statement. The following is an example.

```
EXPLAIN ANALYZE EXECUTE rangeQuery (1,10000);
```

```
QUERY PLAN
-----
Aggregate (cost=393.29..393.30 rows=1 width=8) (actual time=7.251..7.251 rows=1 loops=1)
  -> Index Only Scan using t1_pkey on t1 t  (cost=0.29..368.29 rows=10000 width=0)
      (actual time=0.061..4.859 rows=10000 loops=1)
Index Cond: ((id >= 1) AND (id <= 10000))
      Heap Fetches: 10000
Planning time: 1.408 ms
Execution time: 7.291 ms
Note: An Approved plan was used instead of the minimum cost plan.
SQL Hash: 1984047223, Plan Hash: 512153379
```

The optimizer indicates which plan it will run, but notice that in this example that it found a lower-cost plan. In this case, you capture this new minimum cost plan by turning on automatic plan capture as described in [Automatically capturing plans \(p. 1470\)](#).

The optimizer captures new plans as **Unapproved**. Use the `apg_plan_mgmt.evolve_plan_baselines` function to compare plans and change them to **approved**, **rejected**, or **disabled**. For more information, see [Evaluating plan performance \(p. 1473\)](#).

Maintaining execution plans

Query plan management provides techniques and functions to add, maintain, and improve execution plans.

Topics

- [Evaluating plan performance \(p. 1473\)](#)
- [Validating plans \(p. 1474\)](#)
- [Fixing plans using pg_hint_plan \(p. 1475\)](#)
- [Deleting plans \(p. 1476\)](#)
- [Exporting and importing plans \(p. 1476\)](#)

Evaluating plan performance

After the optimizer captures plans as unapproved, use the `apg_plan_mgmt.evolve_plan_baselines` function to compare plans based on their actual performance. Depending on the outcome of your performance experiments, you can change a plan's status from unapproved to either approved or rejected. You can instead decide to use the `apg_plan_mgmt.evolve_plan_baselines` function to temporarily disable a plan if it does not meet your requirements.

Topics

- [Approving better plans \(p. 1473\)](#)
- [Rejecting or disabling slower plans \(p. 1474\)](#)

Approving better plans

The following example demonstrates how to change the status of managed plans to approved using the `apg_plan_mgmt.evolve_plan_baselines` function.

```
SELECT apg_plan_mgmt.evolve_plan_baselines (
    sql_hash,
    plan_hash,
    min_speedup_factor := 1.0,
    action := 'approve'
)
FROM apg_plan_mgmt.dba_plans WHERE status = 'Unapproved';
```

```
NOTICE:      rangequery (1,10000)
NOTICE:      Baseline [ Planning time 0.761 ms, Execution time 13.261 ms]
NOTICE:      Baseline+1 [ Planning time 0.204 ms, Execution time 8.956 ms]
NOTICE:      Total time benefit: 4.862 ms, Execution time benefit: 4.305 ms
NOTICE:      Unapproved -> Approved
evolve_plan_baselines
-----
0
(1 row)
```

The output shows a performance report for the `rangequery` statement with parameter bindings of 1 and 10,000. The new unapproved plan (`Baseline+1`) is better than the best previously approved plan

(Baseline). To confirm that the new plan is now Approved, check the `apg_plan_mgmt.dba_plans` view.

```
SELECT sql_hash, plan_hash, status, enabled, stmt_name
FROM apg_plan_mgmt.dba_plans;
```

sql_hash	plan_hash	status	enabled	stmt_name
1984047223	512153379	Approved	t	rangequery
1984047223	512284451	Approved	t	rangequery

(2 rows)

The managed plan now includes two approved plans that are the statement's plan baseline. You can also call the `apg_plan_mgmt.set_plan_status` function to directly set a plan's status field to 'Approved', 'Rejected', 'Unapproved', or 'Preferred'.

Rejecting or disabling slower plans

To reject or disable plans, pass 'reject' or 'disable' as the action parameter to the `apg_plan_mgmt.evolve_plan_baselines` function. This example disables any captured Unapproved plan that is slower by at least 10 percent than the best Approved plan for the statement.

```
SELECT apg_plan_mgmt.evolve_plan_baselines(
    sql_hash, -- The managed statement ID
    plan_hash, -- The plan ID
    1.1, -- number of times faster the plan must be
    'disable' -- The action to take. This sets the enabled field to false.
)
FROM apg_plan_mgmt.dba_plans
WHERE status = 'Unapproved' AND -- plan is Unapproved
origin = 'Automatic'; -- plan was auto-captured
```

You can also directly set a plan to rejected or disabled. To directly set a plan's enabled field to true or false, call the `apg_plan_mgmt.set_plan_enabled` function. To directly set a plan's status field to 'Approved', 'Rejected', 'Unapproved', or 'Preferred', call the `apg_plan_mgmt.set_plan_status` function.

Validating plans

Use the `apg_plan_mgmt.validate_plans` function to delete or disable plans that are invalid.

Plans can become invalid or stale when objects that they depend on are removed, such as an index or a table. However, a plan might be invalid only temporarily if the removed object gets recreated. If an invalid plan can become valid later, you might prefer to disable an invalid plan or do nothing rather than delete it.

To find and delete all plans that are invalid and haven't been used in the past week, use the `apg_plan_mgmt.validate_plans` function as follows.

```
SELECT apg_plan_mgmt.validate_plans(sql_hash, plan_hash, 'delete')
FROM apg_plan_mgmt.dba_plans
WHERE last_used < (current_date - interval '7 days');
```

To enable or disabled a plan directly, use the `apg_plan_mgmt.set_plan_enabled` function.

Fixing plans using pg_hint_plan

The query optimizer is well-designed to find an optimal plan for all statements, and in most cases the optimizer finds a good plan. However, occasionally you might know that a much better plan exists than that generated by the optimizer. Two recommended ways to get the optimizer to generate a desired plan include using the `pg_hint_plan` extension or setting Grand Unified Configuration (GUC) variables in PostgreSQL:

- `pg_hint_plan` extension – Specify a "hint" to modify how the planner works by using PostgreSQL's `pg_hint_plan` extension. To install and learn more about how to use the `pg_hint_plan` extension, see the [pg_hint_plan documentation](#).
- GUC variables – Override one or more cost model parameters or other optimizer parameters, such as the `fromCollapse_limit` or `GEOO_threshold`.

When you use one of these techniques to force the query optimizer to use a plan, you can also use query plan management to capture and enforce use of the new plan.

You can use the `pg_hint_plan` extension to change the join order, the join methods, or the access paths for a SQL statement. You use a SQL comment with special `pg_hint_plan` syntax to modify how the optimizer creates a plan. For example, assume the problem SQL statement has a two-way join.

```
SELECT *
FROM t1, t2
WHERE t1.id = t2.id;
```

Then suppose that the optimizer chooses the join order (`t1, t2`), but we know that the join order (`t2, t1`) is faster. The following hint forces the optimizer to use the faster join order, (`t2, t1`). Include EXPLAIN so that the optimizer generates a plan for the SQL statement but does not run the statement. (Output not shown.)

```
/*+ Leading ((t2 t1)) */ EXPLAIN SELECT *
FROM t1, t2
WHERE t1.id = t2.id;
```

The following steps show how to use `pg_hint_plan`.

To modify the optimizer's generated plan and capture the plan using pg_hint_plan

1. Turn on the manual capture mode.

```
SET apg_plan_mgmt.capture_plan_baselines = manual;
```

2. Specify a hint for the SQL statement of interest.

```
/*+ Leading ((t2 t1)) */ EXPLAIN SELECT *
FROM t1, t2
WHERE t1.id = t2.id;
```

After this runs, the optimizer captures the plan in the `apg_plan_mgmt.dba_plans` view. The captured plan doesn't include the special `pg_hint_plan` comment syntax because query plan management normalizes the statement by removing leading comments.

3. View the managed plans by using the `apg_plan_mgmt.dba_plans` view.

```
SELECT sql_hash, plan_hash, status, sql_text, plan_outline
FROM apg_plan_mgmt.dba_plans;
```

4. Set the status of the plan to `Preferred`. Doing so makes sure that the optimizer chooses to run it, instead of selecting from the set of approved plans, when the minimum-cost plan isn't already `Approved` or `Preferred`.

```
SELECT apg_plan_mgmt.set_plan_status(sql_hash, plan_hash, 'preferred' );
```

5. Turn off manual plan capture and enforce the use of managed plans.

```
SET apg_plan_mgmt.capture_plan_baselines = false;
SET apg_plan_mgmt.use_plan_baselines = true;
```

Now, when the original SQL statement runs, the optimizer chooses either an `Approved` or `Preferred` plan. If the minimum-cost plan isn't `Approved` or `Preferred`, then the optimizer chooses the `Preferred` plan.

Deleting plans

Delete plans that have not been used for a long time or that are no longer relevant. Each plan has a `last_used` date that the optimizer updates each time it executes a plan or picks it as the minimum-cost plan for a statement. Use the `last_used` date to determine if a plan has been used recently and is still relevant.

For example, you can use the `apg_plan_mgmt.delete_plan` function as follows. Doing this deletes all plans that haven't been chosen as the minimum-cost plan or haven't run in at least 31 days. However, this example doesn't delete plans that have been explicitly rejected.

```
SELECT SUM(apg_plan_mgmt.delete_plan(sql_hash, plan_hash))
FROM apg_plan_mgmt.dba_plans
WHERE last_used < (current_date - interval '31 days')
AND status <> 'Rejected';
```

To delete any plan that is no longer valid and that you expect not to become valid again, use the `apg_plan_mgmt.validate_plans` function. For more information, see [Validating plans \(p. 1474\)](#).

You can implement your own policy for deleting plans. Plans are automatically deleted when the current date `last_used` is greater than the value of the `apg_plan_mgmt.plan_retention_period` parameter, which defaults to 32 days. You can specify a longer interval, or you can implement your own plan retention policy by calling the `delete_plan` function directly. The `last_used` date is the most recent date that either the optimizer chose a plan as the minimum cost plan or that the plan was executed.

Important

If you don't clean up plans, you might eventually run out of shared memory that is set aside for query plan management. To control how much memory is available for managed plans, use the `apg_plan_mgmt.max_plans` parameter. Set this parameter in your DB instance-level parameter group and reboot your DB instance for changes to take effect. For more information, see the [apg_plan_mgmt.max_plans \(p. 1478\)](#) parameter.

Exporting and importing plans

You can export your managed plans and import them into another DB instance.

To export managed plans

An authorized user can copy any subset of the `apg_plan_mgmt.plans` table to another table, and then save it using the `pg_dump` command. The following is an example.

```
CREATE TABLE plans_copy AS SELECT *  
FROM apg_plan_mgmt.plans [ WHERE predicates ] ;
```

```
% pg_dump --table apg_plan_mgmt.plans_copy -Ft mysourcedatabase > plans_copy.tar
```

```
DROP TABLE apg_plan_mgmt.plans_copy;
```

To import managed plans

1. Copy the .tar file of the exported managed plans to the system where the plans are to be restored.
2. Use the `pg_restore` command to copy the tar file into a new table.

```
% pg_restore --dbname mytargetdatabase -Ft plans_copy.tar
```

3. Merge the `plans_copy` table with the `apg_plan_mgmt.plans` table, as shown in the following example.

Note

In some cases, you might dump from one version of the `apg_plan_mgmt` extension and restore into a different version. In these cases, the columns in the `plans` table might be different. If so, name the columns explicitly instead of using `SELECT *`.

```
INSERT INTO apg_plan_mgmt.plans SELECT * FROM plans_copy  
ON CONFLICT ON CONSTRAINT plans_pkey  
DO UPDATE SET  
status = EXCLUDED.status,  
enabled = EXCLUDED.enabled,  
-- Save the most recent last_used date  
--  
last_used = CASE WHEN EXCLUDED.last_used > plans.last_used  
THEN EXCLUDED.last_used ELSE plans.last_used END,  
-- Save statistics gathered by evolve_plan_baselines, if it ran:  
--  
estimated_startup_cost = EXCLUDED.estimated_startup_cost,  
estimated_total_cost = EXCLUDED.estimated_total_cost,  
planning_time_ms = EXCLUDED.planning_time_ms,  
execution_time_ms = EXCLUDED.execution_time_ms,  
total_time_benefit_ms = EXCLUDED.total_time_benefit_ms,  
execution_time_benefit_ms = EXCLUDED.execution_time_benefit_ms;
```

4. Reload the managed plans into shared memory and remove the temporary plans table.

```
SELECT apg_plan_mgmt.reload(); -- refresh shared memory  
DROP TABLE plans_copy;
```

Parameter reference for query plan management

The `apg_plan_mgmt` extension provides the following parameters.

Parameters

- [apg_plan_mgmt.capture_plan_baselines \(p. 1478\)](#)
- [apg_plan_mgmt.max_databases \(p. 1478\)](#)
- [apg_plan_mgmt.max_plans \(p. 1478\)](#)
- [apg_plan_mgmt.plan_retention_period \(p. 1479\)](#)

- [apg_plan_mgmt.unapproved_plan_execution_threshold \(p. 1479\)](#)
- [apg_plan_mgmt.use_plan_baselines \(p. 1479\)](#)

Set the query plan management parameters at the appropriate level:

- Set at the cluster-level to provide the same settings for all DB instances. For more information, see [Modifying parameters in a DB cluster parameter group \(p. 349\)](#).
- Set at the DB instance level to isolate the settings to an individual DB instance. For more information, see [Modifying parameters in a DB parameter group \(p. 347\)](#).
- Set in a specific client session such as in psql, to isolate the values to only that session.

You can set the `apg_plan_mgmt.max_databases` parameter and the `apg_plan_mgmt.max_plans` parameter at the Aurora DB cluster level or at the DB instance level.

apg_plan_mgmt.capture_plan_baselines

Enable execution plan capture for SQL statements.

```
SET apg_plan_mgmt.capture_plan_baselines = [off | automatic |manual];
```

Value	Description
off	Disable plan capture. This is the default.
automatic	Enable plan capture for subsequent SQL statements that satisfy the eligibility criteria.
manual	Enable plan capture for subsequent SQL statements.

apg_plan_mgmt.max_databases

Sets the maximum number of databases that can use query plan management. You can use the `psql` meta-command (`\l`) to find out how many databases are on the DB instance in the Aurora PostgreSQL DB cluster. By default, query plan management can support 10 databases. You can change the value of this parameter at the DB cluster level or at the DB instance level.

Important

If you change the value of `apg_plan_mgmt.max_databases`, be sure to reboot the DB instance so that the new value takes effect.

```
SET apg_plan_mgmt.max_databases = integer-value;
```

Value	Default	Description
Positive integer	10	A positive integer value.

apg_plan_mgmt.max_plans

Sets the maximum number of SQL statements that the query plan manager can maintain in the `apg_plan_mgmt.dba_plans` view. We recommend setting this parameter to 10000 or higher for all Aurora PostgreSQL versions.

Important

You can set the `apg_plan_mgmt.max_plans` parameter at the cluster level or at the DB instance level. Be sure to reboot the DB instance so that the new value can take effect.

```
SET apg_plan_mgmt.max_plans = integer-value;
```

Value	Default	Version
Positive integer >= 10	10000	Aurora PostgreSQL version 11 and higher versions
	1000	Aurora PostgreSQL version 10 and previous versions

apg_plan_mgmt.plan_retention_period

The number of days that plans are kept in the `apg_plan_mgmt.dba_plans` view before being automatically deleted. A plan is deleted when the current date is the specified number of days since the plan's `last_used` date. The default is 32 days. The `last_used` date is the most recent date that either the optimizer chose a plan as the minimum cost plan or that the plan was executed.

```
SET apg_plan_mgmt.plan_retention_period = integer-value;
```

Value	Default	Description
Positive integer	32	A positive integer value greater or equal to 32, representing days.

apg_plan_mgmt.unapproved_plan_execution_threshold

An estimated total plan cost threshold, below which the optimizer runs an unapproved plan. By default, the optimizer does not run unapproved plans. However, you can set an execution threshold for your fastest unapproved plans. With this setting, the optimizer bypasses the overhead of enforcing only approved plans.

```
SET apg_plan_mgmt.unapproved_plan_execution_threshold = integer-value;
```

Value	Default	Description
Positive integer	0	A positive integer value greater or equal to 0. A value of 0 means no unapproved plans run when <code>use_plan_baselines</code> is <code>true</code> .

With the following example, the optimizer runs an unapproved plan if the estimated cost is less than 550, even if `use_plan_baselines` is `true`.

```
SET apg_plan_mgmt.unapproved_plan_execution_threshold = 550;
```

apg_plan_mgmt.use_plan_baselines

Enforce the optimizer to use managed plans for managed statements.

```
SET apg_plan_mgmt.use_plan_baselines = [true | false];
```

Value	Description
<code>true</code>	Enforce the use of managed plans. When a SQL statement runs and it is a managed statement in the <code>apg_plan_mgmt.dba_plans</code> view, the optimizer chooses a managed plan in the following order. <ol style="list-style-type: none">1. The minimum-cost preferred plan that is valid and enabled.2. The minimum cost approved plan that is valid and enabled.3. The minimum cost unapproved plan that is valid, enabled, and that meets the threshold, if set with the <code>apg_plan_mgmt.unapproved_plan_execution_threshold</code> parameter.4. The optimizer's generated minimum-cost plan.
<code>false</code>	(Default) Do not use managed plans. The optimizer uses its generated minimum-cost plan.

Usage notes

When `use_plan_baselines` is `true`, then the optimizer makes the following execution decisions:

1. If the estimated cost of the optimizer's plan is below the `unapproved_plan_execution_threshold`, then execute it, else
2. If the plan is `approved` or `preferred`, then execute it, else
3. Execute a `minimum-cost preferred` plan, if possible, else
4. Execute a `minimum-cost approved` plan, if possible, else
5. Execute the optimizer's minimum-cost plan.

Function reference for query plan management

The `apg_plan_mgmt` extension provides the following functions.

Functions

- [apg_plan_mgmt.delete_plan \(p. 1480\)](#)
- [apg_plan_mgmt.evolve_plan_baselines \(p. 1481\)](#)
- [apg_plan_mgmt.get_explain_plan \(p. 1482\)](#)
- [apg_plan_mgmt.plan_last_used \(p. 1483\)](#)
- [apg_plan_mgmt.reload \(p. 1483\)](#)
- [apg_plan_mgmt.set_plan_enabled \(p. 1484\)](#)
- [apg_plan_mgmt.set_plan_status \(p. 1484\)](#)
- [apg_plan_mgmt.update_plans_last_used \(p. 1485\)](#)
- [apg_plan_mgmt.validate_plans \(p. 1486\)](#)

apg_plan_mgmt.delete_plan

Delete a managed plan.

Syntax

```
apg_plan_mgmt.delete_plan(
    sql_hash,
    plan_hash
)
```

Return value

Returns 0 if the delete was successful or -1 if the delete failed.

Parameters

Parameter	Description
sql_hash	The <code>sql_hash</code> ID of the plan's managed SQL statement.
plan_hash	The managed plan's <code>plan_hash</code> ID.

[apg_plan_mgmt.evolve_plan_baselines](#)

Verifies whether an already approved plan is faster or whether a plan identified by the query optimizer as a minimum cost plan is faster.

Syntax

```
apg_plan_mgmt.evolve_plan_baselines(
    sql_hash,
    plan_hash,
    min_speedup_factor,
    action
)
```

Return value

The number of plans that were not faster than the best approved plan.

Parameters

Parameter	Description
sql_hash	The <code>sql_hash</code> ID of the plan's managed SQL statement.
plan_hash	The managed plan's <code>plan_hash</code> ID. Use NULL to mean all plans that have the same <code>sql_hash</code> ID value.
min_speedup_factor	The <i>minimum speedup factor</i> can be the number of times faster that a plan must be than the best of the already approved plans to approve it. Alternatively, this factor can be the number of times slower that a plan must be to reject or disable it. This is a positive float value.

Parameter	Description
action	<p>The action the function is to perform. Valid values include the following. Case does not matter.</p> <ul style="list-style-type: none"> • 'disable' – Disable each matching plan that does not meet the minimum speedup factor. • 'approve' – Enable each matching plan that meets the minimum speedup factor and set its status to approved. • 'reject' – For each matching plan that does not meet the minimum speedup factor, set its status to rejected. • NULL – The function simply returns the number of plans that have no performance benefit because they do not meet the minimum speedup factor.

Usage notes

Set specified plans to approved, rejected, or disabled based on whether the planning plus execution time is faster than the best approved plan by a factor that you can set. The action parameter might be set to 'approve' or 'reject' to automatically approve or reject a plan that meets the performance criteria. Alternatively, it might be set to "" (empty string) to do the performance experiment and produce a report, but take no action.

You can avoid pointlessly rerunning of the `apg_plan_mgmt.evolve_plan_baselines` function for a plan on which it was recently run. To do so, restrict the plans to just the recently created unapproved plans. Alternatively, you can avoid running the `apg_plan_mgmt.evolve_plan_baselines` function on any approved plan that has a recent `last_verified` timestamp.

Conduct a performance experiment to compare the planning plus execution time of each plan relative to the other plans in the baseline. In some cases, there is only one plan for a statement and the plan is approved. In such a case, compare the planning plus execution time of the plan to the planning plus execution time of using no plan.

The incremental benefit (or disadvantage) of each plan is recorded in the `apg_plan_mgmt.dba_plans` view in the `total_time_benefit_ms` column. When this value is positive, there is a measurable performance advantage to including this plan in the baseline.

In addition to collecting the planning and execution time of each candidate plan, the `last_verified` column of the `apg_plan_mgmt.dba_plans` view is updated with the `current_timestamp`. The `last_verified` timestamp might be used to avoid running this function again on a plan that recently had its performance verified.

apg_plan_mgmt.get_explain_plan

Generates the text of an EXPLAIN statement for the specified SQL statement.

Syntax

```
apg_plan_mgmt.get_explain_plan(
    sql_hash,
    plan_hash,
    [explainOptionList]
)
```

Return value

Returns runtime statistics for the specified SQL statements. Use without `explainOptionList` to return a simple EXPLAIN plan.

Parameters

Parameter	Description
<code>sql_hash</code>	The <code>sql_hash</code> ID of the plan's managed SQL statement.
<code>plan_hash</code>	The managed plan's <code>plan_hash</code> ID.
<code>explainOptionList</code>	A comma-separated list of explain options. Valid values include ' <code>analyze</code> ', ' <code>verbose</code> ', ' <code>buffers</code> ', ' <code>hashes</code> ', and ' <code>format json</code> '. If the <code>explainOptionList</code> is <code>NULL</code> or an empty string (''), this function generates an <code>EXPLAIN</code> statement, without any statistics.

Usage notes

For the `explainOptionList`, you can use any of the same options that you would use with an `EXPLAIN` statement. The Aurora PostgreSQL optimizer concatenates the list of options you provide to the `EXPLAIN` statement, so you can request any option that `EXPLAIN` supports.

apg_plan_mgmt.plan_last_used

Returns the `last_used` date of the specified plan from shared memory.

Note

The value in shared memory is always current on the primary DB instance in the DB cluster. The value is only periodically flushed to the `last_used` column of the `apg_plan_mgmt.dba_plans` view.

Syntax

```
apg_plan_mgmt.plan_last_used(
    sql_hash,
    plan_hash
)
```

Return value

Returns the `last_used` date.

Parameters

Parameter	Description
<code>sql_hash</code>	The <code>sql_hash</code> ID of the plan's managed SQL statement.
<code>plan_hash</code>	The managed plan's <code>plan_hash</code> ID.

apg_plan_mgmt.reload

Reload plans into shared memory from the `apg_plan_mgmt.dba_plans` view.

Syntax

```
apg_plan_mgmt.reload()
```

Return value

None.

Parameters

None.

Usage notes

Call `reload` for the following situations:

- Use it to refresh the shared memory of a read-only replica immediately, rather than wait for new plans to propagate to the replica.
- Use it after importing managed plans.

apg_plan_mgmt.set_plan_enabled

Enable or disable a managed plan.

Syntax

```
apg_plan_mgmt.set_plan_enabled(  
    sql_hash,  
    plan_hash,  
    [true | false]  
)
```

Return value

Returns 0 if the setting was successful or -1 if the setting failed.

Parameters

Parameter	Description
<code>sql_hash</code>	The <code>sql_hash</code> ID of the plan's managed SQL statement.
<code>plan_hash</code>	The managed plan's <code>plan_hash</code> ID.
<code>enabled</code>	Boolean value of true or false: <ul style="list-style-type: none">• A value of <code>true</code> enables the plan.• A value of <code>false</code> disables the plan.

apg_plan_mgmt.set_plan_status

Set a managed plan's status to `Approved`, `Unapproved`, `Rejected`, or `Preferred`.

Syntax

```
apg_plan_mgmt.set_plan_status(
    sql_hash,
    plan_hash,
    status
)
```

Return value

Returns 0 if the setting was successful or -1 if the setting failed.

Parameters

Parameter	Description
sql_hash	The <code>sql_hash</code> ID of the plan's managed SQL statement.
plan_hash	The managed plan's <code>plan_hash</code> ID.
status	A string with one of the following values: <ul style="list-style-type: none"> • 'Approved' • 'Unapproved' • 'Rejected' • 'Preferred' The case you use does not matter, however the status value is set to initial uppercase in the <code>apg_plan_mgmt.dba_plans</code> view. For more information about these values, see status in Reference for the <code>apg_plan_mgmt.dba_plans</code> view (p. 1467) .

[apg_plan_mgmt.update_plans_last_used](#)

Immediately updates the plans table with the `last_used` date stored in shared memory.

Syntax

```
apg_plan_mgmt.update_plans_last_used()
```

Return value

None.

Parameters

None.

Usage notes

Call `update_plans_last_used` to make sure queries against the `dba_plans.last_used` column use the most current information. If the `last_used` date isn't updated immediately, a background process updates the plans table with the `last_used` date once every hour (by default).

For example, if a statement with a certain `sql_hash` begins to run slowly, you can determine which plans for that statement were executed since the performance regression began. To do that, first flush the data in shared memory to disk so that the `last_used` dates are current, and then query for all plans of the `sql_hash` of the statement with the performance regression. In the query, make sure the

`last_used` date is greater than or equal to the date on which the performance regression began. The query identifies the plan or set of plans that might be responsible for the performance regression. You can use `apg_plan_mgmt.get_explain_plan` with `explainOptionList` set to `verbose`, `hashes`. You can also use `apg_plan_mgmt.evolve_plan_baselines` to analyze the plan and any alternative plans that might perform better.

The `update_plans_last_used` function has an effect only on the primary DB instance of the DB cluster.

apg_plan_mgmt.validate_plans

Validate that the optimizer can still recreate plans. The optimizer validates `Approved`, `Unapproved`, and `Preferred` plans, whether the plan is enabled or disabled. `Rejected` plans are not validated. Optionally, you can use the `apg_plan_mgmt.validate_plans` function to delete or disable invalid plans.

Syntax

```
apg_plan_mgmt.validate_plans(
    sql_hash,
    plan_hash,
    action)

apg_plan_mgmt.validate_plans(
    action)
```

Return value

The number of invalid plans.

Parameters

Parameter	Description
<code>sql_hash</code>	The <code>sql_hash</code> ID of the plan's managed SQL statement.
<code>plan_hash</code>	The managed plan's <code>plan_hash</code> ID. Use <code>NULL</code> to mean all plans for the same <code>sql_hash</code> ID value.
<code>action</code>	<p>The action the function is to perform for invalid plans. Valid string values include the following. Case does not matter.</p> <ul style="list-style-type: none"> • <code>'disable'</code> – Each invalid plan is disabled. • <code>'delete'</code> – Each invalid plan is deleted. • <code>'update_plan_hash'</code> – Updates the <code>plan_hash</code> ID for plans that can't be reproduced exactly. It also allows you to fix a plan by rewriting the SQL. You can then register the good plan as an <code>Approved</code> plan for the original SQL. • <code>NULL</code> – The function simply returns the number of invalid plans. No other action is performed. • <code>" "</code> – An empty string produces a message indicating the number of both valid and invalid plans. <p>Any other value is treated like the empty string.</p>

Usage notes

Use the form `validate_plans(action)` to validate all the managed plans for all the managed statements in the entire `apg_plan_mgmt.dba_plans` view.

Use the form `validate_plans(sql_hash, plan_hash, action)` to validate a managed plan specified with `plan_hash`, for a managed statement specified with `sql_hash`.

Use the form `validate_plans(sql_hash, NULL, action)` to validate all the managed plans for the managed statement specified with `sql_hash`.

Publishing Aurora PostgreSQL logs to Amazon CloudWatch Logs

You can configure your Aurora PostgreSQL DB cluster to publish log data to a log group in Amazon CloudWatch Logs. With CloudWatch Logs, you can perform real-time analysis of the log data, and use CloudWatch to create alarms and view metrics. You can use CloudWatch Logs to store your log records in highly durable storage. Unlike with RDS for PostgreSQL which lets you publish both upgrade and postgresql logs, Aurora PostgreSQL supports uploading postgresql logs only to CloudWatch Logs.

Aurora PostgreSQL supports publishing logs to CloudWatch Logs for the following versions:

- 13.3 and higher 13 versions
- 12.4 and higher 12 versions
- 11.6 and higher 11 versions
- 10.11 and higher 10 versions

Note

Be aware of the following:

- If exporting log data is disabled, Aurora doesn't delete existing log groups or log streams.
- If exporting log data is disabled, existing log data remains available in CloudWatch Logs based on log retention settings, which means you still incur charges for stored audit log data. You can delete log streams and log groups using the CloudWatch Logs console, the AWS CLI, or the CloudWatch Logs API.
- If you don't want to export audit logs to CloudWatch Logs, make sure that all methods of exporting audit logs are disabled. These methods are the AWS Management Console, the AWS CLI, and the RDS API.

Publishing logs to Amazon CloudWatch

You can use the AWS Management Console, the AWS CLI, or the RDS API to publish Aurora PostgreSQL logs to Amazon CloudWatch Logs.

Console

You can publish Aurora PostgreSQL logs to CloudWatch Logs with the console.

To publish Aurora PostgreSQL logs from the console

1. Open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. In the navigation pane, choose **Databases**.
3. Choose the Aurora PostgreSQL DB cluster that you want to publish the log data for.

4. Choose **Modify**.
5. In the **Log exports** section, choose **Postgresql log**.
6. Choose **Continue**, and then choose **Modify cluster** on the summary page.

AWS CLI

You can publish Aurora PostgreSQL logs with the AWS CLI. You can run the [modify-db-cluster](#) AWS CLI command with the following options:

- **--db-cluster-identifier**—The DB cluster identifier.
- **--cloudwatch-logs-export-configuration**—The configuration setting for the log types to be set for export to CloudWatch Logs for the DB cluster.

You can also publish Aurora PostgreSQL logs by running one of the following AWS CLI commands:

- [create-db-cluster](#)
- [restore-db-cluster-from-s3](#)
- [restore-db-cluster-from-snapshot](#)
- [restore-db-cluster-to-point-in-time](#)

Run one of these AWS CLI commands with the following options:

- **--db-cluster-identifier**—The DB cluster identifier.
- **--engine**—The database engine.
- **--enable-cloudwatch-logs-exports**—The configuration setting for the log types to be enabled for export to CloudWatch Logs for the DB cluster.

Other options might be required depending on the AWS CLI command that you run.

Example

The following command creates an Aurora PostgreSQL DB cluster to publish log files to CloudWatch Logs.

For Linux, macOS, or Unix:

```
aws rds create-db-cluster \
  --db-cluster-identifier my-db-cluster \
  --engine aurora-postgresql \
  --enable-cloudwatch-logs-exports postgresql
```

For Windows:

```
aws rds create-db-cluster ^
  --db-cluster-identifier my-db-cluster ^
  --engine aurora-postgresql ^
  --enable-cloudwatch-logs-exports postgresql
```

Example

The following command modifies an existing Aurora PostgreSQL DB cluster to publish log files to CloudWatch Logs. The **--cloudwatch-logs-export-configuration** value is a JSON object. The key for this object is `EnableLogTypes`, and its value is `postgresql`.

For Linux, macOS, or Unix:

```
aws rds modify-db-cluster \
--db-cluster-identifier my-db-cluster \
--cloudwatch-logs-export-configuration '{"EnableLogTypes":["postgresql"]}'
```

For Windows:

```
aws rds modify-db-cluster ^
--db-cluster-identifier my-db-cluster ^
--cloudwatch-logs-export-configuration '{\"EnableLogTypes\":[\"postgresql\"]}'
```

Note

When using the Windows command prompt, make sure to escape double quotation marks ("") in JSON code by prefixing them with a backslash (\).

Example

The following example modifies an existing Aurora PostgreSQL DB cluster to disable publishing log files to CloudWatch Logs. The --cloudwatch-logs-export-configuration value is a JSON object. The key for this object is DisableLogTypes, and its value is postgresql.

For Linux, macOS, or Unix:

```
aws rds modify-db-cluster \
--db-cluster-identifier mydbinstance \
--cloudwatch-logs-export-configuration '{"DisableLogTypes":["postgresql"]}'
```

For Windows:

```
aws rds modify-db-cluster ^
--db-cluster-identifier mydbinstance ^
--cloudwatch-logs-export-configuration "{\"DisableLogTypes\":[\"postgresql\"]}"
```

Note

When using the Windows command prompt, you must escape double quotes ("") in JSON code by prefixing them with a backslash (\).

RDS API

You can publish Aurora PostgreSQL logs with the RDS API. You can run the [ModifyDBCluster](#) operation with the following options:

- **DBClusterIdentifier** – The DB cluster identifier.
- **CloudwatchLogsExportConfiguration** – The configuration setting for the log types to be enabled for export to CloudWatch Logs for the DB cluster.

You can also publish Aurora PostgreSQL logs with the RDS API by running one of the following RDS API operations:

- [CreateDBCluster](#)
- [RestoreDBClusterFromS3](#)
- [RestoreDBClusterFromSnapshot](#)

- [RestoreDBClusterToPointInTime](#)

Run the RDS API action with the following parameters:

- `DBClusterIdentifier`—The DB cluster identifier.
- `Engine`—The database engine.
- `EnableCloudwatchLogsExports`—The configuration setting for the log types to be enabled for export to CloudWatch Logs for the DB cluster.

Other parameters might be required depending on the AWS CLI command that you run.

Monitoring log events in Amazon CloudWatch

After enabling Aurora PostgreSQL log events, you can monitor the events in Amazon CloudWatch Logs. For more information about monitoring, see [View log data sent to CloudWatch Logs](#).

A new log group is automatically created for the Aurora DB cluster under the following prefix. In this prefix, `cluster-name` represents the DB cluster name and `log_type` represents the log type.

```
/aws/rds/cluster/cluster-name/log_type
```

For example, suppose that you configure the export function to include the `postgresql` log for a DB cluster named `my-db-cluster`. In this case, PostgreSQL log data is stored in the `/aws/rds/cluster/my-db-cluster/postgresql` log group.

All of the events from all of the DB instances in a DB cluster are pushed to a log group using different log streams.

If a log group with the specified name exists, Aurora uses that log group to export log data for the Aurora DB cluster. You can use automated configuration, such as AWS CloudFormation, to create log groups with predefined log retention periods, metric filters, and customer access. Otherwise, a new log group is automatically created using the default log retention period, **Never Expire**, in CloudWatch Logs. You can use the CloudWatch Logs console, the AWS CLI, or the CloudWatch Logs API to change the log retention period. For more information about changing log retention periods in CloudWatch Logs, see [Change log data retention in CloudWatch Logs](#).

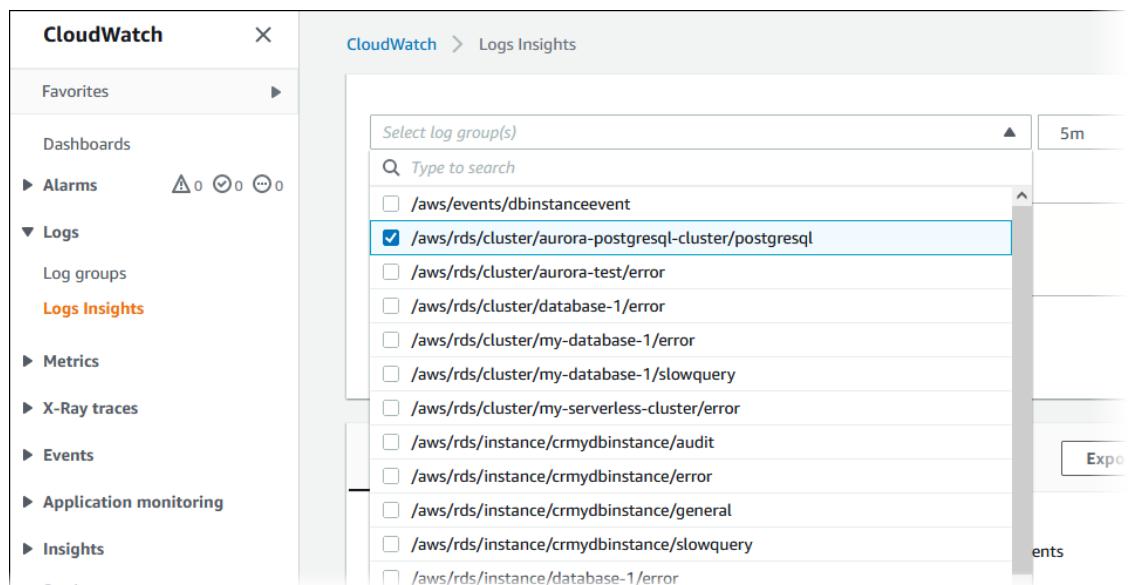
You can use the CloudWatch Logs console, the AWS CLI, or the CloudWatch Logs API to search for information within the log events for a DB cluster. For more information about searching and filtering log data, see [Searching and filtering log data](#).

Analyze Aurora PostgreSQL logs using CloudWatch Logs Insights

After publishing Aurora PostgreSQL logs to CloudWatch Logs, you can analyze logs graphically and create dashboards using CloudWatch Logs Insights.

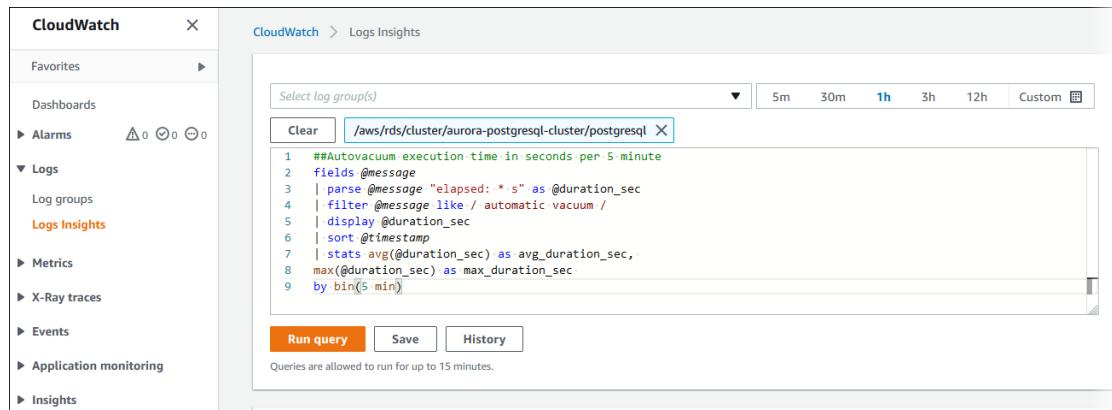
To analyze Aurora PostgreSQL logs with CloudWatch Logs Insights

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, open **Logs** and choose **Log insights**.
3. In **Select log group(s)**, select the log group for your DB cluster.

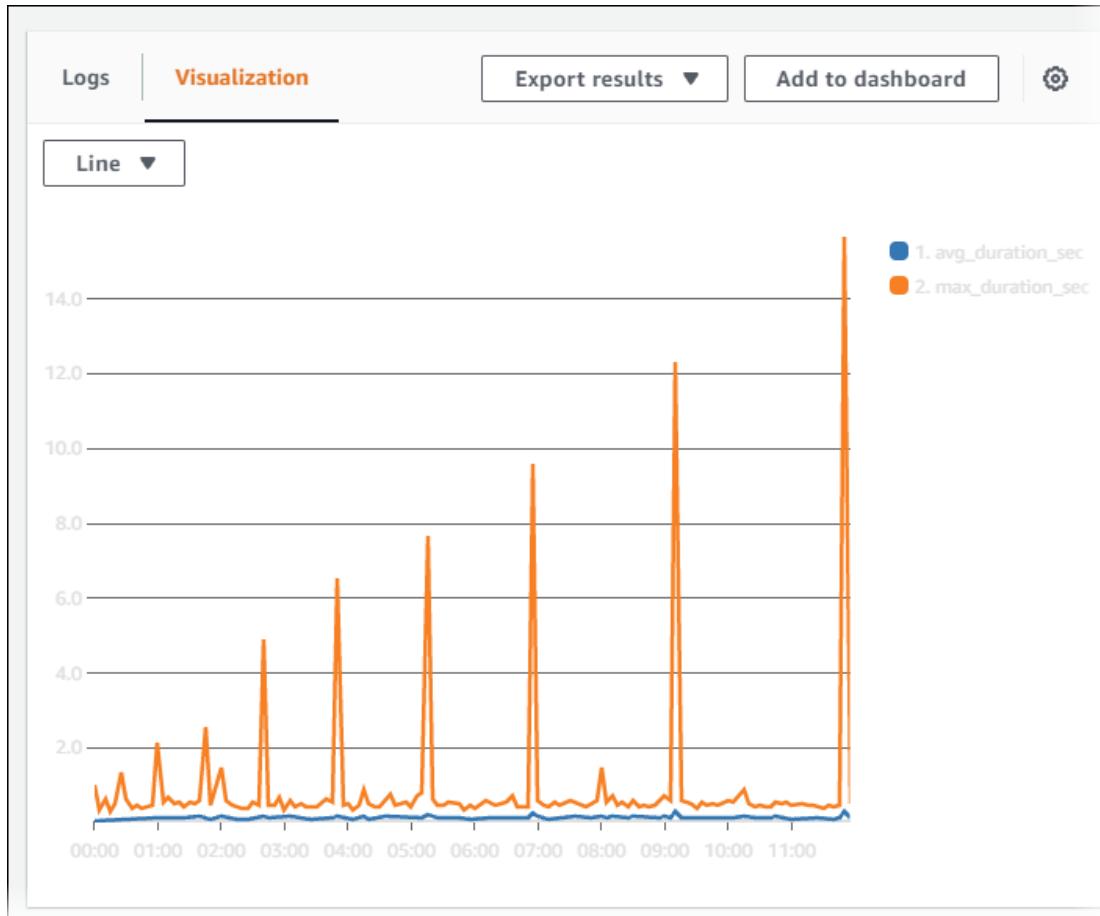


4. In the query editor, delete the query that is currently shown, enter the following, and then choose **Run query**.

```
##Autovacuum execution time in seconds per 5 minute
fields @message
| parse @message "elapsed: * s" as @duration_sec
| filter @message like / automatic vacuum /
| display @duration_sec
| sort @timestamp
| stats avg(@duration_sec) as avg_duration_sec,
max(@duration_sec) as max_duration_sec
by bin(5 min)
```



5. Choose the **Visualization** tab.



6. Choose **Add to dashboard**.
7. In **Select a dashboard**, either select a dashboard or enter a name to create a new dashboard.
8. In **Widget type**, choose a widget type for your visualization.

Add to dashboard

Select a dashboard
Select an existing dashboard or create a new one.
Q AuroraPostgreSQL-LogAnalyze X
Create new

Widget type
Select a widget type to add to the dashboard.
Line ▾

Customize widget title
Widgets get an automatic title. You can optionally customize the title here.
Autovacuum Duration - Avg and Max

Preview
This is how your chart will appear in your dashboard.

Autovacuum Duration - Avg and Max

15.0
10.0
5.0
4.32

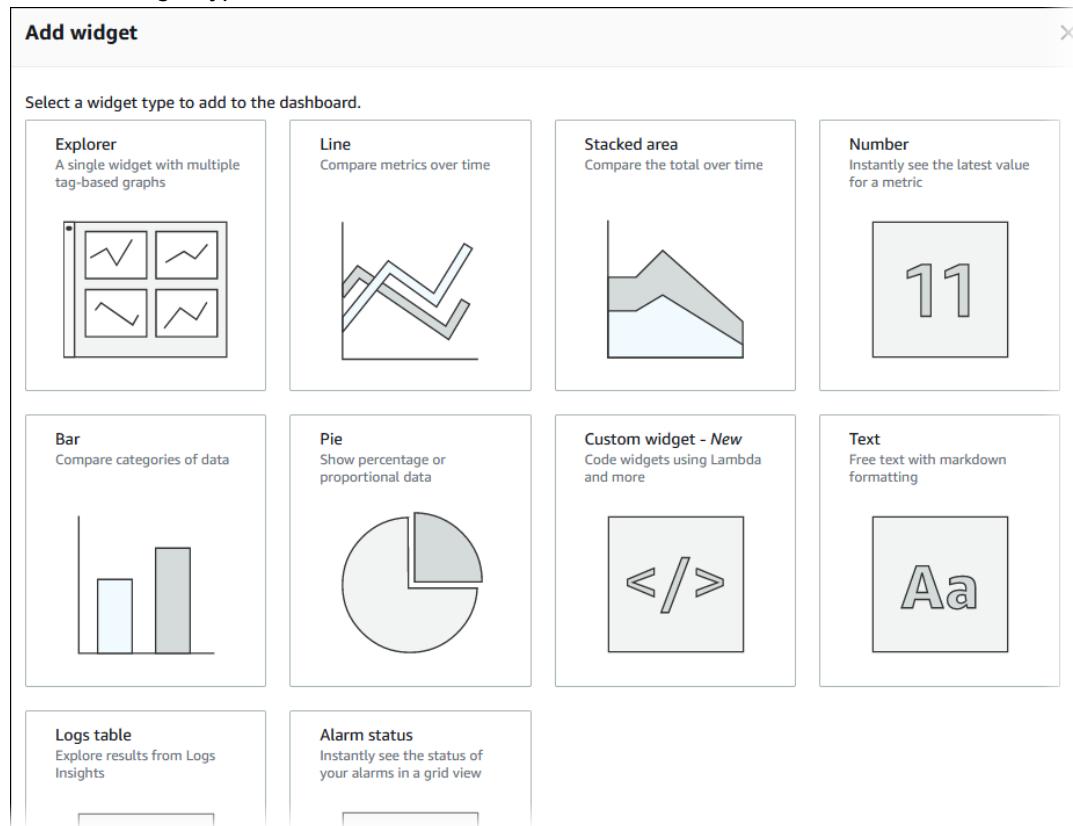
00:00 03:00 06:00 09:00

Cancel Add to dashboard

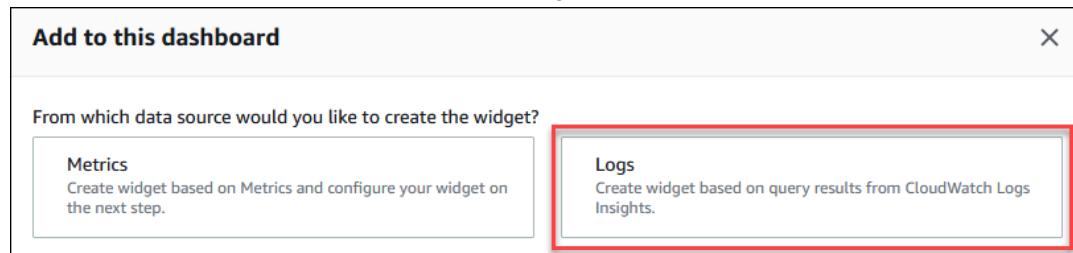
The dialog box for 'Add to dashboard' is open. It shows the search bar with 'AuroraPostgreSQL-LogAnalyze', a 'Create new' button, and a dropdown for 'Widget type' set to 'Line'. Below that is a 'Customize widget title' field containing 'Autovacuum Duration - Avg and Max'. To the right is a preview of the chart with a tooltip showing the value '4.32'. At the bottom are 'Cancel' and 'Add to dashboard' buttons.

9. (Optional) Add more widgets based on your log query results.

- a. Choose **Add widget**.
- b. Choose a widget type, such as **Line**.

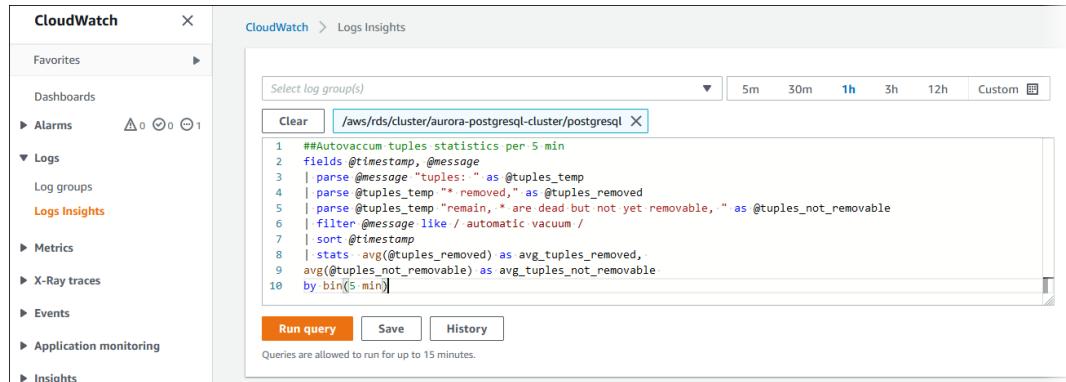


- c. In the **Add to this dashboard** window, choose **Logs**.



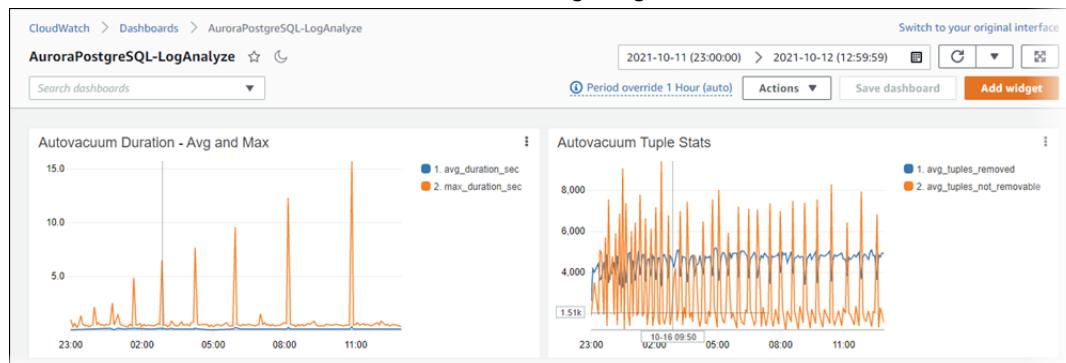
- d. In **Select log group(s)**, select the log group for your DB cluster.
- e. In the query editor, delete the query that is currently shown, enter the following, and then choose **Run query**.

```
##Autovacuum tuples statistics per 5 min
fields @timestamp, @message
| parse @message "tuples: " as @tuples_temp
| parse @tuples_temp "* removed," as @tuples_removed
| parse @tuples_temp "remain, * are dead but not yet removable, " as
@tuples_not_removable
| filter @message like / automatic vacuum /
| sort @timestamp
| stats avg(@tuples_removed) as avg_tuples_removed,
avg(@tuples_not_removable) as avg_tuples_not_removable
by bin(5 min)
```



f. Choose **Create widget**.

Your dashboard should look similar to the following image.



Using machine learning (ML) with Aurora PostgreSQL

With Aurora machine learning, you can add machine learning-based predictions to database applications using the SQL language. Aurora machine learning uses a highly optimized integration between the Aurora database and the AWS machine learning (ML) services SageMaker and Amazon Comprehend.

Benefits of Aurora machine learning include the following:

- You can add ML-based predictions to your existing database applications. You don't need to build custom integrations or learn separate tools. You can embed machine learning processing directly into your SQL query as calls to functions.
- The ML integration is a fast way to enable ML services to work with transactional data. You don't have to move the data out of the database to perform the machine learning operations. You don't have to convert or reimport the results of the machine learning operations to use them in your database application.
- You can use your existing governance policies to control who has access to the underlying data and to the generated insights.

AWS machine learning services are managed services that you set up and run in their own production environments. Currently, Aurora machine learning integrates with Amazon Comprehend for sentiment analysis and SageMaker for a wide variety of ML algorithms.

For general information about Amazon Comprehend, see [Amazon Comprehend](#). For details about using Aurora and Amazon Comprehend together, see [Using Amazon Comprehend for natural language processing \(p. 1497\)](#).

For general information about SageMaker, see [SageMaker](#). For details about using Aurora and SageMaker together, see [Using SageMaker to run your own ML models \(p. 1499\)](#).

Note

Aurora machine learning for PostgreSQL connects an Aurora cluster to SageMaker or Amazon Comprehend services only within the same AWS Region.

Topics

- [Prerequisites for Aurora machine learning \(p. 1495\)](#)
- [Enabling Aurora machine learning \(p. 1495\)](#)
- [Using Amazon Comprehend for natural language processing \(p. 1497\)](#)
- [Exporting data to Amazon S3 for SageMaker model training \(p. 1499\)](#)
- [Using SageMaker to run your own ML models \(p. 1499\)](#)
- [Best practices with Aurora machine learning \(p. 1502\)](#)
- [Monitoring Aurora machine learning \(p. 1506\)](#)
- [PostgreSQL function reference for Aurora machine learning \(p. 1507\)](#)
- [Manually setting up IAM roles for SageMaker and Amazon Comprehend using the AWS CLI \(p. 1509\)](#)

Prerequisites for Aurora machine learning

Aurora machine learning is available on any Aurora cluster that's running a supported Aurora PostgreSQL 10 or higher major version in an AWS Region that supports Aurora machine learning. You can upgrade an Aurora cluster that's running a lower version of Aurora PostgreSQL to a supported higher version if you want to use Aurora machine learning with that cluster. For more information, see [Upgrading the PostgreSQL DB engine for Aurora PostgreSQL \(p. 1681\)](#).

For more information about Regions and Aurora version availability, see [Aurora machine learning \(p. 23\)](#).

Enabling Aurora machine learning

Enabling the ML capabilities involves the following steps.

Topics

- [Setting up IAM access to AWS machine learning services \(p. 1495\)](#)
- [Installing the aws_ml extension for model inference \(p. 1497\)](#)

Setting up IAM access to AWS machine learning services

Before you can access SageMaker and Amazon Comprehend services, you set up AWS Identity and Access Management (IAM) roles. You then add the IAM roles to the Aurora PostgreSQL cluster. These roles authorize the users of your Aurora PostgreSQL database to access AWS ML services.

You can do IAM setup automatically by using the AWS Management Console as shown here. To use the AWS CLI to set up IAM access, see [Manually setting up IAM roles for SageMaker and Amazon Comprehend using the AWS CLI \(p. 1509\)](#).

Automatically connecting an Aurora DB cluster to AWS services using the console

Aurora machine learning requires that your DB cluster use some combination of Amazon S3, SageMaker, and Amazon Comprehend. Amazon Comprehend is for sentiment analysis. SageMaker is for a wide variety of machine learning algorithms.

For Aurora machine learning, you use Amazon S3 only for training SageMaker models. You only need to use Amazon S3 with Aurora machine learning if you don't already have a trained model available and the training is your responsibility.

To connect a DB cluster to these services requires that you set up an AWS Identity and Access Management (IAM) role for each Amazon service. The IAM role enables users of your DB cluster to authenticate with the corresponding service.

To generate the IAM roles for SageMaker, Amazon Comprehend, or Amazon S3, repeat the following steps for each service that you need.

To connect a DB cluster to an Amazon service

1. Sign in to the AWS Management Console and open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. In the navigation pane, choose **Databases**, and then choose the Aurora PostgreSQL DB cluster that you want to use.
3. Choose the **Connectivity & security** tab.
4. Choose **Select a service to connect to this cluster** in the **Manage IAM roles** section.
5. Choose the service that you want to connect to from the list:
 - **Amazon S3**
 - **Amazon Comprehend**
 - **SageMaker**
6. Choose **Connect service**.
7. Enter the required information for the specific service on the **Connect cluster** window:
 - For SageMaker, enter the Amazon Resource Name (ARN) of a SageMaker endpoint.

From the navigation pane of the [SageMaker console](#), choose **Endpoints** and copy the ARN of the endpoint you want to use. For details about what the endpoint represents, see [Deploy a Model in Amazon SageMaker](#).

- For Amazon Comprehend, you don't specify any additional parameters.
- For Amazon S3, enter the ARN of an Amazon S3 bucket to use.

The format of an Amazon S3 bucket ARN is `arn:aws:s3:::bucket_name`. Ensure that the Amazon S3 bucket you use is set up with the requirements for training SageMaker models. When you train a model, your Aurora DB cluster requires permission to export data to the Amazon S3 bucket, and also to import data from the bucket.

For more about an Amazon S3 bucket ARN, see [Specifying resources in a policy](#) in the *Amazon Simple Storage Service User Guide*. For more about using an Amazon S3 bucket with SageMaker, see [Step 1: Create an Amazon S3 bucket](#) in the *Amazon SageMaker Developer Guide*.

8. Choose **Connect service**.
9. Aurora creates a new IAM role and adds it to the DB cluster's list of **Current IAM roles for this cluster**. The IAM role's status is initially **In progress**. The IAM role name is autogenerated with the following pattern for each connected service:

- The Amazon S3 IAM role name pattern is `rds-cluster_ID-S3-role-timestamp`.
- The SageMaker IAM role name pattern is `rds-cluster_ID-SageMaker-role-timestamp`.
- The Amazon Comprehend IAM role name pattern is `rds-cluster_ID-Comprehend-role-timestamp`.

Aurora also creates a new IAM policy and attaches it to the role. The policy name follows a similar naming convention and also has a timestamp.

Installing the aws_ml extension for model inference

After you create the required IAM roles and associate them with the Aurora PostgreSQL DB cluster, install the functions that use the SageMaker and Amazon Comprehend functionality. The `aws_ml` Aurora PostgreSQL extension provides the `aws_sagemaker.invoke_endpoint` function that communicates directly with SageMaker. The `aws_ml` extension also provides the `aws_comprehend.detect_sentiment` function that communicates directly with Amazon Comprehend.

To install these functions in a specific database, enter the following SQL command at a psql prompt.

```
psql>CREATE EXTENSION IF NOT EXISTS aws_ml CASCADE;
```

If you create the `aws_ml` extension in the `template1` default database, then the functions are available in each new database that you create.

To verify the installation, enter the following at a psql prompt.

```
psql>\dx
```

If you set up an IAM role for Amazon Comprehend, you can verify the setup as follows.

```
SELECT sentiment FROM aws_comprehend.detect_sentiment(null, 'I like it!', 'en');
```

When you install the `aws_ml` extension, the `aws_ml` administrative role is created and granted to the `rds_superuser` role. Separate schemas are also created for the `aws_sagemaker` service and for the `aws_comprehend` service. The `rds_superuser` role is made the `OWNER` of both of these schemas.

For users or roles to obtain access to the functions in the `aws_ml` extension, grant `EXECUTE` privilege on those functions. You can subsequently `REVOKE` the privileges, if needed. `EXECUTE` privileges are revoked from `PUBLIC` on the functions of these schemas by default. In a multi-tenant database configuration, to prevent tenants from accessing the functions use `REVOKE USAGE` on one or more of the ML service schemas.

For a reference to the installed functions of the `aws_ml` extension, see [PostgreSQL function reference for Aurora machine learning \(p. 1507\)](#).

Using Amazon Comprehend for natural language processing

Amazon Comprehend uses machine learning to find insights and relationships in text. Amazon Comprehend uses natural language processing to extract insights about the content of documents. It

develops insights by recognizing the entities, key phrases, language, sentiments, and other common elements in a document. You can use this Aurora machine learning service with very little machine learning experience.

Aurora machine learning uses Amazon Comprehend for sentiment analysis of text that is stored in your database. A *sentiment* is an opinion expressed in text. Sentiment analysis identifies and categorizes sentiments to determine if the attitude towards something (such as a topic or product) is positive, negative, or neutral.

Note

Amazon Comprehend is currently available only in some AWS Regions. To check in which AWS Regions you can use Amazon Comprehend, see [the AWS Region table](#) page on the AWS site.

For example, using Amazon Comprehend you can analyze contact center call-in documents to detect caller sentiment and better understand caller-agent dynamics. You can find a further description in the post [Analyzing contact center calls](#) on the AWS Machine Learning blog.

You can also combine sentiment analysis with the analysis of other information in your database using a single query. For example, you can detect the average sentiment of call-in center documents for issues that combine the following:

- Open for more than 30 days.
- About a specific product or feature.
- Made by the customers who have the greatest social media influence.

Using Amazon Comprehend from Aurora machine learning is as easy as calling a SQL function. When you installed the `aws_ml` extension ([Installing the aws_ml extension for model inference \(p. 1497\)](#)), it provides the `aws_comprehend.detect_sentiment (p. 1507)` function to perform sentiment analysis through Amazon Comprehend.

For each text fragment that you analyze, this function helps you determine the sentiment and the confidence level. A typical Amazon Comprehend query looks for table rows where the sentiment has a certain value (POSITIVE or NEGATIVE), with a confidence level greater than a certain percent.

For example, the following query shows how to determine the average sentiment of documents in a database table, `myTable.document`. The query considers only documents where the confidence of the assessment is at least 80 percent. In the following example, English (en) is the language of the sentiment text.

```
SELECT AVG(CASE s.sentiment
    WHEN 'POSITIVE' then 1
    WHEN 'NEGATIVE' then -1
    ELSE 0 END) as avg_sentiment, COUNT(*) AS total
FROM myTable, aws_comprehend.detect_sentiment (myTable.document, 'en') s
WHERE s.confidence >= 0.80;
```

To avoid your being charged for sentiment analysis more than once per table row, you can materialize the results of the analysis once per row. Do this on the rows of interest. In the following example, French (fr) is the language of the sentiment text.

```
-- *Example:* Update the sentiment and confidence of French text.
--
UPDATE clinician_notes
SET sentiment = (aws_comprehend.detect_sentiment (french_notes, 'fr')).sentiment,
    confidence = (aws_comprehend.detect_sentiment (french_notes, 'fr')).confidence
WHERE
    clinician_notes.french_notes IS NOT NULL AND
```

```
LENGTH(TRIM(clinician_notes.french_notes)) > 0 AND
clinician_notes.sentiment IS NULL;
```

For more information on optimizing your function calls, see [Best practices with Aurora machine learning \(p. 1502\)](#).

For information about parameters and return types for the sentiment detection function, see [aws_comprehend.detect_sentiment \(p. 1507\)](#).

Exporting data to Amazon S3 for SageMaker model training

Depending on how your team divides the machine learning tasks, you might not perform model training. If someone else provides the SageMaker model for you, you can skip this section.

To train SageMaker models, you export data to an Amazon S3 bucket. The Amazon S3 bucket is used by SageMaker to train your model before it is deployed. You can query data from an Aurora PostgreSQL DB cluster and save it directly into text files stored in an Amazon S3 bucket. Then SageMaker consumes the data from the Amazon S3 bucket for training. For more about SageMaker model training, see [Train a model with Amazon SageMaker](#).

Note

When you create an S3 bucket for SageMaker model training or batch scoring, always include the text `sagemaker` in the S3 bucket name. For more information about creating an S3 bucket for SageMaker, see [Step 1: Create an Amazon S3 bucket](#).

For more information about exporting your data, see [Exporting data from an Aurora PostgreSQL DB cluster to Amazon S3 \(p. 1450\)](#).

Using SageMaker to run your own ML models

SageMaker is a fully managed machine learning service. With SageMaker, data scientists and developers build and train machine learning models. Then they can directly deploy the models into a production-ready hosted environment.

SageMaker provides access to your data sources so that you can perform exploration and analysis without managing the hardware infrastructure for servers. SageMaker also provides common machine learning algorithms that are optimized to run efficiently against extremely large datasets in a distributed environment. With native support for bring-your-own-algorithms and frameworks, SageMaker offers flexible distributed training options that adjust to your specific workflows.

Note

Currently, Aurora machine learning supports any SageMaker endpoint that can read and write the comma-separated value (CSV) format, through a `ContentType` value of `text/csv`. The built-in SageMaker algorithms that currently accept this format are Random Cut Forest, Linear Learner, and XGBoost.

Be sure to deploy the model you are using in the same AWS Region as your Aurora PostgreSQL cluster. Aurora machine learning always invokes SageMaker endpoints in the same AWS Region as your Aurora cluster.

When you install the `aws_ml` extension (as described in [Installing the aws_ml extension for model inference \(p. 1497\)](#)), it provides the `aws_sagemaker.invoke_endpoint (p. 1508)` function. You use this function to invoke your SageMaker model and perform model inference directly from within your SQL database application.

Topics

- [Creating a user-defined function to invoke a SageMaker model \(p. 1500\)](#)
- [Passing an array as input to a SageMaker model \(p. 1501\)](#)
- [Specifying batch size when invoking a SageMaker model \(p. 1501\)](#)
- [Invoking a SageMaker model that has multiple outputs \(p. 1501\)](#)

Creating a user-defined function to invoke a SageMaker model

Create a separate user-defined function to call `aws_sagemaker.invoke_endpoint` for each of your SageMaker models. Your user-defined function represents the SageMaker endpoint hosting the model. The `aws_sagemaker.invoke_endpoint` function runs within the user-defined function. User-defined functions provide many advantages:

- You can give your ML model its own name instead of only calling `aws_sagemaker.invoke_endpoint` for all of your ML models.
- You can specify the model endpoint URL in just one place in your SQL application code.
- You can control `EXECUTE` privileges to each ML function independently.
- You can declare the model input and output types using SQL types. SQL enforces the number and type of arguments passed to your ML model and performs type conversion if necessary. Using SQL types will also translate SQL `NULL` to the appropriate default value expected by your ML model.
- You can reduce the maximum batch size if you want to return the first few rows a little faster.

To specify a user-defined function, use the SQL data definition language (DDL) statement `CREATE FUNCTION`. When you define the function, you specify the following:

- The input parameters to the model.
- The specific SageMaker endpoint to invoke.
- The return type.

The user-defined function returns the inference computed by the SageMaker endpoint after running the model on the input parameters. The following example creates a user-defined function for an SageMaker model with two input parameters.

```
CREATE FUNCTION classify_event (IN arg1 INT, IN arg2 DATE, OUT category INT)
AS $$$
    SELECT aws_sagemaker.invoke_endpoint (
        'sagemaker_model_endpoint_name', NULL,
        arg1, arg2
        )::INT
    -- model inputs are separate arguments
    -- cast the output to INT
$$$ LANGUAGE SQL PARALLEL SAFE COST 5000;
```

Note the following:

- The `aws_sagemaker.invoke_endpoint` function input can be one or more parameters of any data type.

For more details about parameters, see the [aws_sagemaker.invoke_endpoint \(p. 1508\)](#) function reference.

- This example uses an INT output type. If you cast the output from a `varchar` type to a different type, then it must be cast to a PostgreSQL builtin scalar type such as `INTEGER`, `REAL`, `FLOAT`, or `NUMERIC`. For more information about these types, see [Data types](#) in the PostgreSQL documentation.

- Specify `PARALLEL SAFE` to enable parallel query processing. For more information, see [Exploiting parallel query processing \(p. 1504\)](#).
- Specify `COST 5000` to estimate the cost of running the function. Use a positive number giving the estimated run cost for the function, in units of `cpu_operator_cost`.

Passing an array as input to a SageMaker model

The [aws_sagemaker.invoke_endpoint \(p. 1508\)](#) function can have up to 100 input parameters, which is the limit for PostgreSQL functions. If the SageMaker model requires more than 100 parameters of the same type, pass the model parameters as an array.

The following example creates a user-defined function that passes an array as input to the SageMaker regression model.

```
CREATE FUNCTION regression_model (params REAL[], OUT estimate REAL)
AS $$
    SELECT aws_sagemaker.invoke_endpoint (
        'sagemaker_model_endpoint_name', NULL,
        params
        )::REAL
    -- model input parameters as an array
    -- cast output to REAL
$$ LANGUAGE SQL PARALLEL SAFE COST 5000;
```

Specifying batch size when invoking a SageMaker model

The following example creates a user-defined function for a SageMaker model that sets the batch size default to `NULL`. The function also allows you to provide a different batch size when you invoke it.

```
CREATE FUNCTION classify_event (
    IN event_type INT, IN event_day DATE, IN amount REAL, -- model inputs
    max_rows_per_batch INT DEFAULT NULL, -- optional batch size limit
    OUT category INT) -- model output
AS $$
    SELECT aws_sagemaker.invoke_endpoint (
        'sagemaker_model_endpoint_name', max_rows_per_batch,
        event_type, event_day, COALESCE(amount, 0.0)
        )::INT -- casts output to type INT
$$ LANGUAGE SQL PARALLEL SAFE COST 5000;
```

Note the following:

- Use the optional `max_rows_per_batch` parameter to provide control of the number of rows for a batch-mode function invocation. If you use a value of `NULL`, then the query optimizer automatically chooses the maximum batch size. For more information, see [Optimizing batch-mode execution for Aurora machine learning function calls \(p. 1502\)](#).
- By default, passing `NULL` as a parameter's value is translated to an empty string before passing to SageMaker. For this example the inputs have different types.
- If you have a non-text input, or text input that needs to default to some value other than an empty string, use the `COALESCE` statement. Use `COALESCE` to translate `NULL` to the desired null replacement value in the call to `aws_sagemaker.invoke_endpoint`. For the `amount` parameter in this example, a `NULL` value is converted to `0.0`.

Invoking a SageMaker model that has multiple outputs

The following example creates a user-defined function for a SageMaker model that returns multiple outputs. Your function needs to cast the output of the `aws_sagemaker.invoke_endpoint` function to

a corresponding data type. For example, you could use the built-in PostgreSQL point type for (x,y) pairs or a user-defined composite type.

This user-defined function returns values from a model that returns multiple outputs by using a composite type for the outputs.

```
CREATE TYPE company_forecasts AS (
    six_month_estimated_return real,
    one_year_bankruptcy_probability float);
CREATE FUNCTION analyze_company (
    IN free_cash_flow NUMERIC(18, 6),
    IN debt NUMERIC(18,6),
    IN max_rows_per_batch INT DEFAULT NULL,
    OUT prediction company_forecasts)
AS $$
    SELECT (aws_sagemaker.invoke_endpoint(
        'endpt_name', max_rows_per_batch,
        free_cash_flow, debt))::company_forecasts;
$$ LANGUAGE SQL PARALLEL SAFE COST 5000;
```

For the composite type, use fields in the same order as they appear in the model output and cast the output of `aws_sagemaker.invoke_endpoint` to your composite type. The caller can extract the individual fields either by name or with PostgreSQL `".*"` notation.

Best practices with Aurora machine learning

Most of the work in an `aws_ml` function call happens within the external Aurora machine learning service. This separation allows you to scale the resources for the machine learning service independent of your Aurora cluster. Within Aurora, you mostly focus on making the user-defined function calls themselves as efficient as possible. Some aspects that you can influence from your Aurora cluster include:

- The `max_rows_per_batch` setting for calls to the `aws_ml` functions.
- The number of virtual CPUs of the database instance, which determines the maximum degree of parallelism that the database might use when running your ML functions.
- the PostgreSQL parameters that control parallel query processing.

Topics

- [Optimizing batch-mode execution for Aurora machine learning function calls \(p. 1502\)](#)
- [Exploiting parallel query processing \(p. 1504\)](#)
- [Using materialized views and materialized columns \(p. 1505\)](#)

Optimizing batch-mode execution for Aurora machine learning function calls

Typically PostgreSQL runs functions one row at a time. Aurora machine learning can minimize this overhead by combining the calls to the external Aurora machine learning service for many rows into batches with an approach called *batch-mode execution*. In batch mode, Aurora machine learning receives the responses for a batch of input rows, and then delivers the responses back to the running query one row at a time. This optimization improves the throughput of your Aurora queries without limiting the PostgreSQL query optimizer.

Aurora automatically uses batch mode if the function is referenced from the `SELECT` list, a `WHERE` clause, or a `HAVING` clause. Note that top-level simple `CASE` expressions are eligible for batch-mode execution.

Top-level searched CASE expressions are also eligible for batch-mode execution provided that the first WHEN clause is a simple predicate with a batch-mode function call.

Your user-defined function must be a LANGUAGE SQL function and should specify PARALLEL SAFE and COST 5000.

Topics

- [Function migration from the SELECT statement to the FROM clause \(p. 1503\)](#)
- [Using the max_rows_per_batch parameter \(p. 1503\)](#)
- [Verifying batch-mode execution \(p. 1504\)](#)

Function migration from the SELECT statement to the FROM clause

Usually, an aws_ml function that is eligible for batch-mode execution is automatically migrated by Aurora to the FROM clause.

The migration of eligible batch-mode functions to the FROM clause can be examined manually on a per-query level. To do this, you use EXPLAIN statements (and ANALYZE and VERBOSE) and find the "Batch Processing" information below each batch-mode Function Scan. You can also use EXPLAIN (with VERBOSE) without running the query. You then observe whether the calls to the function appear as a Function Scan under a nested loop join that was not specified in the original statement.

In the following example, the presence of the nested loop join operator in the plan shows that Aurora migrated the anomaly_score function. It migrated this function from the SELECT list to the FROM clause, where it's eligible for batch-mode execution.

```
EXPLAIN (VERBOSE, COSTS false)
SELECT anomaly_score(ts.R.description) from ts.R;
          QUERY PLAN
-----
Nested Loop
  Output: anomaly_score((r.description)::text)
    -> Seq Scan on ts.r
      Output: r.id, r.description, r.score
    -> Function Scan on public.anomaly_score
      Output: anomaly_score.anomaly_score
      Function Call: anomaly_score((r.description)::text)
```

To disable batch-mode execution, set the `apg_enable_function_migration` parameter to `false`. This prevents the migration of aws_ml functions from the SELECT to the FROM clause. The following shows how.

```
SET apg_enable_function_migration = false;
```

The `apg_enable_function_migration` parameter is a Grand Unified Configuration (GUC) parameter that is recognized by the Aurora PostgreSQL `apg_plan_mgmt` extension for query plan management. To disable function migration in a session, use query plan management to save the resulting plan as an approved plan. At runtime, query plan management enforces the approved plan with its `apg_enable_function_migration` setting. This enforcement occurs regardless of the `apg_enable_function_migration` GUC parameter setting. For more information, see [Managing query execution plans for Aurora PostgreSQL \(p. 1460\)](#).

Using the max_rows_per_batch parameter

The `max_rows_per_batch` parameter of the [aws_sagemaker.invoke_endpoint \(p. 1508\)](#) and [aws_comprehend.detect_sentiment \(p. 1507\)](#) functions influences how many rows are transferred to

the Aurora machine learning service. The larger the dataset processed by the user-defined function, the larger you can make the batch size.

Batch-mode functions improve efficiency by building batches of rows that spread the cost of the Aurora machine learning function calls over a large number of rows. However, if a `SELECT` statement finishes early due to a `LIMIT` clause, then the batch can be constructed over more rows than the query uses. This approach can result in additional charges to your AWS account. To gain the benefits of batch-mode execution but avoid building batches that are too large, use a smaller value for the `max_rows_per_batch` parameter in your function calls.

If you do an `EXPLAIN (VERBOSE, ANALYZE)` of a query that uses batch-mode execution, you see a `FunctionScan` operator that is below a nested loop join. The number of loops reported by `EXPLAIN` tells you the number of times a row was fetched from the `FunctionScan` operator. If a statement uses a `LIMIT` clause, the number of fetches is consistent. To optimize the size of the batch, set the `max_rows_per_batch` parameter to this value. However, if the batch-mode function is referenced in a predicate in the `WHERE` clause or `HAVING` clause, then you probably can't know the number of fetches in advance. In this case, use the loops as a guideline and experiment with `max_rows_per_batch` to find a setting that optimizes performance.

Verifying batch-mode execution

To see if a function ran in batch mode, use `EXPLAIN ANALYZE`. If batch-mode execution was used, then the query plan will include the information in a "Batch Processing" section.

```
EXPLAIN ANALYZE SELECT user-defined-function();
Batch Processing: num batches=1 avg/min/max batch size=3333.000/3333.000/3333.000
                  avg/min/max batch call time=146.273/146.273/146.273
```

In this example, there was 1 batch that contained 3,333 rows, which took 146.273 ms to process. The "Batch Processing" section shows the following:

- How many batches there were for this function scan operation
- The batch size average, minimum, and maximum
- The batch execution time average, minimum, and maximum

Typically the final batch is smaller than the rest, which often results in a minimum batch size that is much smaller than the average.

To return the first few rows more quickly, set the `max_rows_per_batch` parameter to a smaller value.

To reduce the number of batch mode calls to the ML service when you use a `LIMIT` in your user-defined function, set the `max_rows_per_batch` parameter to a smaller value.

Exploiting parallel query processing

To dramatically increase performance when processing a large number of rows, you can combine parallel query processing with batch mode processing. You can use parallel query processing for `SELECT`, `CREATE TABLE AS SELECT`, and `CREATE MATERIALIZED VIEW` statements.

Note

PostgreSQL doesn't yet support parallel query for data manipulation language (DML) statements.

Parallel query processing occurs both within the database and within the ML service. The number of cores in the instance class of the database limits the degree of parallelism that can be used when running a query. The database server can construct a parallel query execution plan that partitions the

task among a set of parallel workers. Then each of these workers can build batched requests containing tens of thousands of rows (or as many as are allowed by each service).

The batched requests from all of the parallel workers are sent to the endpoint for the AWS service (SageMaker, for example). Thus, the number and type of instances behind the AWS service endpoint also limits the degree of parallelism that can be usefully exploited. Even a two-core instance class can benefit significantly from parallel query processing. However, to fully exploit parallelism at higher K degrees, you need a database instance class that has at least K cores. You also need to configure the AWS service so that it can process K batched requests in parallel. For SageMaker, you need to configure the SageMaker endpoint for your ML model to have K initial instances of a sufficiently high-performing instance class.

To exploit parallel query processing, you can set the `parallel_workers` storage parameter of the table that contains the data that you plan to pass. You set `parallel_workers` to a batch-mode function such as `aws_comprehend.detect_sentiment`. If the optimizer chooses a parallel query plan, the AWS ML services can be called both in batch and in parallel. You can use the following parameters with the `aws_comprehend.detect_sentiment` function to get a plan with four-way parallelism.

```
-- If you change either of the following two parameters, you must restart
-- the database instance for the changes to take effect.
--
-- SET max_worker_processes to 8; -- default value is 8
-- SET max_parallel_workers to 8; -- not greater than max_worker_processes

--
SET max_parallel_workers_per_gather to 4; -- not greater than max_parallel_workers

-- You can set the parallel_workers storage parameter on the table that the data
-- for the ML function is coming from in order to manually override the degree of
-- parallelism that would otherwise be chosen by the query optimizer
--
ALTER TABLE yourTable SET (parallel_workers = 4);

-- Example query to exploit both batch-mode execution and parallel query
--
EXPLAIN (verbose, analyze, buffers, hashes)
SELECT aws_comprehend.detect_sentiment(description, 'en')).*
FROM yourTable
WHERE id < 100;
```

For more about controlling parallel query, see [Parallel plans](#) in the PostgreSQL documentation.

Using materialized views and materialized columns

When you invoke an AWS service such as SageMaker or Amazon Comprehend from your database, your account is charged according to the pricing policy of that service. To minimize charges to your account, you can materialize the result of calling the AWS service into a materialized column so that the AWS service is not called more than once per input row. If desired, you can add a `materializedAt` timestamp column to record the time at which the columns were materialized.

The latency of an ordinary single-row `INSERT` statement is typically much less than the latency of calling a batch-mode function. Thus, you might not be able to meet the latency requirements of your application if you invoke the batch-mode function for every single-row `INSERT` that your application performs. To materialize the result of calling an AWS service into a materialized column, high-performance applications generally need to populate the materialized columns. To do this, they periodically issue an `UPDATE` statement that operates on a large batch of rows at the same time.

`UPDATE` takes a row-level lock that can impact a running application. So you might need to use `SELECT ... FOR UPDATE SKIP LOCKED`, or use `MATERIALIZED VIEW`.

Analytics queries that operate on a large number of rows in real time can combine batch-mode materialization with real-time processing. To do this, these queries assemble a `UNION ALL` of the pre-

materialized results with a query over the rows that don't yet have materialized results. In some cases, such a `UNION ALL` is needed in multiple places, or the query is generated by a third-party application. If so, you can create a `VIEW` to encapsulate the `UNION ALL` operation so this detail isn't exposed to the rest of the SQL application.

You can use a materialized view to materialize the results of an arbitrary `SELECT` statement at a snapshot in time. You can also use it to refresh the materialized view at any time in the future. Currently PostgreSQL doesn't support incremental refresh, so each time the materialized view is refreshed the materialized view is fully recomputed.

You can refresh materialized views with the `CONCURRENTLY` option, which updates the contents of the materialized view without taking an exclusive lock. Doing this allows a SQL application to read from the materialized view while it's being refreshed.

Monitoring Aurora machine learning

To monitor the functions in the `aws_ml` package, set the `track_functions` parameter and then query the [PostgreSQL pg_stat_user_functions view](#).

For information about monitoring the performance of the SageMaker operations called from Aurora machine learning functions, see [Monitor Amazon SageMaker](#).

To set `track_functions` at the session level, run the following.

```
SET track_functions = 'all';
```

Use one of the following values:

- `all` – Track C language functions and SQL language functions that aren't placed inline. To track the `aws_ml` functions, use `all` because these functions are implemented in C.
- `p1` – Track only procedural-language functions.
- `none` – Disable function statistics tracking.

After enabling `track_functions` and running your user-defined ML function, query the `pg_stat_user_functions` view to get information. The view includes the number of calls, `total_time` and `self_time` for each function. To view the statistics for the `aws_sagemaker.invoke_endpoint` and `aws_comprehend.detect_sentiment` functions, filter the results by schema names starting with `aws_`.

```
run your statement
SELECT * FROM pg_stat_user_functions WHERE schemaname LIKE 'aws_%';
SELECT pg_stat_reset(); -- To clear statistics
```

To find the names of your SQL functions that call the `aws_sagemaker.invoke_endpoint` function, query the source code of the functions in the [PostgreSQL pg_proc catalog](#) table.

```
SELECT proname FROM pg_proc WHERE prosrc LIKE '%invoke_endpoint%';
```

Using query plan management to monitor ML functions

If you captured plans using the `apg_plan_mgmt` extension of query plan management, you can then search through all the statements in your workload that refer to these function names. In your search, you can check `plan_outline` to see if batch-mode execution was used. You can also list statement

statistics such as execution time and plan cost. Plans that use batch-mode function scans contain a `FuncScan` operator in the plan outline. Functions that aren't run as a join don't contain a `FuncScan` operator.

For more about query plan management, see [Managing query execution plans for Aurora PostgreSQL \(p. 1460\)](#).

To find calls to the `aws_sagemaker.invoke_endpoint` function that don't use batch mode, use the following statement.

```
\dx apg_plan_mgmt

SELECT sql_hash, plan_hash, status, environment_variables,
       sql_text::varchar(50), plan_outline
FROM pg_proc, apg_plan_mgmt.dba_plans
WHERE
    prosrc LIKE '%invoke_endpoint%' AND
    sql_text LIKE '%' || proname || '%' AND
    plan_outline NOT LIKE '%FuncScan%';
```

The example preceding searches all statements in your workload that call SQL functions that in turn call the `aws_sagemaker.invoke_endpoint` function.

To obtain detailed runtime statistics for each of these statements, call the `apg_plan_mgmt.get_explain_stmt` function.

```
SELECT apg_plan_mgmt.get_explain_stmt(sql_hash, plan_hash, 'analyze,verbose,buffers')
FROM pg_proc, apg_plan_mgmt.dba_plans
WHERE
    prosrc LIKE '%invoke_endpoint%' AND
    sql_text LIKE '%' || proname || '%' AND
    plan_outline NOT LIKE '%FuncScan%';
```

PostgreSQL function reference for Aurora machine learning

Functions

- [aws_comprehend.detect_sentiment \(p. 1507\)](#)
- [aws_sagemaker.invoke_endpoint \(p. 1508\)](#)

aws_comprehend.detect_sentiment

Performs sentiment analysis using Amazon Comprehend. For more about usage, see [Using Amazon Comprehend for natural language processing \(p. 1497\)](#).

Syntax

```
aws_comprehend.detect_sentiment (
    IN input_text varchar,
    IN language_code varchar,
    IN max_rows_per_batch int,
    OUT sentiment varchar,
    OUT confidence real)
)
```

Input Parameters

input_text

The text to detect sentiment on.

language_code

The language of the `input_text`. For valid values, see [Languages supported in Amazon Comprehend](#).

max_rows_per_batch

The maximum number of rows per batch for batch-mode processing. For more information, see [Optimizing batch-mode execution for Aurora machine learning function calls \(p. 1502\)](#).

Output Parameters

sentiment

The sentiment of the text. Valid values are POSITIVE, NEGATIVE, NEUTRAL, or MIXED.

confidence

The degree of confidence in the `sentiment` value. Values range between 1.0 for 100% to 0.0 for 0%.

[aws_sagemaker.invoke_endpoint](#)

After you train a model and deploy it into production using SageMaker services, your client applications use the `aws_sagemaker.invoke_endpoint` function to get inferences from the model. The model must be hosted at the specified endpoint and must be in the same AWS Region as the database instance. For more about usage, see [Using SageMaker to run your own ML models \(p. 1499\)](#).

Syntax

```
aws_sagemaker.invoke_endpoint(
    IN endpoint_name varchar,
    IN max_rows_per_batch int,
    VARIADIC model_input "any",
    OUT model_output varchar
)
```

Input Parameters

endpoint_name

An endpoint URL that is AWS Region-independent.

max_rows_per_batch

The maximum number of rows per batch for batch-mode processing. For more information, see [Optimizing batch-mode execution for Aurora machine learning function calls \(p. 1502\)](#).

model_input

One or more input parameters for the ML model. These can be any data type.

PostgreSQL allows you to specify up to 100 input parameters for a function. Array data types must be one-dimensional, but can contain as many elements as are expected by the SageMaker model.

The number of inputs to a SageMaker model is bounded only by the SageMaker 5 MB message size limit.

Output Parameters

model_output

The SageMaker model's output parameter, as text.

Usage Notes

The `aws_sagemaker.invoke_endpoint` function connects only to a model endpoint in the same AWS Region. If your database instance has replicas in multiple AWS Regions, always deploy each Amazon SageMaker model to all of those AWS Regions.

Calls to `aws_sagemaker.invoke_endpoint` are authenticated using the SageMaker IAM role for the database instance.

SageMaker model endpoints are scoped to an individual account and are not public. The `endpoint_name` URL doesn't contain the account ID. SageMaker determines the account ID from the authentication token that is supplied by the SageMaker IAM role of the database instance.

Manually setting up IAM roles for SageMaker and Amazon Comprehend using the AWS CLI

Note

If you use the AWS Management Console, AWS does the IAM setup for you automatically. In this case, you can skip the following information and follow the procedure in [Automatically connecting an Aurora DB cluster to AWS services using the console \(p. 1496\)](#).

Setting up the IAM roles for SageMaker or Amazon Comprehend using the AWS CLI or the RDS API consists of the following steps:

1. Create an IAM policy to specify which SageMaker endpoints can be invoked by your Aurora PostgreSQL cluster or to enable access to Amazon Comprehend.
2. Create an IAM role to permit your Aurora PostgreSQL database cluster to access AWS ML services. Also attach the IAM policy created preceding to the IAM role created here.
3. Associate the IAM role that you created preceding to the Aurora PostgreSQL database cluster to permit access to AWS ML services.

Topics

- [Creating an IAM policy to access SageMaker using the AWS CLI \(p. 1509\)](#)
- [Creating an IAM policy to access Amazon Comprehend using the AWS CLI \(p. 1510\)](#)
- [Creating an IAM role to access SageMaker and Amazon Comprehend \(p. 1511\)](#)
- [Associating an IAM role with an Aurora PostgreSQL DB cluster using the AWS CLI \(p. 1511\)](#)

Creating an IAM policy to access SageMaker using the AWS CLI

Note

Aurora can create the IAM policy for you automatically. You can skip the following information and use the procedure in [Automatically connecting an Aurora DB cluster to AWS services using the console \(p. 1496\)](#).

The following policy adds the permissions required by Aurora PostgreSQL to invoke a SageMaker function on your behalf. You can specify all of your SageMaker endpoints that you need your database applications to access from your Aurora PostgreSQL cluster in a single policy.

Note

This policy enables you to specify the AWS Region for a SageMaker endpoint. However, an Aurora PostgreSQL cluster can only invoke SageMaker models deployed in the same AWS Region as the cluster.

```
{ "Version": "2012-10-17", "Statement": [ { "Sid": "AllowAuroraToInvokeRCFEndPoint", "Effect": "Allow", "Action": "sagemaker:InvokeEndpoint", "Resource": "arn:aws:sagemaker:region:123456789012:endpoint/endpointName" } ] }
```

The following AWS CLI command creates an IAM policy with these options.

```
aws iam create-policy --policy-name policy_name --policy-document '{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "AllowAuroraToInvokeRCFEndPoint",  
            "Effect": "Allow",  
            "Action": "sagemaker:InvokeEndpoint",  
            "Resource": "arn:aws:sagemaker:region:123456789012:endpoint/endpointName"  
        }  
    ]  
}'
```

For the next step, see [Creating an IAM role to access SageMaker and Amazon Comprehend \(p. 1511\)](#).

Creating an IAM policy to access Amazon Comprehend using the AWS CLI

Note

Aurora can create the IAM policy for you automatically. You can skip the following information and use the procedure in [Automatically connecting an Aurora DB cluster to AWS services using the console \(p. 1496\)](#).

The following policy adds the permissions required by Aurora PostgreSQL to invoke Amazon Comprehend on your behalf.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "AllowAuroraToInvokeComprehendDetectSentiment",  
            "Effect": "Allow",  
            "Action": [  
                "comprehend:DetectSentiment",  
                "comprehend:BatchDetectSentiment"  
            ],  
            "Resource": "*"  
        }  
    ]  
}
```

To create an IAM policy to grant access to Amazon Comprehend

1. Open the [IAM management console](#).

2. In the navigation pane, choose **Policies**.
3. Choose **Create policy**.
4. On the **Visual editor** tab, choose **Choose a service**, and then choose **Comprehend**.
5. For **Actions**, choose **Detect Sentiment** and **BatchDetectSentiment**.
6. Choose **Review policy**.
7. For **Name**, enter a name for your IAM policy. You use this name when you create an IAM role to associate with your Aurora DB cluster. You can also add an optional **Description** value.
8. Choose **Create policy**.

For the next step, see [Creating an IAM role to access SageMaker and Amazon Comprehend \(p. 1511\)](#).

Creating an IAM role to access SageMaker and Amazon Comprehend

Note

Aurora can create the IAM role for you automatically. You can skip the following information and use the procedure in [Automatically connecting an Aurora DB cluster to AWS services using the console \(p. 1496\)](#).

After you create the IAM policies, create an IAM role that the Aurora PostgreSQL DB cluster can assume for your database users to access ML services. To create an IAM role, follow the steps described in [Creating a role to delegate permissions to an IAM user](#).

Attach the preceding policies to the IAM role you create. For more information, see [Attaching an IAM policy to an IAM user or role \(p. 1748\)](#).

For more information about IAM roles, see [IAM roles in the IAM User Guide](#).

For the next step, see [Associating an IAM role with an Aurora PostgreSQL DB cluster using the AWS CLI \(p. 1511\)](#).

Associating an IAM role with an Aurora PostgreSQL DB cluster using the AWS CLI

Note

Aurora can associate an IAM role with your DB cluster for you automatically. You can skip the following information and use the procedure in [Automatically connecting an Aurora DB cluster to AWS services using the console \(p. 1496\)](#).

The last process in setting up IAM access is to associate the IAM role and its IAM policy with your Aurora PostgreSQL DB cluster. Do the following:

1. Add the role to the list of associated roles for a DB cluster.

To associate the role with your DB cluster, use the AWS Management Console or the [add-role-to-db-cluster](#) AWS CLI command.

- **To add an IAM role for a PostgreSQL DB cluster using the console**

1. Sign in to the AWS Management Console and open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. Choose the PostgreSQL DB cluster name to display its details.
3. On the **Connectivity & security** tab, in the **Manage IAM roles** section, choose the role to add under **Add IAM roles to this cluster**.
4. Under **Feature**, choose **SageMaker** or **Comprehend**.

5. Choose **Add role**.

- **To add an IAM role for a PostgreSQL DB cluster using the CLI**

Use the following command to add the role to the PostgreSQL DB cluster named `my-db-cluster`. Replace `your-role-arn` with the role ARN that you noted in a previous step. For the value of the `--feature-name` option, use `SageMaker`, `Comprehend`, or `s3Export` depending on which service you want to use.

Example

For Linux, macOS, or Unix:

```
aws rds add-role-to-db-cluster \
    --db-cluster-identifier my-db-cluster \
    --feature-name external-service \
    --role-arn your-role-arn \
    --region your-region
```

For Windows:

```
aws rds add-role-to-db-cluster ^
    --db-cluster-identifier my-db-cluster ^
    --feature-name external-service ^
    --role-arn your-role-arn ^
    --region your-region
```

2. Set the cluster-level parameter for each AWS ML service to the ARN for the associated IAM role.

Use the `electroencephalographic`, `miscomprehended`, or both parameters depending on which AWS ML services you intend to use with your Aurora cluster.

Cluster-level parameters are grouped into DB cluster parameter groups. To set the preceding cluster parameters, use an existing custom DB cluster group or create a new one. To create a new DB cluster parameter group, call the `create-db-cluster-parameter-group` command from the AWS CLI, for example:

```
aws rds create-db-cluster-parameter-group --db-cluster-parameter-group-
name AllowAWSAccessToExternalServices \
    --db-parameter-group-family aurora-postgresql-group --description "Allow access to
Amazon S3, Amazon SageMaker, and Amazon Comprehend"
```

Set the appropriate cluster-level parameter or parameters and the related IAM role ARN values in your DB cluster parameter group. Do the following.

```
aws rds modify-db-cluster-parameter-group \
    --db-cluster-parameter-group-name AllowAWSAccessToExternalServices \
    --parameters
    "ParameterName=aws_default_s3_role,ParameterValue=arn:aws:iam::123456789012:role/
    AllowAuroraS3Role,ApplyMethod=pending-reboot" \
    --parameters
    "ParameterName=aws_default_sagemaker_role,ParameterValue=arn:aws:iam::123456789012:role/
    AllowAuroraSageMakerRole,ApplyMethod=pending-reboot" \
    --parameters
    "ParameterName=aws_default_comprehend_role,ParameterValue=arn:aws:iam::123456789012:role/
    AllowAuroraComprehendRole,ApplyMethod=pending-reboot"
```

Modify the DB cluster to use the new DB cluster parameter group. Then, reboot the cluster. The following shows how.

```
aws rds modify-db-cluster --db-cluster-identifier your_cluster_id --db-cluster-parameter-group-nameAllowAWSAccessToExternalServices
aws rds failover-db-cluster --db-cluster-identifier your_cluster_id
```

When the instance has rebooted, your IAM roles are associated with your DB cluster.

Fast recovery after failover with cluster cache management for Aurora PostgreSQL

For fast recovery of the writer DB instance in your Aurora PostgreSQL clusters if there's a failover, use cluster cache management for Amazon Aurora PostgreSQL. Cluster cache management ensures that application performance is maintained if there's a failover.

In a typical failover situation, you might see a temporary but large performance degradation after failover. This degradation occurs because when the failover DB instance starts, the buffer cache is empty. An empty cache is also known as a *cold cache*. A cold cache degrades performance because the DB instance has to read from the slower disk, instead of taking advantage of values stored in the buffer cache.

With cluster cache management, you set a specific reader DB instance as the failover target. Cluster cache management ensures that the data in the designated reader's cache is kept synchronized with the data in the writer DB instance's cache. The designated reader's cache with prefilled values is known as a *warm cache*. If a failover occurs, the designated reader uses values in its warm cache immediately when it's promoted to the new writer DB instance. This approach provides your application much better recovery performance.

Cluster cache management requires that the designated reader instance have the same instance class type and size (db.r5.2xlarge or db.r5.xlarge, for example) as the writer. Keep this in mind when you create your Aurora PostgreSQL DB clusters so that your cluster can recover during a failover. For a listing of instance class types and sizes, see [Hardware specifications for DB instance classes for Aurora](#).

Note

Cluster cache management is not supported for Aurora PostgreSQL DB clusters that are part of Aurora global databases.

Contents

- [Configuring cluster cache management \(p. 1513\)](#)
 - [Enabling cluster cache management \(p. 1514\)](#)
 - [Setting the promotion tier priority for the writer DB instance \(p. 1514\)](#)
 - [Setting the promotion tier priority for a reader DB instance \(p. 1515\)](#)
- [Monitoring the buffer cache \(p. 1516\)](#)

Configuring cluster cache management

To configure cluster cache management, do the following processes in order.

Topics

- [Enabling cluster cache management \(p. 1514\)](#)
- [Setting the promotion tier priority for the writer DB instance \(p. 1514\)](#)

- [Setting the promotion tier priority for a reader DB instance \(p. 1515\)](#)

Note

Allow at least 1 minute after completing these steps for cluster cache management to be fully operational.

Enabling cluster cache management

To enable cluster cache management, take the steps described following.

Console

To enable cluster cache management

1. Sign in to the AWS Management Console and open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. In the navigation pane, choose **Parameter groups**.
3. In the list, choose the parameter group for your Aurora PostgreSQL DB cluster.

The DB cluster must use a parameter group other than the default, because you can't change values in a default parameter group.

4. For **Parameter group actions**, choose **Edit**.
5. Set the value of the `apg_ccm_enabled` cluster parameter to **1**.
6. Choose **Save changes**.

AWS CLI

To enable cluster cache management for an Aurora PostgreSQL DB cluster, use the AWS CLI `modify-db-cluster-parameter-group` command with the following required parameters:

- `--db-cluster-parameter-group-name`
- `--parameters`

Example

For Linux, macOS, or Unix:

```
aws rds modify-db-cluster-parameter-group \
--db-cluster-parameter-group-name my-db-cluster-parameter-group \
--parameters "ParameterName=apg_ccm_enabled,ParameterValue=1,ApplyMethod=immediate"
```

For Windows:

```
aws rds modify-db-cluster-parameter-group ^
--db-cluster-parameter-group-name my-db-cluster-parameter-group ^
--parameters "ParameterName=apg_ccm_enabled,ParameterValue=1,ApplyMethod=immediate"
```

Setting the promotion tier priority for the writer DB instance

For cluster cache management, make sure that the promotion priority is **tier-0** for the writer DB instance of the Aurora PostgreSQL DB cluster. The *promotion tier priority* is a value that specifies the order in which an Aurora reader is promoted to the writer DB instance after a failure. Valid values are 0–15, where

0 is the first priority and 15 is the last priority. For more information about the promotion tier, see [Fault tolerance for an Aurora DB cluster \(p. 69\)](#).

Console

To set the promotion priority for the writer DB instance to tier-0

1. Sign in to the AWS Management Console and open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. In the navigation pane, choose **Databases**.
3. Choose the **Writer** DB instance of the Aurora PostgreSQL DB cluster.
4. Choose **Modify**. The **Modify DB Instance** page appears.
5. On the **Additional configuration** panel, choose **tier-0** for the **Failover priority**.
6. Choose **Continue** and check the summary of modifications.
7. To apply the changes immediately after you save them, choose **Apply immediately**.
8. Choose **Modify DB Instance** to save your changes.

AWS CLI

To set the promotion tier priority to 0 for the writer DB instance using the AWS CLI, call the [modify-db-instance](#) command with the following required parameters:

- `--db-instance-identifier`
- `--promotion-tier`
- `--apply-immediately`

Example

For Linux, macOS, or Unix:

```
aws rds modify-db-instance \
  --db-instance-identifier writer-db-instance \
  --promotion-tier 0 \
  --apply-immediately
```

For Windows:

```
aws rds modify-db-instance ^
  --db-instance-identifier writer-db-instance ^
  --promotion-tier 0 ^
  --apply-immediately
```

Setting the promotion tier priority for a reader DB instance

You set one reader DB instance for cluster cache management. To do so, choose a reader from the Aurora PostgreSQL cluster that is the same instance class and size as the writer DB instance. For example, if the writer uses `db.r5.xlarge`, choose a reader that uses this same instance class type and size. Then set its promotion tier priority to 0.

The *promotion tier priority* is a value that specifies the order in which an Aurora reader is promoted to the writer DB instance after a failure. Valid values are 0–15, where 0 is the first priority and 15 is the last priority.

Console

To set the promotion priority of the reader DB instance to tier-0

1. Sign in to the AWS Management Console and open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. In the navigation pane, choose **Databases**.
3. Choose a **Reader** DB instance of the Aurora PostgreSQL DB cluster that is the same instance class as the writer DB instance.
4. Choose **Modify**. The **Modify DB Instance** page appears.
5. On the **Additional configuration** panel, choose **tier-0** for the **Failover priority**.
6. Choose **Continue** and check the summary of modifications.
7. To apply the changes immediately after you save them, choose **Apply immediately**.
8. Choose **Modify DB Instance** to save your changes.

AWS CLI

To set the promotion tier priority to 0 for the reader DB instance using the AWS CLI, call the `modify-db-instance` command with the following required parameters:

- `--db-instance-identifier`
- `--promotion-tier`
- `--apply-immediately`

Example

For Linux, macOS, or Unix:

```
aws rds modify-db-instance \
  --db-instance-identifier reader-db-instance \
  --promotion-tier 0 \
  --apply-immediately
```

For Windows:

```
aws rds modify-db-instance ^
  --db-instance-identifier reader-db-instance ^
  --promotion-tier 0 ^
  --apply-immediately
```

Monitoring the buffer cache

After setting up cluster cache management, you can monitor the state of synchronization between the writer DB instance's buffer cache and the designated reader's warm buffer cache. To examine the buffer cache contents on both the writer DB instance and the designated reader DB instance, use the PostgreSQL `pg_buffercache` module. For more information, see the [PostgreSQL pg_buffercache documentation](#).

Using the `aurora_ccm_status` Function

Cluster cache management also provides the `aurora_ccm_status` function. Use the `aurora_ccm_status` function on the writer DB instance to get the following information about the progress of cache warming on the designated reader:

- `buffers_sent_last_minute` – How many buffers have been sent to the designated reader in the last minute.
- `buffers_sent_last_scan` – How many buffers have been sent to the designated reader during the last complete scan of the buffer cache.
- `buffers_found_last_scan` – How many buffers have been identified as frequently accessed and needed to be sent during the last complete scan of the buffer cache. Buffers already cached on the designated reader aren't sent.
- `buffers_sent_current_scan` – How many buffers have been sent so far during the current scan.
- `buffers_found_current_scan` – How many buffers have been identified as frequently accessed in the current scan.
- `current_scan_progress` – How many buffers have been visited so far during the current scan.

The following example shows how to use the `aurora_ccm_status` function to convert some of its output into a warm rate and warm percentage.

```
SELECT buffers_sent_last_minute*8/60 AS warm_rate_kbps,
       100*(1.0-buffers_sent_last_scan/buffers_found_last_scan) AS warm_percent
  FROM aurora_ccm_status();
```

Invoking an AWS Lambda function from an Aurora PostgreSQL DB cluster

AWS Lambda is an event-driven compute service that lets you run code without provisioning or managing servers. It's available for use with many AWS services, including Aurora PostgreSQL. For example, you can use Lambda functions to process event notifications from a database, or to load data from files whenever a new file is uploaded to Amazon S3. To learn more about Lambda, see [What is AWS Lambda?](#) in the *AWS Lambda Developer Guide*.

Note

Invoking AWS Lambda functions is supported in Aurora PostgreSQL 11.9 and higher.

Setting up Aurora PostgreSQL to work with Lambda functions is a multi-step process involving AWS Lambda, IAM, your VPC, and your Aurora PostgreSQL DB cluster. Following, you can find summaries of the necessary steps.

For more information about Lambda functions, see [Getting started with Lambda](#) and [AWS Lambda foundations](#) in the *AWS Lambda Developer Guide*.

Topics

- [Step 1: Configure your Aurora PostgreSQL DB cluster for outbound connections to AWS Lambda \(p. 1518\)](#)
- [Step 2: Configure IAM for your Aurora PostgreSQL DB cluster and AWS Lambda \(p. 1518\)](#)
- [Step 3: Install the `aws_lambda` extension for an Aurora PostgreSQL DB cluster \(p. 1519\)](#)
- [Step 4: Use Lambda helper functions with your Aurora PostgreSQL DB cluster \(Optional\) \(p. 1520\)](#)
- [Step 5: Invoke a Lambda function from your Aurora PostgreSQL DB cluster \(p. 1521\)](#)
- [Lambda function error messages \(p. 1524\)](#)
- [Function reference \(p. 1524\)](#)

Step 1: Configure your Aurora PostgreSQL DB cluster for outbound connections to AWS Lambda

Lambda functions always run inside an Amazon VPC owned by the AWS Lambda service. Lambda applies network access and security rules to this VPC and it maintains and monitors the VPC automatically.

Your Aurora PostgreSQL DB cluster needs to send network traffic to the Lambda service's VPC. How you configure this depends on whether your Aurora DB cluster's primary DB instance is public or private.

- If your Aurora PostgreSQL DB cluster is public, you need only configure your security group to allow outbound traffic from your VPC. Your DB cluster's primary DB instance is public if it's located in a public subnet on your VPC, and if the instance's "PubliclyAccessible" property is `true`.

To find the value of this property, you can use the [describe-db-instances](#) AWS CLI command. Or, you can use the AWS Management Console to open the **Connectivity & security** tab and check that **Publicly accessible** is **Yes**. You can also use the AWS Management Console and the AWS CLI to check that the instance is in a public subnet in your VPC.

- If your Aurora PostgreSQL DB cluster is private, you have a couple of choices. You can use a Network Address Translation (NAT) gateway or you can use a VPC endpoint. For information about NAT gateways, see [NAT gateways](#). The summary of steps for using a VPC endpoint follow.

To configure connectivity to AWS Lambda for a public DB instance

- Configure the VPC in which your Aurora PostgreSQL DB cluster is running to allow outbound connections. You do so by creating an outbound rule on your VPC's security group that allows TCP traffic to port 443 and to any IPv4 addresses (0.0.0.0/0).

To configure connectivity to AWS Lambda for a private DB instance

1. Configure your VPC with a VPC endpoint for AWS Lambda. For details, see [VPC endpoints](#) in the *Amazon VPC User Guide*. The endpoint returns responses to calls made by your Aurora PostgreSQL DB cluster to your Lambda functions.
2. Add the endpoint to your VPC's route table. For more information, see [Work with route tables](#) in the *Amazon VPC User Guide*.

Your VPC can now interact with the AWS Lambda VPC at the network level. However, you still need to configure the permissions using IAM.

Step 2: Configure IAM for your Aurora PostgreSQL DB cluster and AWS Lambda

Invoking Lambda functions from your Aurora PostgreSQL DB cluster requires certain privileges. To configure the necessary privileges, we recommend that you create an IAM policy that allows invoking Lambda functions, assign that policy to a role, and then apply the role to your DB cluster. This approach gives the DB cluster privileges to invoke the specified Lambda function on your behalf. The following steps show you how to do this using the AWS CLI.

To configure IAM permissions for using your cluster with Lambda

1. Use the [create-policy](#) AWS CLI command to create an IAM policy that allows your Aurora PostgreSQL DB cluster to invoke the specified Lambda function. (The statement ID (Sid) is an optional description for your policy statement and has no effect on usage.) This policy gives your Aurora DB cluster the minimum permissions needed to invoke the specified Lambda function.

```
aws iam create-policy --policy-name rds-lambda-policy --policy-document '{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "AllowAccessToExampleFunction",  
            "Effect": "Allow",  
            "Action": "lambda:InvokeFunction",  
            "Resource": "arn:aws:lambda:aws-region:444455556666:function:my-function"  
        }  
    ]  
}'
```

Alternatively, you can use the predefined AWSLambdaRole policy that allows you to invoke any of your Lambda functions. For more information, see [Identity-based IAM policies for Lambda](#)

2. Use the [create-role](#) AWS CLI command to create an IAM role that the policy can assume at runtime.

```
aws iam create-role --role-name rds-lambda-role --assume-role-policy-document '{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Principal": {  
                "Service": "rds.amazonaws.com"  
            },  
            "Action": "sts:AssumeRole"  
        }  
    ]  
}'
```

3. Apply the policy to the role by using the [attach-role-policy](#) AWS CLI command.

```
aws iam attach-role-policy \  
    --policy-arn arn:aws:iam::444455556666:policy/rds-lambda-policy \  
    --role-name rds-lambda-role --region aws-region
```

4. Apply the role to your Aurora PostgreSQL DB cluster by using the [add-role-to-db-cluster](#) AWS CLI command. This last step allows your DB cluster's database users to invoke Lambda functions.

```
aws rds add-role-to-db-cluster \  
    --db-cluster-identifier my-cluster-name \  
    --feature-name Lambda \  
    --role-arn arn:aws:iam::444455556666:role/rds-lambda-role \  
    --region aws-region
```

With the VPC and the IAM configurations complete, you can now install the `aws_lambda` extension. (Note that you can install the extension at any time, but until you set up the correct VPC support and IAM privileges, the `aws_lambda` extension adds nothing to your Aurora PostgreSQL DB cluster's capabilities.)

Step 3: Install the `aws_lambda` extension for an Aurora PostgreSQL DB cluster

To use AWS Lambda with your Aurora PostgreSQL DB cluster, the `aws_lambda` PostgreSQL extension to your Aurora PostgreSQL. This extension provides your Aurora PostgreSQL DB cluster with the ability to call Lambda functions from PostgreSQL.

To install the aws_lambda extension in your Aurora PostgreSQL DB cluster

Use the PostgreSQL `psql` command-line or the pgAdmin tool to connect to your Aurora PostgreSQL DB cluster .

1. Connect to your Aurora PostgreSQL DB cluster instance as a user with `rds_superuser` privileges. The default `postgres` user is shown in the example.

```
psql -h cluster-instance.44445556666.aws-region.rds.amazonaws.com -U postgres -p 5432
```

2. Install the `aws_lambda` extension. The `aws_commons` extension is also required. It provides helper functions to `aws_lambda` and many other Aurora extensions for PostgreSQL. If it's not already on your Aurora PostgreSQLDB cluster , it's installed with `aws_lambda` as shown following.

```
CREATE EXTENSION IF NOT EXISTS aws_lambda CASCADE;
NOTICE:  installing required extension "aws_commons"
CREATE EXTENSION
```

The `aws_lambda` extension is installed in your Aurora PostgreSQL DB cluster's primary DB instance. You can now create convenience structures for invoking your Lambda functions.

Step 4: Use Lambda helper functions with your Aurora PostgreSQL DB cluster (Optional)

You can use the helper functions in the `aws_commons` extension to prepare entities that you can more easily invoke from PostgreSQL. To do this, you need to have the following information about your Lambda functions:

- **Function name** – The name, Amazon Resource Name (ARN), version, or alias of the Lambda function. The IAM policy created in [Step 2: Configure IAM for your cluster and Lambda \(p. 1518\)](#) requires the ARN, so we recommend that you use your function's ARN.
- **AWS Region** – (Optional) The AWS Region where the Lambda function is located if it's not in the same Region as your Aurora PostgreSQL DB cluster.

To hold the Lambda function name information, you use the [aws_commons.create_lambda_function_arn \(p. 1526\)](#) function. This helper function creates an `aws_commons._lambda_function_arn_1` composite structure with the details needed by the `invoke` function. Following, you can find three alternative approaches to setting up this composite structure.

```
SELECT aws_commons.create_lambda_function_arn(
    'my-function',
    'aws-region'
) AS aws_lambda_arn_1 \gset
```

```
SELECT aws_commons.create_lambda_function_arn(
    '11112223333:function:my-function',
    'aws-region'
) AS lambda_partial_arn_1 \gset
```

```
SELECT aws_commons.create_lambda_function_arn(
    'arn:aws:lambda:aws-region:11112223333:function:my-function'
) AS lambda_arn_1 \gset
```

Any of these values can be used in calls to the [aws_lambda.invoke \(p. 1524\)](#) function. For examples, see [Step 5: Invoke a Lambda function from your Aurora PostgreSQL DB cluster \(p. 1521\)](#).

Step 5: Invoke a Lambda function from your Aurora PostgreSQL DB cluster

The `aws_lambda.invoke` function behaves synchronously or asynchronously, depending on the `invocation_type`. The two alternatives for this parameter are `RequestResponse` (the default) and `Event`, as follows.

- **RequestResponse** – This invocation type is *synchronous*. It's the default behavior when the call is made without specifying an invocation type. The response payload includes the results of the `aws_lambda.invoke` function. Use this invocation type when your workflow requires receiving results from the Lambda function before proceeding.
- **Event** – This invocation type is *asynchronous*. The response doesn't include a payload containing results. Use this invocation type when your workflow doesn't need a result from the Lambda function to continue processing.

As a simple test of your setup, you can connect to your DB instance using `psql` and invoke an example function from the command line. Suppose that you have one of the basic functions set up on your Lambda service, such as the simple Python function shown in the following screenshot.

The screenshot shows the AWS Lambda console interface. The top navigation bar has tabs for Code, Test, Monitor, Configuration, Aliases, and Versions. The Code tab is selected. Below the tabs is a toolbar with File, Edit, Find, View, Go, Tools, Window, Test, Deploy, and Changes deployed. The main area is titled "Code source" with an Info link. On the left, there's a sidebar for Environment with a "simple -/" folder containing a "lambda_function.py" file. The code editor window shows a Python script:

```
1 import json
2 def lambda_handler(event, context):
3     #TODO implement
4     return {
5         'statusCode': 200,
6         'body': json.dumps('Hello from Lambda!')
7     }
8 |
```

To invoke an example function

1. Connect to your primary DB instance using `psql` or pgAdmin.

```
psql -h cluster.444455556666.aws-region.rds.amazonaws.com -U postgres -p 5432
```

2. Invoke the function using its ARN.

```
SELECT * from
aws_lambda.invoke(aws_commons.create_lambda_function_arn('arn:aws:lambda:aws-
region:444455556666:function:simple', 'us-west-1'), '{"body": "Hello from
Postgres!"}':json );
```

The response looks as follows.

status_code log_result	payload	executed_version
--------------------------	---------	------------------

```
-----+-----  
+-----+-----  
| 200 | {"statusCode": 200, "body": "\"Hello from Lambda!\""} | $LATEST  
|  
(1 row)
```

If your invocation attempt doesn't succeed, see [Lambda function error messages \(p. 1524\)](#).

Following, you can find several examples of calling the [aws_lambda.invoke \(p. 1524\)](#) function. Most all the examples use the composite structure `aws_lambda_arn_1` that you create in [Step 4: Use Lambda helper functions with your Aurora PostgreSQL DB cluster \(Optional\) \(p. 1520\)](#) to simplify passing the function details. For an example of asynchronous invocation, see [Example: Asynchronous \(Event\) invocation of Lambda functions \(p. 1523\)](#). All the other examples listed use synchronous invocation.

To learn more about Lambda invocation types, see [Invoking Lambda functions](#) in the [AWS Lambda Developer Guide](#). For more information about `aws_lambda_arn_1`, see [aws_commons.create_lambda_function_arn \(p. 1526\)](#).

Examples list

- [Example: Synchronous \(RequestResponse\) invocation of Lambda functions \(p. 1522\)](#)
- [Example: Asynchronous \(Event\) invocation of Lambda functions \(p. 1523\)](#)
- [Example: Capturing the Lambda execution log in a function response \(p. 1523\)](#)
- [Example: Including client context in a Lambda function \(p. 1523\)](#)
- [Example: Invoking a specific version of a Lambda function \(p. 1523\)](#)

Example: Synchronous (RequestResponse) invocation of Lambda functions

Following are two examples of a synchronous Lambda function invocation. The results of these `aws_lambda.invoke` function calls are the same.

```
SELECT * FROM aws_lambda.invoke(:'aws_lambda_arn_1', '{"body": "Hello from Postgres!"'}::json);
```

```
SELECT * FROM aws_lambda.invoke(:'aws_lambda_arn_1', '{"body": "Hello from Postgres!"'}::json, 'RequestResponse');
```

The parameters are described as follows:

- `: 'aws_lambda_arn_1'` – This parameter identifies the composite structure created in [Step 4: Use Lambda helper functions with your Aurora PostgreSQL DB cluster \(Optional\) \(p. 1520\)](#), with the `aws_commons.create_lambda_function_arn` helper function. You can also create this structure inline within your `aws_lambda.invoke` call as follows.

```
SELECT * FROM aws_lambda.invoke(aws_commons.create_lambda_function_arn('my-function',  
    'aws-region'),  
    '{"body": "Hello from Postgres!"'}::json  
);
```

- `'{"body": "Hello from PostgreSQL!"'}::json` – The JSON payload to pass to the Lambda function.
- `'RequestResponse'` – The Lambda invocation type.

Example: Asynchronous (Event) invocation of Lambda functions

Following is an example of an asynchronous Lambda function invocation. The `Event` invocation type schedules the Lambda function invocation with the specified input payload and returns immediately. Use the `Event` invocation type in certain workflows that don't depend on the results of the Lambda function.

```
SELECT * FROM aws_lambda.invoke(:'aws_lambda_arn_1', '{"body": "Hello from Postgres!"'}::json, 'Event');
```

Example: Capturing the Lambda execution log in a function response

You can include the last 4 KB of the execution log in the function response by using the `log_type` parameter in your `aws_lambda.invoke` function call. By default, this parameter is set to `None`, but you can specify `Tail` to capture the results of the Lambda execution log in the response, as shown following.

```
SELECT *, select convert_from(decode(log_result, 'base64'), 'utf-8') as log FROM aws_lambda.invoke(:'aws_lambda_arn_1', '{"body": "Hello from Postgres!"'}::json, 'RequestResponse', 'Tail');
```

Set the [aws_lambda.invoke \(p. 1524\)](#) function's `log_type` parameter to `Tail` to include the execution log in the response. The default value for the `log_type` parameter is `None`.

The `log_result` that's returned is a base64 encoded string. You can decode the contents using a combination of the `decode` and `convert_from` PostgreSQL functions.

For more information about `log_type`, see [aws_lambda.invoke \(p. 1524\)](#).

Example: Including client context in a Lambda function

The `aws_lambda.invoke` function has a `context` parameter that you can use to pass information separate from the payload, as shown following.

```
SELECT *, convert_from(decode(log_result, 'base64'), 'utf-8') as log FROM aws_lambda.invoke(:'aws_lambda_arn_1', '{"body": "Hello from Postgres!"'}::json, 'RequestResponse', 'Tail');
```

To include client context, use a JSON object for the [aws_lambda.invoke \(p. 1524\)](#) function's `context` parameter.

For more information about the `context` parameter, see the [aws_lambda.invoke \(p. 1524\)](#) reference.

Example: Invoking a specific version of a Lambda function

You can specify a particular version of a Lambda function by including the `qualifier` parameter with the `aws_lambda.invoke` call. Following, you can find an example that does this using '`custom_version`' as an alias for the version.

```
SELECT * FROM aws_lambda.invoke(:'aws_lambda_arn_1', '{"body": "Hello from Postgres!"'}::json, 'RequestResponse', 'None', NULL, 'custom_version');
```

You can also supply a Lambda function qualifier with the function name details instead, as follows.

```
SELECT * FROM aws_lambda.invoke(aws_commons.create_lambda_function_arn('my-
function:custom_version', 'us-west-2'),
'{"body": "Hello from Postgres!"}':json);
```

For more information about `qualifier` and other parameters, see the [aws_lambda.invoke \(p. 1524\)](#) reference.

Lambda function error messages

Incorrect VPC configuration can result in error messages, such as the following.

```
ERROR: invoke API failed
DETAIL: AWS Lambda client returned 'Unable to connect to endpoint'.
CONTEXT: SQL function "invoke" statement 1
```

The first thing to check is your VPC security group. Make sure you have an outbound rule for TCP open on port 443 so that your VPC can connect to the Lambda VPC.

If a Lambda function throws an exception during request processing, `aws_lambda.invoke` fails with a PostgreSQL error such as the following.

```
SELECT * FROM aws_lambda.invoke(:'aws_lambda_arn_1', '{"body": "Hello from
Postgres!"}':json);
ERROR: lambda invocation failed
DETAIL: "arn:aws:lambda:us-west-2:555555555555:function:my-function" returned error
"Unhandled", details: "<Error details string>".
```

Be sure to handle errors in your Lambda functions or in your PostgreSQL application.

Function reference

Following is the reference for the functions to use for invoking Lambda functions with Aurora PostgreSQL .

Functions

- [aws_lambda.invoke \(p. 1524\)](#)
- [aws_commons.create_lambda_function_arn \(p. 1526\)](#)

aws_lambda.invoke

Runs a Lambda function for an Aurora PostgreSQL DB cluster .

For more details about invoking Lambda functions, see also [Invoke in the AWS Lambda Developer Guide](#).

Syntax

JSON

```
aws_lambda.invoke(
IN function_name TEXT,
IN payload JSON,
IN region TEXT DEFAULT NULL,
IN invocation_type TEXT DEFAULT 'RequestResponse',
```

```
IN log_type TEXT DEFAULT 'None',
IN context JSON DEFAULT NULL,
IN qualifier VARCHAR(128) DEFAULT NULL,
OUT status_code INT,
OUT payload JSON,
OUT executed_version TEXT,
OUT log_result TEXT)
```

```
aws_lambda.invoke(
IN function_name aws_commons._lambda_function_arn_1,
IN payload JSON,
IN invocation_type TEXT DEFAULT 'RequestResponse',
IN log_type TEXT DEFAULT 'None',
IN context JSON DEFAULT NULL,
IN qualifier VARCHAR(128) DEFAULT NULL,
OUT status_code INT,
OUT payload JSON,
OUT executed_version TEXT,
OUT log_result TEXT)
```

JSONB

```
aws_lambda.invoke(
IN function_name TEXT,
IN payload JSONB,
IN region TEXT DEFAULT NULL,
IN invocation_type TEXT DEFAULT 'RequestResponse',
IN log_type TEXT DEFAULT 'None',
IN context JSONB DEFAULT NULL,
IN qualifier VARCHAR(128) DEFAULT NULL,
OUT status_code INT,
OUT payload JSONB,
OUT executed_version TEXT,
OUT log_result TEXT)
```

```
aws_lambda.invoke(
IN function_name aws_commons._lambda_function_arn_1,
IN payload JSONB,
IN invocation_type TEXT DEFAULT 'RequestResponse',
IN log_type TEXT DEFAULT 'None',
IN context JSONB DEFAULT NULL,
IN qualifier VARCHAR(128) DEFAULT NULL,
OUT status_code INT,
OUT payload JSONB,
OUT executed_version TEXT,
OUT log_result TEXT
)
```

Input parameters

function_name

The identifying name of the Lambda function. The value can be the function name, an ARN, or a partial ARN. For a listing of possible formats, see [Lambda function name formats](#) in the *AWS Lambda Developer Guide*.

payload

The input for the Lambda function. The format can be JSON or JSONB. For more information, see [JSON Types](#) in the PostgreSQL documentation.

region

(Optional) The Lambda Region for the function. By default, Aurora resolves the AWS Region from the full ARN in the `function_name` or it uses the Aurora PostgreSQL DB instance Region. If this Region value conflicts with the one provided in the `function_name` ARN, an error is raised.

invocation_type

The invocation type of the Lambda function. The value is case-sensitive. Possible values include the following:

- `RequestResponse` – The default. This type of invocation for a Lambda function is synchronous and returns a response payload in the result. Use the `RequestResponse` invocation type when your workflow depends on receiving the Lambda function result immediately.
- `Event` – This type of invocation for a Lambda function is asynchronous and returns immediately without a returned payload. Use the `Event` invocation type when you don't need results of the Lambda function before your workflow moves on.
- `DryRun` – This type of invocation tests access without running the Lambda function.

log_type

The type of Lambda log to return in the `log_result` output parameter. The value is case-sensitive. Possible values include the following:

- `Tail` – The returned `log_result` output parameter will include the last 4 KB of the execution log.
- `None` – No Lambda log information is returned.

context

Client context in JSON or JSONB format. Fields to use include `custom` and `env`.

qualifier

A qualifier that identifies a Lambda function's version to be invoked. If this value conflicts with one provided in the `function_name` ARN, an error is raised.

Output parameters

status_code

An HTTP status response code. For more information, see [Lambda Invoke response elements](#) in the *AWS Lambda Developer Guide*.

payload

The information returned from the Lambda function that ran. The format is in JSON or JSONB.

executed_version

The version of the Lambda function that ran.

log_result

The execution log information returned if the `log_type` value is `Tail` when the Lambda function was invoked. The result contains the last 4 KB of the execution log encoded in Base64.

[aws_commons.create_lambda_function_arn](#)

Creates an `aws_commons._lambda_function_arn_1` structure to hold Lambda function name information. You can use the results of the `aws_commons.create_lambda_function_arn` function in the `function_name` parameter of the `aws_lambda.invoke` ([p. 1524](#)) function.

Syntax

```
aws_commons.create_lambda_function_arn(  
    function_name TEXT,  
    region TEXT DEFAULT NULL  
)  
RETURNS aws_commons._lambda_function_arn_1
```

Input parameters

function_name

A required text string containing the Lambda function name. The value can be a function name, a partial ARN, or a full ARN.

region

An optional text string containing the AWS Region that the Lambda function is in. For a listing of Region names and associated values, see [Regions and Availability Zones \(p. 11\)](#).

Using the oracle_fdw extension to access foreign data in Aurora PostgreSQL

For easy and efficient access to Oracle databases for Aurora PostgreSQL, you can use the PostgreSQL oracle_fdw extension, which provides a foreign data wrapper. For a complete description of this extension, see the [oracle_fdw](#) documentation.

The oracle_fdw extension is supported on Amazon RDS for PostgreSQL versions 12.7, 13.3, and higher.

Turning on the oracle_fdw extension

To use the oracle_fdw extension, perform the following procedure.

To turn on the oracle_fdw extension

- Run the following command using an account that has `rds_superuser` permissions.

```
CREATE EXTENSION oracle_fdw;
```

Example using a foreign server linked to an RDS for Oracle database

The following example demonstrates using a foreign server linked to an RDS for Oracle database.

To create a foreign server linked to an RDS for Oracle database

- Note the following for the RDS for Oracle DB instance:

- Endpoint
- Port
- Database name

2. Create a foreign server.

```
test=> CREATE SERVER oradb FOREIGN DATA WRAPPER oracle_fdw OPTIONS (dbserver
'//endpoint:port/DB_name');
CREATE SERVER
```

3. Grant usage to a user who doesn't have `rds_superuser` permissions, for example `user1`.

```
test=> GRANT USAGE ON FOREIGN SERVER oradb TO user1;
GRANT
```

4. Connect as `user1` and create a mapping to an Oracle user.

```
test=> CREATE USER MAPPING FOR user1 SERVER oradb OPTIONS (user 'oracleuser', password
'mypassword');
CREATE USER MAPPING
```

5. Create a foreign table linked to an Oracle table.

```
test=> create foreign table mytab (a int) SERVER oradb OPTIONS (table 'MYTABLE');
CREATE FOREIGN TABLE
```

6. Query the foreign table.

```
test=> select * from mytab;
a
---
1
(1 row)
```

If the query reports the following error, check your security group and access control list (ACL) to make sure that both instances can communicate.

```
ERROR: connection for foreign table "mytab" cannot be established
DETAIL: ORA-12170: TNS:Connect timeout occurred
```

Working with encryption in transit

PostgreSQL-to-Oracle encryption in transit is based on a combination of client and server configuration parameters. For an example using Oracle 21c, see [About the Values for Negotiating Encryption and Integrity](#) in the Oracle documentation. The client used for `oracle_fdw` on RDS is configured with `ACCEPTED`, meaning that the encryption depends on the Oracle database server configuration.

pg_user_mapping and pg_user_mappings permissions

In the following table, you can find an illustration of user mapping permissions using the example roles. The `rdssu1` and `rdssu2` users have the `rds_superuser` role, and the `user1` user doesn't.

Note

You can use the `\du` metacommand in `psql` to list existing roles.

```
test=> \du
```

List of roles

Role name	Attributes
Member of	
rdssu1	
{rds_superuser}	
rdssu2	
{rds_superuser}	
user1	{}

Users with the `rds_superuser` role can't query the `pg_user_mapping` table. The following example uses `rdssu1`,

```
test=> SET SESSION AUTHORIZATION rdssu1;
SET
test=> select * from pg_user_mapping;
ERROR: permission denied for table pg_user_mapping
```

On RDS for PostgreSQL, all users—even ones with the `rds_superuser` role—can see only their own `umoptions` values in the `pg_user_mappings` table. The following example demonstrates.

```
test=> SET SESSION AUTHORIZATION rdssu1;
SET
test=> select * from pg_user_mappings;
 umid | srvid | srvname | umuser | username | umoptions
-----+-----+-----+-----+-----+
 16414 | 16411 | oradb | 16412 | user1 |
 16423 | 16411 | oradb | 16421 | rdssu1 | {user=oracleuser,password=mypwd}
 16424 | 16411 | oradb | 16422 | rdssu2 |
(3 rows)

test=> SET SESSION AUTHORIZATION rdssu2;
SET
test=> select * from pg_user_mappings;
 umid | srvid | srvname | umuser | username | umoptions
-----+-----+-----+-----+-----+
 16414 | 16411 | oradb | 16412 | user1 |
 16423 | 16411 | oradb | 16421 | rdssu1 |
 16424 | 16411 | oradb | 16422 | rdssu2 | {user=oracleuser,password=mypwd}
(3 rows)

test=> SET SESSION AUTHORIZATION user1;
SET
test=> select * from pg_user_mappings;
 umid | srvid | srvname | umuser | username | umoptions
-----+-----+-----+-----+-----+
 16414 | 16411 | oradb | 16412 | user1 | {user=oracleuser,password=mypwd}
 16423 | 16411 | oradb | 16421 | rdssu1 |
 16424 | 16411 | oradb | 16422 | rdssu2 |
(3 rows)
```

Because of differences in implementation of `information_schema._pg_user_mappings` and `pg_catalog.pg_user_mappings`, a manually created `rds_superuser` requires additional permissions to view passwords in `pg_catalog.pg_user_mappings`.

No additional permissions are required for a user with the `rds_superuser` role to view passwords in `information_schema._pg_user_mappings`.

Users who don't have the `rds_superuser` role can view passwords in `pg_user_mappings` only under the following conditions:

- The current user is the user being mapped and owns the server or holds the `USAGE` privilege on it.

- The current user is the server owner and the mapping is for PUBLIC.

Managing PostgreSQL partitions with the pg_partman extension

PostgreSQL table partitioning provides a framework for high-performance handling of data input and reporting. Use partitioning for databases that require very fast input of large amounts of data. Partitioning also provides for faster queries of large tables. Partitioning helps maintain data without impacting the database instance because it requires less I/O resources.

By using partitioning, you can split data into custom-sized chunks for processing. For example, you can partition time-series data for ranges such as hourly, daily, weekly, monthly, quarterly, yearly, custom, or any combination of these. For a time-series data example, if you partition the table by hour, each partition contains one hour of data. If you partition the time-series table by day, the partitions holds one day's worth of data, and so on. The partition key controls the size of a partition.

When you use an `INSERT` or `UPDATE` SQL command on a partitioned table, the database engine routes the data to the appropriate partition. PostgreSQL table partitions that store the data are child tables of the main table.

During database query reads, the PostgreSQL optimizer examines the `WHERE` clause of the query and, if possible, directs the database scan to only the relevant partitions.

Starting with version 10, PostgreSQL uses declarative partitioning to implement table partitioning. This is also known as native PostgreSQL partitioning. Before PostgreSQL version 10, you used triggers to implement partitions.

PostgreSQL table partitioning provides the following features:

- Creation of new partitions at any time.
- Variable partition ranges.
- Detachable and reattachable partitions using data definition language (DDL) statements.

For example, detachable partitions are useful for removing historical data from the main partition but keeping historical data for analysis.

- New partitions inherit the parent database table properties, including the following:
 - Indexes
 - Primary keys, which must include the partition key column
 - Foreign keys
 - Check constraints
 - References
- Creating indexes for the full table or each specific partition.

You can't alter the schema for an individual partition. However, you can alter the parent table (such as adding a new column), which propagates to partitions.

Topics

- [Overview of the PostgreSQL pg_partman extension \(p. 1531\)](#)
- [Enabling the pg_partman extension \(p. 1531\)](#)
- [Configuring partitions using the create_parent function \(p. 1532\)](#)
- [Configuring partition maintenance using the run_maintenance_proc function \(p. 1533\)](#)

Overview of the PostgreSQL pg_partman extension

You can use the PostgreSQL `pg_partman` extension to automate the creation and maintenance of table partitions. For more general information, see [PG Partition Manager](#) in the `pg_partman` documentation.

Note

The `pg_partman` extension is supported on Aurora PostgreSQL versions 12.6 and higher.

Instead of having to manually create each partition, you configure `pg_partman` with the following settings:

- Table to be partitioned
- Partition type
- Partition key
- Partition granularity
- Partition precreation and management options

After you create a PostgreSQL partitioned table, you register it with `pg_partman` by calling the `create_parent` function. Doing this creates the necessary partitions based on the parameters you pass to the function.

The `pg_partman` extension also provides the `run_maintenance_proc` function, which you can call on a scheduled basis to automatically manage partitions. To ensure that the proper partitions are created as needed, schedule this function to run periodically (such as hourly). You can also ensure that partitions are automatically dropped.

Enabling the pg_partman extension

If you have multiple databases inside the same PostgreSQL DB instance for which you want to manage partitions, enable the `pg_partman` extension separately for each database. To enable the `pg_partman` extension for a specific database, create the partition maintenance schema and then create the `pg_partman` extension as follows.

```
CREATE SCHEMA partman;
CREATE EXTENSION pg_partman WITH SCHEMA partman;
```

Note

To create the `pg_partman` extension, make sure that you have `rds_superuser` privileges.

If you receive an error such as the following, grant the `rds_superuser` privileges to the account or use your superuser account.

```
ERROR: permission denied to create extension "pg_partman"
HINT: Must be superuser to create this extension.
```

To grant `rds_superuser` privileges, connect with your superuser account and run the following command.

```
GRANT rds_superuser TO user-or-role;
```

For the examples that show using the `pg_partman` extension, we use the following sample database table and partition. This database uses a partitioned table based on a timestamp. A schema `data_mart` contains a table named `events` with a column named `created_at`. The following settings are included in the `events` table:

- Primary keys `event_id` and `created_at`, which must have the column used to guide the partition.

- A check constraint `ck_valid_operation` to enforce values for an operation table column.
- Two foreign keys, where one (`fk_orga_membership`) points to the external table `organization` and the other (`fk_parent_event_id`) is a self-referenced foreign key.
- Two indexes, where one (`idx_org_id`) is for the foreign key and the other (`idx_event_type`) is for the event type.

The follow DDL statements create these objects, which are automatically included on each partition.

```

CREATE SCHEMA data_mart;
CREATE TABLE data_mart.organization (
    org_id BIGSERIAL,
    org_name TEXT,
    CONSTRAINT pk_organization PRIMARY KEY (org_id)
);

CREATE TABLE data_mart.events(
    event_id      BIGSERIAL,
    operation     CHAR(1),
    value         FLOAT(24),
    parent_event_id BIGINT,
    event_type    VARCHAR(25),
    org_id        BIGSERIAL,
    created_at    timestamp,
    CONSTRAINT pk_data_mart_event PRIMARY KEY (event_id, created_at),
    CONSTRAINT ck_valid_operation CHECK (operation = 'C' OR operation = 'D'),
    CONSTRAINT fk_orga_membership
        FOREIGN KEY(org_id)
        REFERENCES data_mart.organization (org_id),
    CONSTRAINT fk_parent_event_id
        FOREIGN KEY(parent_event_id, created_at)
        REFERENCES data_mart.events (event_id,created_at)
) PARTITION BY RANGE (created_at);

CREATE INDEX idx_org_id      ON data_mart.events(org_id);
CREATE INDEX idx_event_type ON data_mart.events(event_type);

```

Configuring partitions using the `create_parent` function

After you enable the `pg_partman` extension, use the `create_parent` function to configure partitions inside the partition maintenance schema. The following example uses the `events` table example created in [Enabling the pg_partman extension \(p. 1531\)](#). Call the `create_parent` function as follows.

```

SELECT partman.create_parent(
    p_parent_table => 'data_mart.events',
    p_control => 'created_at',
    p_type => 'native',
    p_interval=> 'daily',
    p_premake => 30);

```

The parameters are as follows:

- `p_parent_table` – The parent partitioned table. This table must already exist and be fully qualified, including the schema.
- `p_control` – The column on which the partitioning is to be based. The data type must be an integer or time-based.
- `p_type` – The type is either '`'native'` or '`'partman'`'. You typically use the `native` type for its performance improvements and flexibility. The `partman` type relies on inheritance.

- `p_interval` – The time interval or integer range for each partition. Example values include `daily`, `hourly`, and so on.
- `p_premake` – The number of partitions to create in advance to support new inserts.

For a complete description of the `create_parent` function, see [Creation Functions](#) in the `pg_partman` documentation.

Configuring partition maintenance using the `run_maintenance_proc` function

You can run partition maintenance operations to automatically create new partitions, detach partitions, or remove old partitions. Partition maintenance relies on the `run_maintenance_proc` function of the `pg_partman` extension and the `pg_cron` extension, which initiates an internal scheduler. The `pg_cron` scheduler automatically executes SQL statements, functions, and procedures defined in your databases.

The following example uses the `events` table example created in [Enabling the pg_partman extension \(p. 1531\)](#) to set partition maintenance operations to run automatically. As a prerequisite, add `pg_cron` to the `shared_preload_libraries` parameter in the DB instance's parameter group.

```
CREATE EXTENSION pg_cron;

UPDATE partman.part_config
SET infinite_time_partitions = true,
    retention = '3 months',
    retention_keep_table=true
WHERE parent_table = 'data_mart.events';
SELECT cron.schedule('@hourly', $$CALL partman.run_maintenance_proc()$$);
```

Following, you can find a step-by-step explanation of the preceding example:

1. Modify the parameter group associated with your DB instance and add `pg_cron` to the `shared_preload_libraries` parameter value. This change requires a DB instance restart for it to take effect. For more information, see [Modifying parameters in a DB parameter group \(p. 347\)](#).
2. Run the command `CREATE EXTENSION pg_cron;` using an account that has the `rds_superuser` permissions. Doing this enables the `pg_cron` extension. For more information, see [Scheduling maintenance with the PostgreSQL pg_cron extension \(p. 1370\)](#).
3. Run the command `UPDATE partman.part_config` to adjust the `pg_partman` settings for the `data_mart.events` table.
4. Run the command `SET ...` to configure the `data_mart.events` table, with these clauses:
 - a. `infinite_time_partitions = true`, – Configures the table to be able to automatically create new partitions without any limit.
 - b. `retention = '3 months'`, – Configures the table to have a maximum retention of three months.
 - c. `retention_keep_table=true` – Configures the table so that when the retention period is due, the table isn't deleted automatically. Instead, partitions that are older than the retention period are only detached from the parent table.
5. Run the command `SELECT cron.schedule ...` to make a `pg_cron` function call. This call defines how often the scheduler runs the `pg_partman` maintenance procedure, `partman.run_maintenance_proc`. For this example, the procedure runs every hour.

For a complete description of the `run_maintenance_proc` function, see [Maintenance Functions](#) in the `pg_partman` documentation.

Using Kerberos authentication with Aurora PostgreSQL

You can use Kerberos authentication to authenticate users when they connect to your DB cluster running PostgreSQL. In this case, your DB instance works with AWS Directory Service for Microsoft Active Directory to enable Kerberos authentication. AWS Directory Service for Microsoft Active Directory is also called AWS Managed Microsoft AD.

You create an AWS Managed Microsoft AD directory to store user credentials. You then provide to your PostgreSQL DB cluster the Active Directory's domain and other information. When users authenticate with the PostgreSQL DB cluster, authentication requests are forwarded to the AWS Managed Microsoft AD directory.

Keeping all of your credentials in the same directory can save you time and effort. You have a centralized place for storing and managing credentials for multiple DB clusters. Using a directory can also improve your overall security profile.

You can also access credentials from your own on-premises Microsoft Active Directory. To do so you create a trusting domain relationship so that the AWS Managed Microsoft AD directory trusts your on-premises Microsoft Active Directory. In this way, your users can access your PostgreSQL clusters with the same Windows single sign-on (SSO) experience as when they access workloads in your on-premises network.

A database can use Kerberos, AWS Identity and Access Management (IAM), or both Kerberos and IAM authentication. However, since Kerberos and IAM authentication provide different authentication methods, a specific user can log in to a database using only one or the other authentication method but not both. For more information about IAM authentication, see [IAM database authentication \(p. 1743\)](#).

Topics

- [Availability of Kerberos authentication \(p. 1534\)](#)
- [Overview of Kerberos authentication for PostgreSQL DB clusters \(p. 1535\)](#)
- [Setting up Kerberos authentication for PostgreSQL DB clusters \(p. 1536\)](#)
- [Managing a DB cluster in a Domain \(p. 1545\)](#)
- [Connecting to PostgreSQL with Kerberos authentication \(p. 1546\)](#)

Availability of Kerberos authentication

Kerberos authentication is supported on the following engine versions:

- All PostgreSQL 13 versions
- PostgreSQL 12.4 and higher 12 versions
- PostgreSQL 11.6 and higher 11 versions
- PostgreSQL 10.11 and higher 10 versions

For more information, see [Amazon Aurora PostgreSQL releases and engine versions \(p. 1599\)](#).

Amazon Aurora supports Kerberos authentication for PostgreSQL DB clusters in the following AWS Regions:

Region name	Region
US East (Ohio)	us-east-2

Region name	Region
US East (N. Virginia)	us-east-1
US West (N. California)	us-west-1
US West (Oregon)	us-west-2
Asia Pacific (Mumbai)	ap-south-1
Asia Pacific (Seoul)	ap-northeast-2
Asia Pacific (Singapore)	ap-southeast-1
Asia Pacific (Sydney)	ap-southeast-2
Asia Pacific (Tokyo)	ap-northeast-1
Canada (Central)	ca-central-1
China (Beijing)	cn-north-1
China (Ningxia)	cn-northwest-1
Europe (Frankfurt)	eu-central-1
Europe (Ireland)	eu-west-1
Europe (London)	eu-west-2
Europe (Paris)	eu-west-3
Europe (Stockholm)	eu-north-1
South America (São Paulo)	sa-east-1

Overview of Kerberos authentication for PostgreSQL DB clusters

To set up Kerberos authentication for a PostgreSQL DB cluster, take the following steps, described in more detail later:

1. Use AWS Managed Microsoft AD to create an AWS Managed Microsoft AD directory. You can use the AWS Management Console, the AWS CLI, or the AWS Directory Service API to create the directory. Make sure to open the relevant outbound ports on the directory security group so that the directory can communicate with the cluster.
2. Create a role that provides Amazon Aurora access to make calls to your AWS Managed Microsoft AD directory. To do so, create an AWS Identity and Access Management (IAM) role that uses the managed IAM policy `AmazonRDSDirectoryServiceAccess`.

For the IAM role to allow access, the AWS Security Token Service (AWS STS) endpoint must be activated in the correct AWS Region for your AWS account. AWS STS endpoints are active by default in all AWS Regions, and you can use them without any further actions. For more information, see [Activating and deactivating AWS STS in an AWS Region](#) in the *IAM User Guide*.

3. Create and configure users in the AWS Managed Microsoft AD directory using the Microsoft Active Directory tools. For more information about creating users in your Active Directory, see [Manage users and groups in AWS Managed Microsoft AD](#) in the *AWS Directory Service Administration Guide*.

4. If you plan to locate the directory and the DB instance in different AWS accounts or virtual private clouds (VPCs), configure VPC peering. For more information, see [What is VPC peering?](#) in the *Amazon VPC Peering Guide*.
 5. Create or modify a PostgreSQL DB cluster either from the console, CLI, or RDS API using one of the following methods:
 - [Creating a DB cluster and connecting to a database on an Aurora PostgreSQL DB cluster \(p. 96\)](#)
 - [Modifying an Amazon Aurora DB cluster \(p. 372\)](#)
 - [Restoring from a DB cluster snapshot \(p. 497\)](#)
 - [Restoring a DB cluster to a specified time \(p. 537\)](#)
- You can locate the cluster in the same Amazon Virtual Private Cloud (VPC) as the directory or in a different AWS account or VPC. When you create or modify the PostgreSQL DB cluster, do the following:
- Provide the domain identifier (`d-*` identifier) that was generated when you created your directory.
 - Provide the name of the IAM role that you created.
 - Ensure that the DB instance security group can receive inbound traffic from the directory security group.
6. Use the RDS master user credentials to connect to the PostgreSQL DB cluster. Create the user in PostgreSQL to be identified externally. Externally identified users can log in to the PostgreSQL DB cluster using Kerberos authentication.

Setting up Kerberos authentication for PostgreSQL DB clusters

You use AWS Directory Service for Microsoft Active Directory (AWS Managed Microsoft AD) to set up Kerberos authentication for a PostgreSQL DB cluster. To set up Kerberos authentication, take the following steps.

Topics

- [Step 1: Create a directory using AWS Managed Microsoft AD \(p. 1536\)](#)
- [Step 2: \(Optional\) create a trust for an on-premises Active Directory \(p. 1540\)](#)
- [Step 3: Create an IAM role for Amazon Aurora to access the AWS Directory Service \(p. 1541\)](#)
- [Step 4: Create and configure users \(p. 1542\)](#)
- [Step 5: Enable cross-VPC traffic between the directory and the DB instance \(p. 1542\)](#)
- [Step 6: Create or modify a PostgreSQL DB cluster \(p. 1543\)](#)
- [Step 7: Create Kerberos authentication PostgreSQL logins \(p. 1544\)](#)
- [Step 8: Configure a PostgreSQL client \(p. 1544\)](#)

Step 1: Create a directory using AWS Managed Microsoft AD

AWS Directory Service creates a fully managed Active Directory in the AWS Cloud. When you create an AWS Managed Microsoft AD directory, AWS Directory Service creates two domain controllers and DNS servers for you. The directory servers are created in different subnets in a VPC. This redundancy helps make sure that your directory remains accessible even if a failure occurs.

When you create an AWS Managed Microsoft AD directory, AWS Directory Service performs the following tasks on your behalf:

- Sets up an Active Directory within your VPC.

- Creates a directory administrator account with the user name **Admin** and the specified password. You use this account to manage your directory.

Important

Make sure to save this password. AWS Directory Service doesn't store this password, and it can't be retrieved or reset.

- Creates a security group for the directory controllers. The security group must permit communication with the PostgreSQL DB cluster.

When you launch AWS Directory Service for Microsoft Active Directory, AWS creates an Organizational Unit (OU) that contains all of your directory's objects. This OU, which has the NetBIOS name that you entered when you created your directory, is located in the domain root. The domain root is owned and managed by AWS.

The **Admin** account that was created with your AWS Managed Microsoft AD directory has permissions for the most common administrative activities for your OU:

- Create, update, or delete users
- Add resources to your domain such as file or print servers, and then assign permissions for those resources to users in your OU
- Create additional OUs and containers
- Delegate authority
- Restore deleted objects from the Active Directory Recycle Bin
- Run Active Directory and Domain Name Service (DNS) modules for Windows PowerShell on the Active Directory Web Service

The **Admin** account also has rights to perform the following domain-wide activities:

- Manage DNS configurations (add, remove, or update records, zones, and forwarders)
- View DNS event logs
- View security event logs

To create a directory with AWS Managed Microsoft AD

- In the [AWS Directory Service console](#) navigation pane, choose **Directories**, and then choose **Set up directory**.
- Choose **AWS Managed Microsoft AD**. AWS Managed Microsoft AD is the only option currently supported for use with Amazon Aurora.
- Choose **Next**.
- On the **Enter directory information** page, provide the following information:

Edition

Choose the edition that meets your requirements.

Directory DNS name

The fully qualified name for the directory, such as **corp.example.com**.

Directory NetBIOS name

An optional short name for the directory, such as **CORP**.

Directory description

An optional description for the directory.

Admin password

The password for the directory administrator. The directory creation process creates an administrator account with the user name `Admin` and this password.

The directory administrator password can't include the word "admin." The password is case-sensitive and must be 8–64 characters in length. It must also contain at least one character from three of the following four categories:

- Lowercase letters (a–z)
- Uppercase letters (A–Z)
- Numbers (0–9)
- Nonalphanumeric characters (~!@#\$%^&*_+=`|\{}[];"'<>,.?/)

Confirm password

Retype the administrator password.

Important

Make sure that you save this password. AWS Directory Service doesn't store this password, and it can't be retrieved or reset.

5. Choose **Next**.
6. On the **Choose VPC and subnets** page, provide the following information:

VPC

Choose the VPC for the directory. You can create the PostgreSQL DB cluster in this same VPC or in a different VPC.

Subnets

Choose the subnets for the directory servers. The two subnets must be in different Availability Zones.

7. Choose **Next**.
8. Review the directory information. If changes are needed, choose **Previous** and make the changes. When the information is correct, choose **Create directory**.

Review & create

Review

Directory type	VPC
Microsoft AD	vpc-8b6b78e9 ([REDACTED])
Directory DNS name	Subnets
corp.example.com	subnet-75128d10 ([REDACTED], us-east-1a) subnet-f51665dd ([REDACTED], us-east-1b)
Directory NetBIOS name	
CORP	
Directory description	
My directory	

Pricing

Edition	Free trial eligible Learn more 30-day limited trial
Standard	
~USD [REDACTED] *	
* Includes two domain controllers, USD [REDACTED] /mo for each additional domain controller.	

[Cancel](#) [Previous](#) [Create directory](#)

It takes several minutes for the directory to be created. When it has been successfully created, the **Status** value changes to **Active**.

To see information about your directory, choose the directory ID in the directory listing. Make a note of the **Directory ID** value. You need this value when you create or modify your PostgreSQL DB instance.

The screenshot shows the 'Directory details' page for a Microsoft AD directory. The 'Directory ID' field, which contains the value 'd-90670a8d36', is circled in red. Other visible fields include 'Directory type' (Microsoft AD), 'Edition' (Standard), 'Status' (Active), 'Subnets' (subnet-7d36a227, subnet-a2ab49c6), 'Last updated' (Tuesday, January 7, 2020), 'Launch time' (Tuesday, January 7, 2020), 'Availability zones' (us-east-1c, us-east-1d), 'DNS address' (redacted), 'Directory DNS name' (corp.example.com), 'Directory NetBIOS name' (CORP), and a 'Description' field containing 'My directory'. Below the table, there are tabs for 'Application management' (which is selected), 'Scale & share', 'Networking & security', and 'Maintenance'.

Step 2: (Optional) create a trust for an on-premises Active Directory

If you don't plan to use your own on-premises Microsoft Active Directory, skip to [Step 3: Create an IAM role for Amazon Aurora to access the AWS Directory Service \(p. 1541\)](#).

To get Kerberos authentication using your on-premises Active Directory, you need to create a trusting domain relationship using a forest trust between your on-premises Microsoft Active Directory and the AWS Managed Microsoft AD directory (created in [Step 1: Create a directory using AWS Managed Microsoft AD \(p. 1536\)](#)). The trust can be one-way, where the AWS Managed Microsoft AD directory trusts the on-premises Microsoft Active Directory. The trust can also be two-way, where both Active Directories trust each other. For more information about setting up trusts using AWS Directory Service, see [When to create a trust relationship in the AWS Directory Service Administration Guide](#).

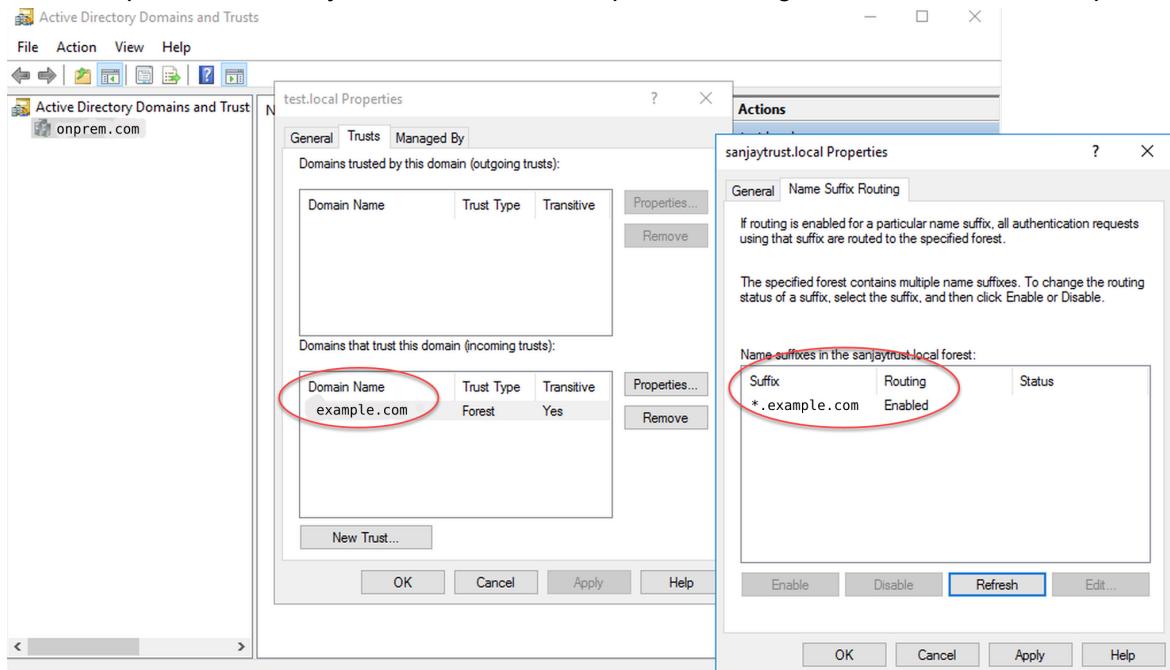
Note

If you use an on-premises Microsoft Active Directory:

- Windows clients must connect using specialized endpoints as described in [Connecting to PostgreSQL with Kerberos authentication \(p. 1546\)](#).
- Windows clients can't connect with custom endpoints ([p. 36](#)).

- For [global databases \(p. 225\)](#):
 - Windows clients can connect using instance endpoints or cluster endpoints in the primary AWS Region of the global database.
 - Windows clients can't connect using cluster endpoints in secondary AWS Regions.

Make sure that your on-premises Microsoft Active Directory domain name includes a DNS suffix routing that corresponds to the newly created trust relationship. The following screenshot shows an example.



Step 3: Create an IAM role for Amazon Aurora to access the AWS Directory Service

For Amazon Aurora to call AWS Directory Service for you, an IAM role that uses the managed IAM policy `AmazonRDSDirectoryServiceAccess` is required. This role allows Amazon Aurora to make calls to AWS Directory Service. (Note that this IAM role to access the AWS Directory Service is different than the IAM role used for [IAM database authentication \(p. 1743\)](#).)

When a DB instance is created using the AWS Management Console and the console user has the `iam:CreateRole` permission, the console creates this role automatically. In this case, the role name is `rds-directoryservice-kerberos-access-role`. Otherwise, create the IAM role manually. Choose **RDS** and then **RDS - Directory Service**. Attach the AWS managed policy `AmazonRDSDirectoryServiceAccess` to this role.

For more information about creating IAM roles for a service, see [Creating a role to delegate permissions to an AWS service](#) in the *IAM User Guide*.

Note

The IAM role used for Windows Authentication for RDS for Microsoft SQL Server can't be used for Amazon Aurora.

Optionally, you can create policies with the required permissions instead of using the managed IAM policy `AmazonRDSDirectoryServiceAccess`. In this case, the IAM role must have the following IAM trust policy.

```
{
```

```
"Version": "2012-10-17",
"Statement": [
  {
    "Sid": "",
    "Effect": "Allow",
    "Principal": {
      "Service": [
        "directoryservice.rds.amazonaws.com",
        "rds.amazonaws.com"
      ]
    },
    "Action": "sts:AssumeRole"
  }
]
```

The role must also have the following IAM role policy.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "ds:DescribeDirectories",
        "ds:AuthorizeApplication",
        "ds:UnauthorizeApplication",
        "ds:GetAuthorizedApplicationDetails"
      ],
      "Effect": "Allow",
      "Resource": "*"
    }
  ]
}
```

Step 4: Create and configure users

You can create users by using the Active Directory Users and Computers tool. This is one of the Active Directory Domain Services and Active Directory Lightweight Directory Services tools. In this case, *users* are individual people or entities who have access to your directory.

To create users in an AWS Directory Service directory, you must be connected to a Windows-based Amazon EC2 instance. Also, this EC2 instance must be a member of the AWS Directory Service directory. At the same time, you must be logged in as a user that has privileges to create users. For more information, see [Create a user](#) in the *AWS Directory Service Administration Guide*.

Step 5: Enable cross-VPC traffic between the directory and the DB instance

If you plan to locate the directory and the DB cluster in the same VPC, skip this step and move on to [Step 6: Create or modify a PostgreSQL DB cluster \(p. 1543\)](#).

If you plan to locate the directory and the DB instance in different VPCs, configure cross-VPC traffic using VPC peering or [AWS Transit Gateway](#).

The following procedure enables traffic between VPCs using VPC peering. Follow the instructions in [What is VPC peering?](#) in the *Amazon Virtual Private Cloud Peering Guide*.

To enable cross-VPC traffic using VPC peering

1. Set up appropriate VPC routing rules to ensure that network traffic can flow both ways.

2. Ensure that the DB instance security group can receive inbound traffic from the directory security group.
3. Ensure that there is no network access control list (ACL) rule to block traffic.

If a different AWS account owns the directory, you must share the directory.

To share the directory between AWS accounts

1. Start sharing the directory with the AWS account that the DB instance will be created in by following the instructions in [Tutorial: Sharing your AWS Managed Microsoft AD directory for seamless EC2 Domain-join](#) in the *AWS Directory Service Administration Guide*.
2. Sign in to the AWS Directory Service console using the account for the DB instance, and ensure that the domain has the SHARED status before proceeding.
3. While signed into the AWS Directory Service console using the account for the DB instance, note the **Directory ID** value. You use this directory ID to join the DB instance to the domain.

Step 6: Create or modify a PostgreSQL DB cluster

Create or modify a PostgreSQL DB cluster for use with your directory. You can use the console, CLI, or RDS API to associate a DB cluster with a directory. You can do this in one of the following ways:

- Create a new PostgreSQL DB cluster using the console, the [create-db-cluster](#) CLI command, or the [CreateDBCluster](#) RDS API operation. For instructions, see [Creating a DB cluster and connecting to a database on an Aurora PostgreSQL DB cluster \(p. 96\)](#).
- Modify an existing PostgreSQL DB cluster using the console, the [modify-db-cluster](#) CLI command, or the [ModifyDBCluster](#) RDS API operation. For instructions, see [Modifying an Amazon Aurora DB cluster \(p. 372\)](#).
- Restore a PostgreSQL DB cluster from a DB snapshot using the console, the [restore-db-cluster-from-db-snapshot](#) CLI command, or the [RestoreDBClusterFromDBSnapshot](#) RDS API operation. For instructions, see [Restoring from a DB cluster snapshot \(p. 497\)](#).
- Restore a PostgreSQL DB cluster to a point-in-time using the console, the [restore-db-instance-to-point-in-time](#) CLI command, or the [RestoreDBClusterToPointInTime](#) RDS API operation. For instructions, see [Restoring a DB cluster to a specified time \(p. 537\)](#).

Kerberos authentication is only supported for PostgreSQL DB clusters in a VPC. The DB cluster can be in the same VPC as the directory, or in a different VPC. The DB cluster must use a security group that allows ingress and egress within the directory's VPC so the DB cluster can communicate with the directory.

Console

When you use the console to create, modify, or restore a DB cluster, choose **Kerberos authentication** in the **Database authentication** section. Then choose **Browse Directory**. Select the directory or choose **Create a new directory** to use the Directory Service.

AWS CLI

When you use the AWS CLI, the following parameters are required for the DB cluster to be able to use the directory that you created:

- For the `--domain` parameter, use the domain identifier ("d-*" identifier) generated when you created the directory.
- For the `--domain-iam-role-name` parameter, use the role you created that uses the managed IAM policy `AmazonRDSDirectoryServiceAccess`.

For example, the following CLI command modifies a DB cluster to use a directory.

```
aws rds modify-db-cluster --db-cluster-identifier mydbinstance --domain d-Directory-ID --domain-iam-role-name role-name
```

Important

If you modify a DB cluster to enable Kerberos authentication, reboot the DB cluster after making the change.

Step 7: Create Kerberos authentication PostgreSQL logins

Use the RDS master user credentials to connect to the PostgreSQL DB cluster as you do with any other DB cluster . The DB instance is joined to the AWS Managed Microsoft AD domain. Thus, you can provision PostgreSQL logins and users from the Microsoft Active Directory users and groups in your domain. To manage database permissions, you grant and revoke standard PostgreSQL permissions to these logins.

To allow an Active Directory user to authenticate with PostgreSQL, use the RDS master user credentials. You use these credentials to connect to the PostgreSQL DB cluster as you do with any other DB cluster . After you're logged in, create an externally authenticated user in PostgreSQL and grant the `rds_ad` role to this user.

```
CREATE USER "username@CORP.EXAMPLE.COM" WITH LOGIN;  
GRANT rds_ad TO "username@CORP.EXAMPLE.COM";
```

Replace `username` with the user name and include the domain name in uppercase. Users (both humans and applications) from your domain can now connect to the RDS PostgreSQL cluster from a domain-joined client machine using Kerberos authentication.

Note that a database user can use either Kerberos or IAM authentication but not both, so this user can't also have the `rds_iam` role. This also applies to nested memberships. For more information, see [IAM database authentication \(p. 1743\)](#).

Step 8: Configure a PostgreSQL client

To configure a PostgreSQL client, take the following steps:

- Create a `krb5.conf` file (or equivalent) to point to the domain.
- Verify that traffic can flow between the client host and AWS Directory Service. Use a network utility such as Netcat for the following:
 - Verify traffic over DNS for port 53.
 - Verify traffic over TCP/UDP for port 53 and for Kerberos, which includes ports 88 and 464 for AWS Directory Service.
- Verify that traffic can flow between the client host and the DB instance over the database port. For example, use `psql` to connect and access the database.

The following is sample `krb5.conf` content for AWS Managed Microsoft AD.

```
[libdefaults]  
default_realm = EXAMPLE.COM  
[realms]  
EXAMPLE.COM = {  
    kdc = example.com  
    admin_server = example.com  
}  
[domain_realm]
```

```
.example.com = EXAMPLE.COM
example.com = EXAMPLE.COM
```

The following is sample krb5.conf content for an on-premises Microsoft Active Directory.

```
[libdefaults]
default_realm = EXAMPLE.COM
[realms]
EXAMPLE.COM = {
    kdc = example.com
    admin_server = example.com
}
ONPREM.COM = {
    kdc = onprem.com
    admin_server = onprem.com
}
[domain_realm]
.example.com = EXAMPLE.COM
example.com = EXAMPLE.COM
.onprem.com = ONPREM.COM
onprem.com = ONPREM.COM
.rds.amazonaws.com = EXAMPLE.COM
.amazonaws.com.cn = EXAMPLE.COM
.amazon.com = EXAMPLE.COM
```

Managing a DB cluster in a Domain

You can use the console, the CLI, or the RDS API to manage your DB cluster and its relationship with your Microsoft Active Directory. For example, you can associate an Active Directory to enable Kerberos authentication. You can also remove the association for an Active Directory to disable Kerberos authentication. You can also move a DB cluster to be externally authenticated by one Microsoft Active Directory to another.

For example, using the CLI, you can do the following:

- To reattempt enabling Kerberos authentication for a failed membership, use the [modify-db-cluster](#) CLI command. Specify the current membership's directory ID for the --domain option.
- To disable Kerberos authentication on a DB instance, use the [modify-db-cluster](#) CLI command. Specify none for the --domain option.
- To move a DB instance from one domain to another, use the [modify-db-cluster](#) CLI command. Specify the domain identifier of the new domain for the --domain option.

Understanding Domain membership

After you create or modify your DB cluster, the DB instances become members of the domain. You can view the status of the domain membership in the console or by running the [describe-db-instances](#) CLI command. The status of the DB instance can be one of the following:

- **kerberos-enabled** – The DB instance has Kerberos authentication enabled.
- **enabling-kerberos** – AWS is in the process of enabling Kerberos authentication on this DB instance.
- **pending-enable-kerberos** – Enabling Kerberos authentication is pending on this DB instance.
- **pending-maintenance-enable-kerberos** – AWS will attempt to enable Kerberos authentication on the DB instance during the next scheduled maintenance window.
- **pending-disable-kerberos** – Disabling Kerberos authentication is pending on this DB instance.
- **pending-maintenance-disable-kerberos** – AWS will attempt to disable Kerberos authentication on the DB instance during the next scheduled maintenance window.

- **enable-kerberos-failed** – A configuration problem prevented AWS from enabling Kerberos authentication on the DB instance. Correct the configuration problem before reissuing the command to modify the DB instance.
- **disabling-kerberos** – AWS is in the process of disabling Kerberos authentication on this DB instance.

A request to enable Kerberos authentication can fail because of a network connectivity issue or an incorrect IAM role. In some cases, the attempt to enable Kerberos authentication might fail when you create or modify a DB cluster. If so, make sure that you are using the correct IAM role, then modify the DB cluster to join the domain.

Connecting to PostgreSQL with Kerberos authentication

You can connect to PostgreSQL with Kerberos authentication with the pgAdmin interface or with a command line interface such as psql. For more information about connecting, see [Connecting to an Amazon Aurora PostgreSQL DB cluster \(p. 285\)](#).

pgAdmin

To use pgAdmin to connect to PostgreSQL with Kerberos authentication, take the following steps:

1. Launch the pgAdmin application on your client computer.
2. On the **Dashboard** tab, choose **Add New Server**.
3. In the **Create - Server** dialog box, enter a name on the **General** tab to identify the server in pgAdmin.
4. On the **Connection** tab, enter the following information from your Aurora PostgreSQL database:
 - For **Host**, enter the endpoint. Use a format such as *PostgreSQL-endpoint.AWS-Region.rds.amazonaws.com*.

If you're using an on-premises Microsoft Active Directory from a Windows client, then you need to connect using a specialized endpoint. Instead of using the Amazon domain `rds.amazonaws.com` in the host endpoint, use the domain name of the AWS Managed Active Directory.

For example, suppose that the domain name for the AWS Managed Active Directory is `corp.example.com`. Then for **Host**, use the format *PostgreSQL-endpoint.AWS-Region.corp.example.com*.

- For **Port**, enter the assigned port.
- For **Maintenance database**, enter the name of the initial database to which the client will connect.
- For **Username**, enter the user name that you entered for Kerberos authentication in [Step 7: Create Kerberos authentication PostgreSQL logins \(p. 1544\)](#).

5. Choose **Save**.

Psq

To use psql to connect to PostgreSQL with Kerberos authentication, take the following steps:

1. At a command prompt, run the following command.

```
kinit username
```

Replace `username` with the user name. At the prompt, enter the password stored in the Microsoft Active Directory for the user.

2. If the PostgreSQL DB cluster is using a publicly accessible VPC, put a private IP address for your DB cluster endpoint in your /etc/hosts file on the EC2 client. For example, the following commands obtain the private IP address and then put it in the /etc/hosts file.

```
% dig +short PostgreSQL-endpoint.AWS-Region.rds.amazonaws.com
;; Truncated, retrying in TCP mode.
ec2-34-210-197-118.AWS-Region.compute.amazonaws.com.
34.210.197.118

% echo " 34.210.197.118  PostgreSQL-endpoint.AWS-Region.rds.amazonaws.com" >> /etc/hosts
```

If you're using an on-premises Microsoft Active Directory from a Windows client, then you need to connect using a specialized endpoint. Instead of using the Amazon domain `rds.amazonaws.com` in the host endpoint, use the domain name of the AWS Managed Active Directory.

For example, suppose that the domain name for your AWS Managed Active Directory is `corp.example.com`. Then use the format `PostgreSQL-endpoint.AWS-Region.corp.example.com` for the endpoint and put it in the /etc/hosts file.

```
% echo " 34.210.197.118  PostgreSQL-endpoint.AWS-Region.corp.example.com" >> /etc/hosts
```

3. Use the following psql command to log in to a PostgreSQL DB cluster that is integrated with Active Directory. Use a cluster or instance endpoint.

```
psql -U username@CORP.EXAMPLE.COM -p 5432 -h PostgreSQL-endpoint.AWS-Region.rds.amazonaws.com postgres
```

To log in to the PostgreSQL DB cluster from a Windows client using an on-premises Active Directory, use the following psql command with the domain name from the previous step (`corp.example.com`):

```
psql -U username@CORP.EXAMPLE.COM -p 5432 -h PostgreSQL-endpoint.AWS-Region.corp.example.com postgres
```

Amazon Aurora PostgreSQL reference

Topics

- [Amazon Aurora PostgreSQL parameters \(p. 1547\)](#)
- [Amazon Aurora PostgreSQL wait events \(p. 1570\)](#)
- [Aurora PostgreSQL functions reference \(p. 1588\)](#)

Amazon Aurora PostgreSQL parameters

You manage your Amazon Aurora DB cluster in the same way that you manage Amazon RDS DB instances, by using parameters in a DB parameter group. However, Amazon Aurora differs from Amazon RDS in that an Aurora DB cluster has multiple DB instances. Some of the parameters that you use to manage your Amazon Aurora DB cluster apply to the entire cluster, while other parameters apply only to a given DB instance in the DB cluster, as follows:

- **DB cluster parameter group** – A DB cluster parameter group contains the set of engine configuration parameters that apply throughout the Aurora DB cluster. For example, cluster cache management is a feature of an Aurora DB cluster that's controlled by the `apg_ccm_enabled` parameter which is part of

the DB cluster parameter group. The DB cluster parameter group also contains default settings for the DB parameter group for the DB instances that make up the cluster.

- **DB parameter group** – A DB parameter group is the set of engine configuration values that apply to a specific DB instance of that engine type. The DB parameter groups for the PostgreSQL DB engine are used by an RDS for PostgreSQL DB instance, and Aurora PostgreSQL DB cluster. These configuration settings apply to properties that can vary among the DB instances within an Aurora cluster, such as the sizes for memory buffers.

You manage cluster-level parameters in DB cluster parameter groups. You manage instance-level parameters in DB parameter groups. You can manage parameters using the Amazon RDS console, the AWS CLI, or the Amazon RDS API. There are separate commands for managing cluster-level parameters and instance-level parameters.

- To manage cluster-level parameters in a DB cluster parameter group, use the [modify-db-cluster-parameter-group](#) AWS CLI command.
- To manage instance-level parameters in a DB parameter group for a DB instance in a DB cluster, use the [modify-db-parameter-group](#) AWS CLI command.

To learn more about the AWS CLI, see [Using the AWS CLI in the AWS Command Line Interface User Guide](#).

For more information about parameter groups, see [Working with DB parameter groups and DB cluster parameter groups \(p. 339\)](#).

Viewing Aurora PostgreSQL DB cluster and DB parameters

You can view all available default parameter groups for RDS for PostgreSQL DB instances and for Aurora PostgreSQL DB clusters in the AWS Management Console. The default parameter groups for all DB engines and DB cluster types and versions are listed for each AWS Region. Any custom parameter groups are also listed.

Rather than viewing in the AWS Management Console, you can also list parameters contained in DB cluster parameter groups and DB parameter groups by using the AWS CLI or the Amazon RDS API. For example, to list parameters in a DB cluster parameter group you use the [describe-db-cluster-parameters](#) AWS CLI command as follows:

```
aws rds describe-db-cluster-parameters --db-cluster-parameter-group-name default.aurora-postgresql12
```

The command returns detailed JSON descriptions of each parameter. To reduce the amount of information returned, you can specify what you want by using the `--query` option. For example, you can get the parameter name, its description, and allowed values for the default Aurora PostgreSQL 12 DB cluster parameter group as follows:

For Linux, macOS, or Unix:

```
aws rds describe-db-cluster-parameters --db-cluster-parameter-group-name default.aurora-postgresql12 \
    --query 'Parameters[]'.
[{"ParameterName":ParameterName, "Description":Description, "ApplyType":ApplyType, "AllowedValues":AllowedValues}]
```

For Windows:

```
aws rds describe-db-cluster-parameters --db-cluster-parameter-group-name default.aurora-postgresql12 ^
```

```
--query "Parameters[]".
[{"ParameterName":ParameterName,Description:Description,ApplyType:ApplyType,AllowedValues:AllowedValues}]
```

An Aurora DB cluster parameter group includes the DB instance parameter group and default values for a given Aurora DB engine. You can get the list of DB parameters from the same default Aurora PostgreSQL default parameter group by usng the [describe-db-parameters](#) AWS CLI command as shown following.

For Linux, macOS, or Unix:

```
aws rds describe-db-parameters --db-parameter-group-name default.aurora-postgresql12 \
--query 'Parameters[]'.
[{"ParameterName":ParameterName,Description:Description,ApplyType:ApplyType,AllowedValues:AllowedValues}]
```

For Windows:

```
aws rds describe-db-parameters --db-parameter-group-name default.aurora-postgresql12 ^
--query "Parameters[]".
[{"ParameterName":ParameterName,Description:Description,ApplyType:ApplyType,AllowedValues:AllowedValues}]
```

The preceding commands return lists of parameters from the DB cluster or DB parameter group with descriptions and other details specified in the query. Following is an example response.

```
[  
  [  
    {  
      "ParameterName": "apg_enable_batch_mode_function_execution",  
      "ApplyType": "dynamic",  
      "Description": "Enables batch-mode functions to process sets of rows at a  
time.",  
      "AllowedValues": "0,1"  
    }  
  ],  
  [  
    {  
      "ParameterName": "apg_enable_correlated_any_transform",  
      "ApplyType": "dynamic",  
      "Description": "Enables the planner to transform correlated ANY Sublink (IN/NOT  
IN subquery) to JOIN when possible.",  
      "AllowedValues": "0,1"  
    }  
  ],...  
]
```

Following are tables containing values for the default DB cluster parameter and DB parameter for Aurora PostgreSQL version 13.

Aurora PostgreSQL cluster-level parameters

The following table lists some of parameters available in the default DB cluster parameter group for Aurora PostgreSQL version 13. If you create an Aurora PostgreSQL DB cluster without specifying your own custom DB parameter group, your DB cluster is created using the default Aurora DB cluster parameter group for the version chosen, such as `default.aurora-postgresql13`, `default.aurora-postgresql12`, and so on.

Note

All parameters in the following table are *dynamic* unless otherwise noted in the description.

For a listing of the DB instance parameters for the same default Aurora parameter group, see [Aurora PostgreSQL instance-level parameters \(p. 1560\)](#).

Parameter name	Description
<code>ansi_constraint_trigger_order</code>	Change the firing order of constraint triggers to be compatible with the ANSI SQL standard.
<code>ansi_force_foreign_key_check</code>	Ensure referential actions such as cascaded delete or cascaded update will always occur regardless of the various trigger contexts that exist for the action.
<code>ansi_qualified_update_set</code>	Support table and schema qualifiers in UPDATE ... SET statements.
<code>apg_ccm_enabled</code>	Enable or disable cluster cache management for the cluster.
<code>apg_enable_batch_mode_function</code>	Enables batch mode functions to process sets of rows at a time.
<code>apg_enable_correlated_any_subquery</code>	Enables the planner to transform correlated ANY Sublink (IN/NOT IN subquery) to JOIN when possible.
<code>apg_enable_function_migration</code>	Enables the planner to migrate eligible scalar functions to the FROM clause.
<code>apg_enable_not_in_transform</code>	Enables the planner to transform NOT IN subquery to ANTI JOIN when possible.
<code>apg_enable_remove_redundant_innerjoin</code>	Enables the planner to remove redundant inner joins.
<code>apg_enable_semijoin_pushdown</code>	Enables the use of semijoin filters for hash joins.
<code>apg_plan_mgmt.capture_plan</code>	Captures plan baseline mode. manual - enable plan capture for any SQL statement off - disable plan capture automatic - enable plan capture for statements in pg_stat_statements that satisfy the eligibility criteria.
<code>apg_plan_mgmt.max_databases</code>	Static. Sets the maximum number of databases that may manage queries using apg_plan_mgmt.
<code>apg_plan_mgmt.max_plans</code>	Static. Sets the maximum number of plans that may be cached by apg_plan_mgmt.
<code>apg_plan_mgmt.plan_retention</code>	Static. Maximum number of days since a plan was last_used before a plan will be automatically deleted.
<code>apg_plan_mgmt.unapproved_plan_threshold</code>	Estimated total plan cost below which an Unapproved plan will be executed.
<code>apg_plan_mgmt.use_plan_baseline</code>	Allows approved or fixed plans for managed statements.

Parameter name	Description
array_nulls	Enable input of NULL elements in arrays.
authentication_timeout	(s) Sets the maximum allowed time to complete client authentication.
auto_explain.log_analyze	Use EXPLAIN ANALYZE for plan logging.
auto_explain.log_buffers	Log buffers usage.
auto_explain.log_format	EXPLAIN format to be used for plan logging.
auto_explain.log_min_duration	Sets the minimum execution time above which plans will be logged.
auto_explain.log_nested_statements	Logs nested statements.
auto_explain.log_timing	Collect timing data not just row counts.
auto_explain.log_triggers	Include trigger statistics in plans.
auto_explain.log_verbose	Use EXPLAIN VERBOSE for plan logging.
auto_explain.sample_rate	Fraction of queries to process.
autovacuum	Starts the autovacuum subprocess.
autovacuum_analyze_scale_factor	Number of tuple inserts updates or deletes prior to analyze as a fraction of reltuples.
autovacuum_analyze_threshold	Minimum number of tuple inserts updates or deletes prior to analyze.
autovacuum_freeze_max_age	Static. Age at which to autovacuum a table to prevent transaction ID wraparound.
autovacuum_max_workers	Static. Sets the maximum number of simultaneously running autovacuum worker processes.
autovacuum_multixact_freeze_threshold	Static. Multixact age at which to autovacuum a table to prevent multixact wraparound.
autovacuum_naptime	(s) Time to sleep between autovacuum runs.
autovacuum_vacuum_cost_delay	(ms) Vacuum cost delay in milliseconds for autovacuum.
autovacuum_vacuum_cost_limit	Vacuum cost amount available before napping for autovacuum.
autovacuum_vacuum_insert_scale_factor	Number of tuple inserts prior to vacuum as a fraction of reltuples.
autovacuum_vacuum_insert_threshold	Minimum number of tuple inserts prior to vacuum or -1 to disable insert vacuums.
autovacuum_vacuum_scale_factor	Number of tuple updates or deletes prior to vacuum as a fraction of reltuples.
autovacuum_vacuum_threshold	Minimum number of tuple updates or deletes prior to vacuum.
autovacuum_work_mem	(kB) Sets the maximum memory to be used by each autovacuum worker process.

Parameter name	Description
<code>babelfishpg_tsquery.default_locale</code>	Static. Default locale to be used for collations created by CREATE COLLATION.
<code>babelfishpg_tds.port</code>	Static. Sets the TDS TCP port the server listens on.
<code>babelfishpg_tds.tds_debug</code>	Sets logging level in TDS 0 disables logging
<code>babelfishpg_tds.tds_default_precision</code>	Sets the default precision of numeric type to be sent in the TDS column metadata if the engine does not specify one.
<code>babelfishpg_tds.tds_default_scale</code>	Sets the default scale of numeric type to be sent in the TDS column metadata if the engine does not specify one.
<code>babelfishpg_tds.tds_default_packet_size</code>	Sets the default packet size for all the SQL Server clients being connected
<code>babelfishpg_tds.tds_default_protocol_version</code>	Sets a default TDS protocol version for all the clients being connected
<code>babelfishpg_tds.tds_ssl_encrypt</code>	Sets the SSL Encryption option
<code>babelfishpg_tds.tds_ssl_max_version</code>	Sets the maximum SSL/TLS protocol version to use for tds session.
<code>babelfishpg_tds.tds_ssl_min_version</code>	Sets the minimum SSL/TLS protocol version to use for tds session.
<code>babelfishpg_tsquery.migration</code>	Static. Defines if multiple user databases are supported
<code>backend_flush_after</code>	(8Kb) Number of pages after which previously performed writes are flushed to disk.
<code>backslash_quote</code>	Sets whether \\ is allowed in string literals.
<code>bytea_output</code>	Sets the output format for bytea.
<code>check_function_bodies</code>	Check function bodies during CREATE FUNCTION.
<code>client_min_messages</code>	Sets the message levels that are sent to the client.
<code>constraint_exclusion</code>	Enables the planner to use constraints to optimize queries.
<code>cpu_index_tuple_cost</code>	Sets the planners estimate of the cost of processing each index entry during an index scan.
<code>cpu_operator_cost</code>	Sets the planners estimate of the cost of processing each operator or function call.
<code>cpu_tuple_cost</code>	Sets the planners estimate of the cost of processing each tuple (row).
<code>cron.log_run</code>	Static. Log all jobs runs into the job_run_details table
<code>cron.log_statement</code>	Static. Log all cron statements prior to execution.
<code>cron.max_running_jobs</code>	Static. Maximum number of jobs that can run concurrently.
<code>cursor_tuple_fraction</code>	Sets the planners estimate of the fraction of a cursors rows that will be retrieved.
<code>db_user_namespace</code>	Enables per-database user names.

Parameter name	Description
deadlock_timeout	(ms) Sets the time to wait on a lock before checking for deadlock.
debug_pretty_print	Indents parse and plan tree displays.
debug_print_parse	Logs each query's parse tree.
debug_print_plan	Logs each query's execution plan.
debug_print_rewritten	Logs each query's rewritten parse tree.
default_statistics_target	Sets the default statistics target.
default_transaction_deferrable	Sets the default deferrable status of new transactions.
default_transaction_isolation	Sets the transaction isolation level of each new transaction.
default_transaction_read_only	Sets the default read-only status of new transactions.
effective_cache_size	(8kB) Sets the planner's assumption about the size of the disk cache.
effective_io_concurrency	Number of simultaneous requests that can be handled efficiently by the disk subsystem.
enable_bitmapscan	Enables the planner's use of bitmap-scan plans.
enable_gathermerge	Enables the planner's use of gather merge plans.
enable_hashagg	Enables the planner's use of hashed aggregation plans.
enable_hashjoin	Enables the planner's use of hash join plans.
enable_incremental_sort	Enables the planner's use of incremental sort steps.
enable_indexonlyscan	Enables the planner's use of index-only-scan plans.
enable_indexscan	Enables the planner's use of index-scan plans.
enable_material	Enables the planner's use of materialization.
enable_mergejoin	Enables the planner's use of merge join plans.
enable_nestloop	Enables the planner's use of nested-loop join plans.
enable_parallel_append	Enables the planner's use of parallel append plans.
enable_parallel_hash	Enables the planner's use of parallel hash plans.
enable_partition_pruning	Enables plan-time and run-time partition pruning.
enable_partitionwise_aggregation	Enables partitionwise aggregation and grouping.
enable_partitionwise_join	Enables partitionwise join.
enable_seqscan	Enables the planner's use of sequential-scan plans.
enable_sort	Enables the planner's use of explicit sort steps.
enable_tidscan	Enables the planner's use of TID scan plans.
escape_string_warning	Warn about backslash escapes in ordinary string literals.

Parameter name	Description
<code>exit_on_error</code>	Terminate session on any error.
<code>extra_float_digits</code>	Sets the number of digits displayed for floating-point values.
<code>force_parallel_mode</code>	Forces use of parallel query facilities.
<code>fromCollapse_limit</code>	Sets the FROM-list size beyond which subqueries are not collapsed.
<code>geqo</code>	Enables genetic query optimization.
<code>geqo_effort</code>	GEQO: effort is used to set the default for other GEQO parameters.
<code>geqo_generations</code>	GEQO: number of iterations of the algorithm.
<code>geqo_pool_size</code>	GEQO: number of individuals in the population.
<code>geqo_seed</code>	GEQO: seed for random path selection.
<code>geqo_selection_bias</code>	GEQO: selective pressure within the population.
<code>geqo_threshold</code>	Sets the threshold of FROM items beyond which GEQO is used.
<code>gin_fuzzy_search_limit</code>	Sets the maximum allowed result for exact search by GIN.
<code>gin_pending_list_limit</code>	(kB) Sets the maximum size of the pending list for GIN index.
<code>hash_mem_multiplier</code>	Multiple of work_mem to use for hash tables.
<code>hot_standby_feedback</code>	Allows feedback from a hot standby to the primary that will avoid query conflicts.
<code>huge_pages</code>	Static. Use of huge pages on Linux.
<code>idle_in_transaction_session_timeout</code>	(ms) Sets the maximum allowed duration of any idling transaction.
<code>intervalstyle</code>	Sets the display format for interval values.
<code>joinCollapse_limit</code>	Sets the FROM-list size beyond which JOIN constructs are not flattened.
<code>lo_compat_privileges</code>	Enables backward compatibility mode for privilege checks on large objects.
<code>log_autovacuum_min_duration</code>	(ms) Sets the minimum execution time above which autovacuum actions will be logged.
<code>log_connections</code>	Logs each successful connection.
<code>log_destination</code>	Sets the destination for server log output.
<code>log_disconnections</code>	Logs end of a session including duration.
<code>log_duration</code>	Logs the duration of each completed SQL statement.
<code>log_error_verbosity</code>	Sets the verbosity of logged messages.
<code>log_executor_stats</code>	Writes executor performance statistics to the server log.
<code>log_filename</code>	Sets the file name pattern for log files.
<code>log_hostname</code>	Logs the host name in the connection logs.

Parameter name	Description
<code>log_lock_waits</code>	Logs long lock waits.
<code>log_min_duration_sample</code>	(ms) Sets the minimum execution time above which a sample of statements will be logged. Sampling is determined by <code>log_statement_sample_rate</code> .
<code>log_min_duration_statement</code>	(ms) Sets the minimum execution time above which statements will be logged.
<code>log_min_error_statement</code>	Causes all statements generating error at or above this level to be logged.
<code>log_min_messages</code>	Sets the message levels that are logged.
<code>log_parameter_max_length</code>	(B) When logging statements limit logged parameter values to first N bytes.
<code>log_parameter_max_length_or_error</code>	(B) When reporting an error limit logged parameter values to first N bytes.
<code>log_parser_stats</code>	Writes parser performance statistics to the server log.
<code>log_planner_stats</code>	Writes planner performance statistics to the server log.
<code>log_replication_commands</code>	Logs each replication command.
<code>log_rotation_age</code>	(min) Automatic log file rotation will occur after N minutes.
<code>log_rotation_size</code>	(kB) Automatic log file rotation will occur after N kilobytes.
<code>log_statement</code>	Sets the type of statements logged.
<code>log_statement_sample_rate</code>	Fraction of statements exceeding <code>log_min_duration_sample</code> to be logged.
<code>log_statement_stats</code>	Writes cumulative performance statistics to the server log.
<code>log_temp_files</code>	(kB) Log the use of temporary files larger than this number of kilobytes.
<code>log_transaction_sample_rate</code>	Set the fraction of transactions to log for new transactions.
<code>log_truncate_on_rotation</code>	Truncate existing log files of same name during log rotation.
<code>logging_collector</code>	Static. Start a subprocess to capture stderr output and/or csvlogs into log files.
<code>logical_decoding_work_mem</code>	(kB) This much memory can be used by each internal reorder buffer before spilling to disk.
<code>maintenance_io_concurrency</code>	A variant of <code>effective_io_concurrency</code> that is used for maintenance work.
<code>maintenance_work_mem</code>	(kB) Sets the maximum memory to be used for maintenance operations.
<code>max_connections</code>	Static. Sets the maximum number of concurrent connections.
<code>max_files_per_process</code>	Static. Sets the maximum number of simultaneously open files for each server process.

Parameter name	Description
max_locks_per_transaction	Static. Sets the maximum number of locks per transaction.
max_logical_replication_workers	Static. Maximum number of logical replication worker processes.
max_parallel_maintenance_workers	Sets the maximum number of parallel processes per maintenance operation.
max_parallel_workers	Sets the maximum number of parallel workers than can be active at one time.
max_parallel_workers_per_gather	Sets the maximum number of parallel processes per executor node.
max_pred_locks_per_page	Sets the maximum number of predicate-locked tuples per page.
max_pred_locks_per_relation	Sets the maximum number of predicate-locked pages and tuples per relation.
max_pred_locks_per_transaction	Static. Sets the maximum number of predicate locks per transaction.
max_prepared_transactions	Static. Sets the maximum number of simultaneously prepared transactions.
max_replication_slots	Static. Sets the maximum number of replication slots that the server can support.
max_slot_wal_keep_size	(MB) Replication slots will be marked as failed and segments released for deletion or recycling if this much space is occupied by WAL on disk.
max_stack_depth	(kB) Sets the maximum stack depth in kilobytes.
max_standby_streaming_delay	(ms) Sets the maximum delay before canceling queries when a hot standby server is processing streamed WAL data.
max_sync_workers_per_subscription	Maximum number of synchronization workers per subscription
max_wal_senders	Static. Sets the maximum number of simultaneously running WAL sender processes.
max_worker_processes	Static. Sets the maximum number of concurrent worker processes.
min_parallel_index_scan_size	(8kB) Sets the minimum amount of index data for a parallel scan.
min_parallel_table_scan_size	(8kB) Sets the minimum amount of table data for a parallel scan.
old_snapshot_threshold	Static. (min) Time before a snapshot is too old to read pages changed after the snapshot was taken.
operator_precedence_warning	Emit a warning for constructs that changed meaning since PostgreSQL 9.4.
parallel_leader_participation	Controls whether Gather and Gather Merge also run subplans.
parallel_setup_cost	Sets the planners estimate of the cost of starting up worker processes for parallel query.
parallel_tuple_cost	Sets the planners estimate of the cost of passing each tuple (row) from worker to master backend.

Parameter name	Description
<code>password_encryption</code>	Encrypt passwords.
<code>pg_bigm.enable_recheck</code>	It specifies whether to perform Recheck which is an internal process of full text search.
<code>pg_bigm.gin_key_limit</code>	It specifies the maximum number of 2-grams of the search keyword to be used for full text search.
<code>pg_hint_plan.debug_print</code>	Logs results of hint parsing.
<code>pg_hint_plan.enable_hint</code>	Force planner to use plans specified in the hint comment preceding to the query.
<code>pg_hint_plan.enable_hint_table</code>	Forces planner to not get hint by using table lookups.
<code>pg_hint_plan.message_level</code>	Message level of debug messages.
<code>pg_hint_plan.parse_messages</code>	Message level of parse errors.
<code>pg_prewarm.autoprewarm</code>	Starts the autoprewarm worker.
<code>pg_prewarm.autoprewarm_interval</code>	Sets the interval between dumps of shared buffers
<code>pg_stat_statements.max</code>	Static. Sets the maximum number of statements tracked by pg_stat_statements.
<code>pg_stat_statements.save</code>	Save pg_stat_statements statistics across server shutdowns.
<code>pg_stat_statements.track</code>	Selects which statements are tracked by pg_stat_statements.
<code>pg_stat_statements.track_plans</code>	Selects whether planning duration is tracked by pg_stat_statements.
<code>pg_stat_statements.track_utilities</code>	Selects whether utility commands are tracked by pg_stat_statements.
<code>pgaudit.log</code>	Specifies which classes of statements will be logged by session audit logging.
<code>pgaudit.log_catalog</code>	Specifies that session logging should be enabled in the case where all relations in a statement are in pg_catalog.
<code>pgaudit.log_level</code>	Specifies the log level that will be used for log entries.
<code>pgaudit.log_parameter</code>	Specifies that audit logging should include the parameters that were passed with the statement.
<code>pgaudit.log_relation</code>	Specifies whether session audit logging should create a separate log entry for each relation (TABLE VIEW etc.) referenced in a SELECT or DML statement.
<code>pgaudit.log_statement_once</code>	Specifies whether logging will include the statement text and parameters with the first log entry for a statement/substatement combination or with every entry.
<code>pgaudit.role</code>	Specifies the master role to use for object audit logging.
<code>pglogical.batch_inserts</code>	Static. Batch inserts if possible
<code>pglogical.conflict_log_level</code>	Sets log level used for logging resolved conflicts.

Parameter name	Description
<code>pglogical.conflict_resolution</code>	Sets method used for conflict resolution for resolvable conflicts.
<code>pglogical.synchronous_commit</code>	Static. pglogical specific synchronous commit value
<code>pglogical.use_spi</code>	Use SPI instead of low-level API for applying changes
<code>plan_cache_mode</code>	Controls the planner selection of custom or generic plan.
<code>postgis.gdal_enabled_drivers</code>	Static. Enable or disable GDAL drivers used with PostGIS in Postgres 9.3.5 and above.
<code>quote_all_identifiers</code>	When generating SQL fragments quote all identifiers.
<code>random_page_cost</code>	Sets the planners estimate of the cost of a nonsequentially fetched disk page.
<code>rdkit.do_chiral_sss</code>	Should stereochemistry be taken into account in substructure matching. If false no stereochemistry information is used in substructure matches.
<code>rds.adaptive_autovacuum</code>	RDS parameter to enable/disable adaptive autovacuum.
<code>rds.babelfish_status</code>	Static. RDS parameter to enable/disable Babelfish for Aurora PostgreSQL.
<code>rds.enable_plan_management</code>	Static. Enable or disable the apg_plan_mgmt extension.
<code>rds.force_admin_logging_level</code>	Set log messages for RDS admin user actions in customer databases.
<code>rds.force_autovacuum_logging_level</code>	Set log messages related to autovacuum operations.
<code>rds.force_ssl</code>	Force SSL connections.
<code>rds.global_db_rpo</code>	(s) Recovery point objective threshold in seconds that blocks user commits when it is violated.
<code>rds.log_retention_period</code>	Amazon RDS will delete PostgreSQL log that are older than N minutes.
<code>rds.logical_replication</code>	Static. Enables logical decoding.
<code>rds.pg_stat_ramdisk_size</code>	Static. Size of the stats ramdisk in MB. A nonzero value will setup the ramdisk.
<code>rds.rds_superuser_reserved_slots</code>	Static. Sets the number of connection slots reserved for rds_superusers.
<code>rds.restrict_password_commands</code>	Static. Restricts password-related commands to members of rds_password
<code>restart_after_crash</code>	Reinitialize server after backend crash.
<code>row_security</code>	Enable row security.
<code>seq_page_cost</code>	Sets the planners estimate of the cost of a sequentially fetched disk page.
<code>session_replication_role</code>	Sets the sessions behavior for triggers and rewrite rules.

Parameter name	Description
shared_buffers	(8kB) Sets the number of shared memory buffers used by the server.
shared_preload_libraries	Static. Lists shared libraries to preload into server.
update_process_title	Updates the process title to show the active SQL command.
vacuum_cleanup_index_scale	Number of tuple inserts prior to index cleanup as a fraction of reltuples.
vacuum_cost_delay	(ms) Vacuum cost delay in milliseconds.
vacuum_cost_limit	Vacuum cost amount available before napping.
vacuum_cost_page_dirty	Vacuum cost for a page dirtied by vacuum.
vacuum_cost_page_hit	Vacuum cost for a page found in the buffer cache.
vacuum_cost_page_miss	Vacuum cost for a page not found in the buffer cache.
vacuum_defer_cleanup_age	Number of transactions by which VACUUM and HOT cleanup should be deferred if any.
vacuum_freeze_min_age	Minimum age at which VACUUM should freeze a table row.
vacuum_freeze_table_age	Age at which VACUUM should scan whole table to freeze tuples.
vacuum_multixact_freeze_min	Minimum age at which VACUUM should freeze a MultiXactId in a table row.
vacuum_multixact_freeze_table	MultiXact age at which VACUUM should scan whole table to freeze tuples.
wal_buffers	Static. (8kB) Sets the number of disk-page buffers in shared memory for WAL.
wal_receiver_create_temp_slot	Sets whether a WAL receiver should create a temporary replication slot if no permanent slot is configured.
wal_receiver_status_interval	(s) Sets the maximum interval between WAL receiver status reports to the primary.
wal_receiver_timeout	(ms) Sets the maximum wait time to receive data from the primary.
wal_sender_timeout	(ms) Sets the maximum time to wait for WAL replication.
work_mem	(kB) Sets the maximum memory to be used for query workspaces.

Aurora PostgreSQL instance-level parameters

The following table shows all of the parameters that apply to a specific DB instance in an Aurora PostgreSQL DB cluster. This list was generated by running the [describe-db-parameters](#) AWS CLI command with `default.aurora-postgresql13` for the `--db-parameter-group-name` value.

Note

All parameters in the following table are *dynamic* unless otherwise noted in the description.

For a listing of the DB cluster parameters for the default Aurora parameter group, see [Aurora PostgreSQL cluster-level parameters \(p. 1550\)](#).

Parameter name	Description
<code>apg_enable_batch_mode_func</code>	Enables <code>batch mode</code> functions to process sets of rows at a time.
<code>apg_enable_correlated_any_in_subquery</code>	Enables the planner to transform correlated ANY Sublink (IN/NOT IN subquery) to JOIN when possible.
<code>apg_enable_function_migration</code>	Enables the planner to migrate eligible scalar functions to the FROM clause.
<code>apg_enable_not_in_transform</code>	Enables the planner to transform NOT IN subquery to ANTI JOIN when possible.
<code>apg_enable_remove_redundant_inner_joins</code>	Enables the planner to remove redundant inner joins.
<code>apg_enable_semijoin_pushdown</code>	Enables the use of semijoin filters for hash joins.
<code>apg_plan_mgmt.capture_plan</code>	Captures plan baseline mode. manual - enable plan capture for any SQL statement, off - disable plan capture, automatic - enable plan capture for statements in pg_stat_statements that satisfy the eligibility criteria.
<code>apg_plan_mgmt.max_databases</code>	Static. Sets the maximum number of databases that may manage queries using apg_plan_mgmt.
<code>apg_plan_mgmt.max_plans</code>	Static. Sets the maximum number of plans that may be cached by apg_plan_mgmt.
<code>apg_plan_mgmt.plan_retention</code>	Static. Maximum number of days since a plan was last_used before a plan will be automatically deleted.
<code>apg_plan_mgmt.unapproved_plan_threshold</code>	Estimated total plan restime below which an Unapproved plan will be executed.
<code>apg_plan_mgmt.use_plan_baseline</code>	Used only approved or fixed plans for managed statements.
<code>application_name</code>	Sets the application name to be reported in statistics and logs.
<code>authentication_timeout</code>	(s) Sets the maximum allowed time to complete client authentication.
<code>auto_explain.log_analyze</code>	Use EXPLAIN ANALYZE for plan logging.
<code>auto_explain.log_buffers</code>	Log buffers usage.
<code>auto_explain.log_format</code>	EXPLAIN format to be used for plan logging.
<code>auto_explain.log_min_duration</code>	Sets the minimum execution time above which plans will be logged.

Parameter name	Description
auto_explain.log_nested_statements	Logs nested statements.
auto_explain.log_timing	Collect timing data, not just row counts.
auto_explain.log_triggers	Include trigger statistics in plans.
auto_explain.log_verbose	Use EXPLAIN VERBOSE for plan logging.
auto_explain.sample_rate	Fraction of queries to process.
babelfishpg_tds.listen_addresses	Sets the host name or IP address(es) to listen TDS to.
babelfishpg_tds.tds_debug_level	Sets logging level in TDS, 0 disables logging
backend_flush_after	(8Kb) Number of pages after which previously performed writes are flushed to disk.
bytea_output	Sets the output format for bytea.
check_function_bodies	Check function bodies during CREATE FUNCTION.
client_min_messages	Sets the message levels that are sent to the client.
config_file	Static. Sets the servers main configuration file.
constraint_exclusion	Enables the planner to use constraints to optimize queries.
cpu_index_tuple_cost	Sets the planners estimate of the cost of processing each index entry during an index scan.
cpu_operator_cost	Sets the planners estimate of the cost of processing each operator or function call.
cpu_tuple_cost	Sets the planners estimate of the cost of processing each tuple (row).
cron.database_name	Static. Sets the database to store pg_cron metadata tables
cron.log_run	Static. Log all jobs runs into the job_run_details table
cron.log_statement	Static. Log all cron statements prior to execution.
cron.max_running_jobs	Static. Maximum number of jobs that can run concurrently.
cron.use_background_workers	Static. Enables background workers for pg_cron
cursor_tuple_fraction	Sets the planners estimate of the fraction of a cursors rows that will be retrieved.
db_user_namespace	Enables per-database user names.
deadlock_timeout	(ms) Sets the time to wait on a lock before checking for deadlock.
debug_pretty_print	Indents parse and plan tree displays.
debug_print_parse	Logs each querys parse tree.
debug_print_plan	Logs each querys execution plan.
debug_print_rewritten	Logs each querys rewritten parse tree.

Parameter name	Description
default_statistics_target	Sets the default statistics target.
default_transaction_deferrable	Sets the default deferrable status of new transactions.
default_transaction_isolation	Sets the transaction isolation level of each new transaction.
default_transaction_read_only	Sets the default read-only status of new transactions.
effective_cache_size	(8kB) Sets the planners assumption about the size of the disk cache.
effective_io_concurrency	Number of simultaneous requests that can be handled efficiently by the disk subsystem.
enable_bitmapscan	Enables the planners use of bitmap-scan plans.
enable_gathermerge	Enables the planners use of gather merge plans.
enable_hashagg	Enables the planners use of hashed aggregation plans.
enable_hashjoin	Enables the planners use of hash join plans.
enable_incremental_sort	Enables the planners use of incremental sort steps.
enable_indexonlyscan	Enables the planners use of index-only-scan plans.
enable_indexscan	Enables the planners use of index-scan plans.
enable_material	Enables the planners use of materialization.
enable_mergejoin	Enables the planners use of merge join plans.
enable_nestloop	Enables the planners use of nested-loop join plans.
enable_parallel_append	Enables the planners use of parallel append plans.
enable_parallel_hash	Enables the planners user of parallel hash plans.
enable_partition_pruning	Enable plan-time and run-time partition pruning.
enable_partitionwise_aggregation	Enables partitionwise aggregation and grouping.
enable_partitionwise_join	Enables partitionwise join.
enable_seqscan	Enables the planners use of sequential-scan plans.
enable_sort	Enables the planners use of explicit sort steps.
enable_tidscan	Enables the planners use of TID scan plans.
escape_string_warning	Warn about backslash escapes in ordinary string literals.
exit_on_error	Terminate session on any error.
force_parallel_mode	Forces use of parallel query facilities.
fromCollapse_limit	Sets the FROM-list size beyond which subqueries are not collapsed.
geqo	Enables genetic query optimization.
geqo_effort	GEQO: effort is used to set the default for other GEQO parameters.

Parameter name	Description
geqo_generations	GEQO: number of iterations of the algorithm.
geqo_pool_size	GEQO: number of individuals in the population.
geqo_seed	GEQO: seed for random path selection.
geqo_selection_bias	GEQO: selective pressure within the population.
geqo_threshold	Sets the threshold of FROM items beyond which GEQO is used.
gin_fuzzy_search_limit	Sets the maximum allowed result for exact search by GIN.
gin_pending_list_limit	(kB) Sets the maximum size of the pending list for GIN index.
hash_mem_multiplier	Multiple of work_mem to use for hash tables.
hba_file	Static. Sets the servers hba configuration file.
hot_standby_feedback	Allows feedback from a hot standby to the primary that will avoid query conflicts.
ident_file	Static. Sets the servers ident configuration file.
idle_in_transaction_session_timeout	(ms) Sets the maximum allowed duration of any idling transaction.
joinCollapse_limit	Sets the FROM-list size beyond which JOIN constructs are not flattened.
lc_messages	Sets the language in which messages are displayed.
listen_addresses	Static. Sets the host name or IP address(es) to listen to.
lo_compat_privileges	Enables backward compatibility mode for privilege checks on large objects.
log_connections	Logs each successful connection.
log_destination	Sets the destination for server log output.
log_directory	Sets the destination directory for log files.
log_disconnections	Logs end of a session, including duration.
log_duration	Logs the duration of each completed SQL statement.
log_error_verbosity	Sets the verbosity of logged messages.
log_executor_stats	Writes executor performance statistics to the server log.
log_file_mode	Sets the file permissions for log files.
log_filename	Sets the file name pattern for log files.
logging_collector	Static. Start a subprocess to capture stderr output and/or csvlogs into log files.
log_hostname	Logs the host name in the connection logs.
logical_decoding_work_mem	(kB) This much memory can be used by each internal reorder buffer before spilling to disk.

Parameter name	Description
<code>log_line_prefix</code>	Controls information prefixed to each log line.
<code>log_lock_waits</code>	Logs long lock waits.
<code>log_min_duration_sample</code>	(ms) Sets the minimum execution time above which a sample of statements will be logged. Sampling is determined by <code>log_statement_sample_rate</code> .
<code>log_min_duration_statement</code>	(ms) Sets the minimum execution time above which statements will be logged.
<code>log_min_error_statement</code>	Causes all statements generating error at or above this level to be logged.
<code>log_min_messages</code>	Sets the message levels that are logged.
<code>log_parameter_max_length</code>	(B) When logging statements, limit logged parameter values to first N bytes.
<code>log_parameter_max_length_on_error</code>	(B) When reporting an error, limit logged parameter values to first N bytes.
<code>log_parser_stats</code>	Writes parser performance statistics to the server log.
<code>log_planner_stats</code>	Writes planner performance statistics to the server log.
<code>log_replication_commands</code>	Logs each replication command.
<code>log_rotation_age</code>	(min) Automatic log file rotation will occur after N minutes.
<code>log_rotation_size</code>	(kB) Automatic log file rotation will occur after N kilobytes.
<code>log_statement</code>	Sets the type of statements logged.
<code>log_statement_sample_rate</code>	Fraction of statements exceeding <code>log_min_duration_sample</code> to be logged.
<code>log_statement_stats</code>	Writes cumulative performance statistics to the server log.
<code>log_temp_files</code>	(kB) Log the use of temporary files larger than this number of kilobytes.
<code>log_timezone</code>	Sets the time zone to use in log messages.
<code>log_truncate_on_rotation</code>	Truncate existing log files of same name during log rotation.
<code>maintenance_io_concurrency</code>	A variant of <code>effective_io_concurrency</code> that is used for maintenance work.
<code>maintenance_work_mem</code>	(kB) Sets the maximum memory to be used for maintenance operations.
<code>max_connections</code>	Static. Sets the maximum number of concurrent connections.
<code>max_files_per_process</code>	Static. Sets the maximum number of simultaneously open files for each server process.
<code>max_locks_per_transaction</code>	Static. Sets the maximum number of locks per transaction.

Parameter name	Description
<code>max_parallel_maintenance_workers</code>	Sets the maximum number of parallel processes per maintenance operation.
<code>max_parallel_workers</code>	Sets the maximum number of parallel workers than can be active at one time.
<code>max_parallel_workers_per_gather</code>	Sets the maximum number of parallel processes per executor node.
<code>max_pred_locks_per_page</code>	Sets the maximum number of predicate-locked tuples per page.
<code>max_pred_locks_per_relation</code>	Sets the maximum number of predicate-locked pages and tuples per relation.
<code>max_pred_locks_per_transaction</code>	Static. Sets the maximum number of predicate locks per transaction.
<code>max_slot_wal_keep_size</code>	(MB) Replication slots will be marked as failed, and segments released for deletion or recycling, if this much space is occupied by WAL on disk.
<code>max_stack_depth</code>	(kB) Sets the maximum stack depth, in kilobytes.
<code>max_standby_streaming_delay</code>	(ms) Sets the maximum delay before canceling queries when a hot standby server is processing streamed WAL data.
<code>max_worker_processes</code>	Static. Sets the maximum number of concurrent worker processes.
<code>min_parallel_index_scan_size</code>	(8kB) Sets the minimum amount of index data for a parallel scan.
<code>min_parallel_table_scan_size</code>	(8kB) Sets the minimum amount of table data for a parallel scan.
<code>old_snapshot_threshold</code>	Static. (min) Time before a snapshot is too old to read pages changed after the snapshot was taken.
<code>operator_precedence_warning</code>	Emit a warning for constructs that changed meaning since PostgreSQL 9.4.
<code>parallel_leader_participation</code>	Controls whether Gather and Gather Merge also run subplans.
<code>parallel_setup_cost</code>	Sets the planners estimate of the cost of starting up worker processes for parallel query.
<code>parallel_tuple_cost</code>	Sets the planners estimate of the cost of passing each tuple (row) from worker to master backend.
<code>pgaudit.log</code>	Specifies which classes of statements will be logged by session audit logging.
<code>pgaudit.log_catalog</code>	Specifies that session logging should be enabled in the case where all relations in a statement are in pg_catalog.
<code>pgaudit.log_level</code>	Specifies the log level that will be used for log entries.
<code>pgaudit.log_parameter</code>	Specifies that audit logging should include the parameters that were passed with the statement.
<code>pgaudit.log_relation</code>	Specifies whether session audit logging should create a separate log entry for each relation (TABLE, VIEW, etc.) referenced in a SELECT or DML statement.

Parameter name	Description
<code>pgaudit.log_statement_once</code>	Specifies whether logging will include the statement text and parameters with the first log entry for a statement/substatement combination or with every entry.
<code>pgaudit.role</code>	Specifies the master role to use for object audit logging.
<code>pg_bigm.enable_recheck</code>	It specifies whether to perform Recheck which is an internal process of full text search.
<code>pg_bigm.gin_key_limit</code>	It specifies the maximum number of 2-grams of the search keyword to be used for full text search.
<code>pg_bigm.last_update</code>	Static. It reports the last updated date of the pg_bigm module.
<code>pg_bigm.similarity_limit</code>	It specifies the minimum threshold used by the similarity search.
<code>pg_hint_plan.debug_print</code>	Logs results of hint parsing.
<code>pg_hint_plan.enable_hint</code>	Force planner to use plans specified in the hint comment preceding to the query.
<code>pg_hint_plan.enable_hint_table</code>	Forces planner to not get hint by using table lookups.
<code>pg_hint_plan.message_level</code>	Message level of debug messages.
<code>pg_hint_plan.parse_messages</code>	Message level of parse errors.
<code>pglogical.batch_inserts</code>	Static. Batch inserts if possible
<code>pglogical.conflict_log_level</code>	Sets log level used for logging resolved conflicts.
<code>pglogical.conflict_resolution</code>	Sets method used for conflict resolution for resolvable conflicts.
<code>pglogical.extra_connection_options</code>	option options to add to all peer node connections
<code>pglogical.synchronous_commit</code>	Static. pglogical specific synchronous commit value
<code>pglogical.use_spi</code>	Static. Use SPI instead of low-level API for applying changes
<code>pg_similarity.block_is_normalized</code>	Sets if the result value is normalized or not.
<code>pg_similarity.block_threshold</code>	Sets the threshold used by the Block similarity function.
<code>pg_similarity.block_tokenizer</code>	Sets the tokenizer for Block similarity function.
<code>pg_similarity.cosine_is_normalized</code>	Sets if the result value is normalized or not.
<code>pg_similarity.cosine_threshold</code>	Sets the threshold used by the Cosine similarity function.
<code>pg_similarity.cosine_tokenizer</code>	Sets the tokenizer for Cosine similarity function.
<code>pg_similarity.dice_is_normalized</code>	Sets if the result value is normalized or not.
<code>pg_similarity.dice_threshold</code>	Sets the threshold used by the Dice similarity measure.
<code>pg_similarity.dice_tokenizer</code>	Sets the tokenizer for Dice similarity measure.
<code>pg_similarity.euclidean_is_normalized</code>	Sets if the result value is normalized or not.
<code>pg_similarity.euclidean_threshold</code>	Sets the threshold used by the Euclidean similarity measure.

Parameter name	Description
pg_similarity.euclidean_tokenizer	Sets the tokenizer for Euclidean similarity measure.
pg_similarity.hamming_is_norm	Sets if the result value is normalized or not.
pg_similarity.hamming_threshold	Sets the threshold used by the Block similarity metric.
pg_similarity.jaccard_is_norm	Sets if the result value is normalized or not.
pg_similarity.jaccard_threshold	Sets the threshold used by the Jaccard similarity measure.
pg_similarity.jaccard_tokenizer	Sets the tokenizer for Jaccard similarity measure.
pg_similarity.jaro_is_norm	Sets if the result value is normalized or not.
pg_similarity.jaro_threshold	Sets the threshold used by the Jaro similarity measure.
pg_similarity.jarowinkler_is_norm	Sets if the result value is normalized or not.
pg_similarity.jarowinkler_threshold	Sets the threshold used by the Jarowinkler similarity measure.
pg_similarity.levenshtein_is_norm	Sets if the result value is normalized or not.
pg_similarity.levenshtein_threshold	Sets the threshold used by the Levenshtein similarity measure.
pg_similarity.matching_is_norm	Sets if the result value is normalized or not.
pg_similarity.matching_threshold	Sets the threshold used by the Matching Coefficient similarity measure.
pg_similarity.matching_tokenizer	Sets the tokenizer for Matching Coefficient similarity measure.
pg_similarity.mongeelkan_is_norm	Sets if the result value is normalized or not.
pg_similarity.mongeelkan_threshold	Sets the threshold used by the Monge-Elkan similarity measure.
pg_similarity.mongeelkan_tokenizer	Sets the tokenizer for Monge-Elkan similarity measure.
pg_similarity.nw_gap_penalty	Sets the gap penalty used by the Needleman-Wunsch similarity measure.
pg_similarity.nw_is_norm	Sets if the result value is normalized or not.
pg_similarity.nw_threshold	Sets the threshold used by the Needleman-Wunsch similarity measure.
pg_similarity.overlap_is_norm	Sets if the result value is normalized or not.
pg_similarity.overlap_threshold	Sets the threshold used by the Overlap Coefficient similarity measure.
pg_similarity.overlap_tokenizer	Sets the tokenizer for Overlap Coefficientsimilarity measure.
pg_similarity.qgram_is_norm	Sets if the result value is normalized or not.
pg_similarity.qgram_threshold	Sets the threshold used by the Q-Gram similarity measure.
pg_similarity.qgram_tokenizer	Sets the tokenizer for Q-Gram measure.
pg_similarity.swg_is_norm	Sets if the result value is normalized or not.

Parameter name	Description
<code>pg_similarity.swg_threshold</code>	Sets the threshold used by the Smith-Waterman-Gotoh similarity measure.
<code>pg_similarity.sw_is_normalized</code>	Sets if the result value is normalized or not.
<code>pg_similarity.sw_threshold</code>	Sets the threshold used by the Smith-Waterman similarity measure.
<code>pg_stat_statements.max</code>	Sets the maximum number of statements tracked by <code>pg_stat_statements</code> .
<code>pg_stat_statements.save</code>	Save <code>pg_stat_statements</code> statistics across server shutdowns.
<code>pg_stat_statements.track</code>	Selects which statements are tracked by <code>pg_stat_statements</code> .
<code>pg_stat_statements.track_plans</code>	Selects whether planning duration is tracked by <code>pg_stat_statements</code> .
<code>pg_stat_statements.track_utils</code>	Selects whether utility commands are tracked by <code>pg_stat_statements</code> .
<code>postgis.gdal_enabled_drivers</code>	Enable or disable GDAL drivers used with PostGIS in Postgres 9.3.5 and above.
<code>quote_all_identifiers</code>	When generating SQL fragments, quote all identifiers.
<code>random_page_cost</code>	Sets the planners estimate of the cost of a nonsequentially fetched disk page.
<code>rds.force_admin_logging_level</code>	See log messages for RDS admin user actions in customer databases.
<code>rds.log_retention_period</code>	Amazon RDS will delete PostgreSQL log that are older than N minutes.
<code>rds.pg_stat_ramdisk_size</code>	Size of the stats ramdisk in MB. A nonzero value will setup the ramdisk.
<code>rds.rds_superuser_reserved_slots</code>	Set the number of connection slots reserved for rds_superusers.
<code>rds.superuser_variables</code>	List of superuser-only variables for which we elevate rds_superuser modification statements.
<code>restart_after_crash</code>	Reinitialize server after backend crash.
<code>row_security</code>	Enable row security.
<code>search_path</code>	Sets the schema search order for names that are not schema-qualified.
<code>seq_page_cost</code>	Sets the planners estimate of the cost of a sequentially fetched disk page.
<code>session_replication_role</code>	Sets the sessions behavior for triggers and rewrite rules.
<code>shared_buffers</code>	(8kB) Sets the number of shared memory buffers used by the server.
<code>shared_preload_libraries</code>	Lists shared libraries to preload into server.
<code>ssl_ca_file</code>	Location of the SSL server authority file.

Parameter name	Description
<code>ssl_cert_file</code>	Location of the SSL server certificate file.
<code>ssl_key_file</code>	Location of the SSL server private key file
<code>standard_conforming_strings</code>	Causes ... strings to treat backslashes literally.
<code>statement_timeout</code>	(ms) Sets the maximum allowed duration of any statement.
<code>stats_temp_directory</code>	Writes temporary statistics files to the specified directory.
<code>superuser_reserved_connections</code>	Static. Sets the number of connection slots reserved for superusers.
<code>synchronize_seqscans</code>	Enable synchronized sequential scans.
<code>tcp_keepalives_count</code>	Maximum number of TCP keepalive retransmits.
<code>tcp_keepalives_idle</code>	(s) Time between issuing TCP keepalives.
<code>tcp_keepalives_interval</code>	(s) Time between TCP keepalive retransmits.
<code>temp_buffers</code>	(8kB) Sets the maximum number of temporary buffers used by each session.
<code>temp_file_limit</code>	Constrains the total amount disk space in kilobytes that a given PostgreSQL process can use for temporary files, excluding space used for explicit temporary tables
<code>temp_tablespaces</code>	Sets the tablespace(s) to use for temporary tables and sort files.
<code>track_activities</code>	Collects information about executing commands.
<code>track_activity_query_size</code>	Static. Sets the size reserved for pg_stat_activity.current_query, in bytes.
<code>track_counts</code>	Collects statistics on database activity.
<code>track_functions</code>	Collects function-level statistics on database activity.
<code>track_io_timing</code>	Collects timing statistics on database IO activity.
<code>transform_null_equals</code>	Treats expr=NULL as expr IS NULL.
<code>update_process_title</code>	Updates the process title to show the active SQL command.
<code>vacuum_cleanup_index_scale_factor</code>	Number of tuple inserts prior to index cleanup as a fraction of reltuples.
<code>wal_receiver_status_interval</code>	(s) Sets the maximum interval between WAL receiver status reports to the primary.
<code>work_mem</code>	(kB) Sets the maximum memory to be used for query workspaces.
<code>xmlbinary</code>	Sets how binary values are to be encoded in XML.
<code>xmloption</code>	Sets whether XML data in implicit parsing and serialization operations is to be considered as documents or content fragments.

Amazon Aurora PostgreSQL wait events

The following are common wait events for Aurora PostgreSQL.

Activity:ArchiverMain

The archiver process is waiting for activity.

Activity:AutoVacuumMain

The autovacuum launcher process is waiting for activity.

Activity:BgWriterHibernate

The background writer process is hibernating while waiting for activity.

Activity:BgWriterMain

The background writer process is waiting for activity.

Activity:CheckpointerMain

The checkpointer process is waiting for activity.

Activity:LogicalApplyMain

The logical replication apply process is waiting for activity.

Activity:LogicalLauncherMain

The logical replication launcher process is waiting for activity.

Activity:PgStatMain

The statistics collector process is waiting for activity.

Activity:RecoveryWalAll

A process is waiting for the write-ahead log (WAL) from a stream at recovery.

Activity:RecoveryWalStream

The startup process is waiting for the write-ahead log (WAL) to arrive during streaming recovery.

Activity:SysLoggerMain

The syslogger process is waiting for activity.

Activity:WalReceiverMain

The write-ahead log (WAL) receiver process is waiting for activity.

Activity:WalSenderMain

The write-ahead log (WAL) sender process is waiting for activity.

Activity:WalWriterMain

The write-ahead log (WAL) writer process is waiting for activity.

BufferPin:BufferPin

A process is waiting to acquire an exclusive pin on a buffer.

Client:GSSOpenServer

A process is waiting to read data from the client while establishing a Generic Security Service Application Program Interface (GSSAPI) session.

Client:ClientRead

A backend process is waiting to receive data from a PostgreSQL client. For more information, see [Client:ClientRead \(p. 1381\)](#).

Client:ClientWrite

A backend process is waiting to send more data to a PostgreSQL client. For more information, see [Client:ClientWrite \(p. 1384\)](#).

Client:LibPQWalReceiverConnect

A process is waiting in the write-ahead log (WAL) receiver to establish connection to remote server.

Client:LibPQWalReceiverReceive

A process is waiting in the write-ahead log (WAL) receiver to receive data from remote server.

Client:SSLOpenServer

A process is waiting for Secure Sockets Layer (SSL) while attempting connection.

Client:WalReceiverWaitStart

A process is waiting for startup process to send initial data for streaming replication.

Client:WalSenderWaitForWAL

A process is waiting for the write-ahead log (WAL) to be flushed in the WAL sender process.

Client:WalSenderWriteData

A process is waiting for any activity when processing replies from the write-ahead log (WAL) receiver in the WAL sender process.

CPU

A backend process is active in or is waiting for CPU. For more information, see [CPU \(p. 1385\)](#).

Extension:extension

A backend process is waiting for a condition defined by an extension or module.

IO:AuroraStorageLogAllocate

A session is allocating metadata and preparing for a transaction log write.

IO:BuffFileRead

When operations require more memory than the amount defined by working memory parameters, the engine creates temporary files on disk. This wait event occurs when operations read from the temporary files. For more information, see [IO:BuffFileRead and IO:BufFileWrite \(p. 1389\)](#).

IO:BuffFileWrite

When operations require more memory than the amount defined by working memory parameters, the engine creates temporary files on disk. This wait event occurs when operations write to the temporary files. For more information, see [IO:BuffFileRead and IO:BufFileWrite \(p. 1389\)](#).

IO:ControlFileRead

A process is waiting for a read from the pg_control file.

IO:ControlFileSync

A process is waiting for the pg_control file to reach durable storage.

IO:ControlFileSyncUpdate

A process is waiting for an update to the pg_control file to reach durable storage.

IO:ControlFileWrite

A process is waiting for a write to the pg_control file.

IO:ControlFileWriteUpdate

A process is waiting for a write to update the pg_control file.

IO:CopyFileRead

A process is waiting for a read during a file copy operation.

IO:CopyFileWrite

A process is waiting for a write during a file copy operation.

IO:DataFileExtend

A process is waiting for a relation data file to be extended.

IO:DataFileFlush

A process is waiting for a relation data file to reach durable storage.

IO:DataFileImmediateSync

A process is waiting for an immediate synchronization of a relation data file to durable storage.

IO:DataFilePrefetch

A process is waiting for an asynchronous prefetch from a relation data file.

IO:DataFileSync

A process is waiting for changes to a relation data file to reach durable storage.

IO:DataFileRead

A backend process tried to find a page in the shared buffers, didn't find it, and so read it from storage. For more information, see [IO:DataFileRead \(p. 1395\)](#).

IO:DataFileTruncate

A process is waiting for a relation data file to be truncated.

IO:DataFileWrite

A process is waiting for a write to a relation data file.

IO:DSMFillZeroWrite

A process is waiting to write zero bytes to a dynamic shared memory backing file.

IO:LockFileAddToDataDirRead

A process is waiting for a read while adding a line to the data directory lock file.

IO:LockFileAddToDataDirSync

A process is waiting for data to reach durable storage while adding a line to the data directory lock file.

IO:LockFileAddToDataDirWrite

A process is waiting for a write while adding a line to the data directory lock file.

IO:LockFileCreateRead

A process is waiting to read while creating the data directory lock file.

IO:LockFileCreateSync

A process is waiting for data to reach durable storage while creating the data directory lock file.

IO:LockFileCreateWrite

A process is waiting for a write while creating the data directory lock file.

IO:LockFileReCheckDataDirRead

A process is waiting for a read during recheck of the data directory lock file.

IO:LogicalRewriteCheckpointSync

A process is waiting for logical rewrite mappings to reach durable storage during a checkpoint.

IO:LogicalRewriteMappingSync

A process is waiting for mapping data to reach durable storage during a logical rewrite.

IO:LogicalRewriteMappingWrite

A process is waiting for a write of mapping data during a logical rewrite.

IO:LogicalRewriteSync

A process is waiting for logical rewrite mappings to reach durable storage.

IO:LogicalRewriteTruncate

A process is waiting for the truncation of mapping data during a logical rewrite.

IO:LogicalRewriteWrite

A process is waiting for a write of logical rewrite mappings.

IO:RelationMapRead

A process is waiting for a read of the relation map file.

IO:RelationMapSync

A process is waiting for the relation map file to reach durable storage.

IO:RelationMapWrite

A process is waiting for a write to the relation map file.

IO:ReorderBufferRead

A process is waiting for a read during reorder buffer management.

IO:ReorderBufferWrite

A process is waiting for a write during reorder buffer management.

IO:ReorderLogicalMappingRead

A process is waiting for a read of a logical mapping during reorder buffer management.

IO:ReplicationSlotRead

A process is waiting for a read from a replication slot control file.

IO:ReplicationSlotRestoreSync

A process is waiting for a replication slot control file to reach durable storage while restoring it to memory.

IO:ReplicationSlotSync

A process is waiting for a replication slot control file to reach durable storage.

IO:ReplicationSlotWrite

A process is waiting for a write to a replication slot control file.

IO:SLRUFlushSync

A process is waiting for segmented least-recently used (SLRU) data to reach durable storage during a checkpoint or database shutdown.

IO:SLRURead

A process is waiting for a read of a segmented least-recently used (SLRU) page.

IO:SLRUSync

A process is waiting for segmented least-recently used (SLRU) data to reach durable storage following a page write.

IO:SLRUWrite

A process is waiting for a write of a segmented least-recently used (SLRU) page.

IO:SnapbuildRead

A process is waiting for a read of a serialized historical catalog snapshot.

IO:SnapbuildSync

A process is waiting for a serialized historical catalog snapshot to reach durable storage.

IO:SnapbuildWrite

A process is waiting for a write of a serialized historical catalog snapshot.

IO:TimelineHistoryFileSync

A process is waiting for a timeline history file received through streaming replication to reach durable storage.

IO:TimelineHistoryFileWrite

A process is waiting for a write of a timeline history file received through streaming replication.

IO:TimelineHistoryRead

A process is waiting for a read of a timeline history file.

IO:TimelineHistorySync

A process is waiting for a newly created timeline history file to reach durable storage.

IO:TimelineHistoryWrite

A process is waiting for a write of a newly created timeline history file.

IO:TwophaseFileRead

A process is waiting for a read of a two phase state file.

IO:TwophaseFileSync

A process is waiting for a two phase state file to reach durable storage.

IO:TwophaseFileWrite

A process is waiting for a write of a two phase state file.

IO:WALBootstrapSync

A process is waiting for the write-ahead log (WAL) to reach durable storage during bootstrapping.

IO:WALBootstrapWrite

A process is waiting for a write of a write-ahead log (WAL) page during bootstrapping.

IO:WALCopyRead

A process is waiting for a read when creating a new write-ahead log (WAL) segment by copying an existing one.

IO:WALCopySync

A process is waiting for a new write-ahead log (WAL) segment created by copying an existing one to reach durable storage.

IO:WALCopyWrite

A process is waiting for a write when creating a new write-ahead log (WAL) segment by copying an existing one.

IO:WALInitSync

A process is waiting for a newly initialized write-ahead log (WAL) file to reach durable storage.

IO:WALInitWrite

A process is waiting for a write while initializing a new write-ahead log (WAL) file.

IO:WALRead

A process is waiting for a read from a write-ahead log (WAL) file.

IO:WALSenderTimelineHistoryRead

A process is waiting for a read from a timeline history file during a WAL sender timeline command.

IO:WALSync

A process is waiting for a write-ahead log (WAL) file to reach durable storage.

IO:WALSyncMethodAssign

A process is waiting for data to reach durable storage while assigning a new write-ahead log (WAL) sync method.

IO:WALWrite

A process is waiting for a write to a write-ahead log (WAL) file.

IO:XactSync

A backend process is waiting for the Aurora storage subsystem to acknowledge the commit of a regular transaction, or the commit or rollback of a prepared transaction. For more information, see [IO:XactSync \(p. 1401\)](#).

IPC:BackupWaitWalArchive

A process is waiting for write-ahead log (WAL) files required for a backup to be successfully archived.

IPC:BgWorkerShutdown

A process is waiting for a background worker to shut down.

IPC:BgWorkerStartup

A process is waiting for a background worker to start.

IPC:BtreePage

A process is waiting for the page number needed to continue a parallel B-tree scan to become available.

IPC:CheckpointDone

A process is waiting for a checkpoint to complete.

IPC:CheckpointStart

A process is waiting for a checkpoint to start.

IPC:ClogGroupUpdate

A process is waiting for the group leader to update the transaction status at a transaction's end.

pc:damrecordtxack

A backend process has generated a database activity streams event and is waiting for the event to become durable. For more information, see [ipc:damrecordtxack \(p. 1403\)](#).

IPC:ExecuteGather

A process is waiting for activity from a child process while executing a Gather plan node.

IPC:Hash/Batch/Allocating

A process is waiting for an elected parallel hash participant to allocate a hash table.

IPC:Hash/Batch/Electing

A process is electing a parallel hash participant to allocate a hash table.

IPC:Hash/Batch>Loading

A process is waiting for other parallel hash participants to finish loading a hash table.

IPC:Hash/Build/Allocating

A process is waiting for an elected parallel hash participant to allocate the initial hash table.

IPC:Hash/Build/Electing

A process is electing a parallel hash participant to allocate the initial hash table.

IPC:Hash/Build/HashingInner

A process is waiting for other parallel hash participants to finish hashing the inner relation.

IPC:Hash/Build/HashingOuter

A process is waiting for other parallel hash participants to finish partitioning the outer relation.

IPC:Hash/GrowBatches/Allocating

A process is waiting for an elected parallel hash participant to allocate more batches.

IPC:Hash/GrowBatches/Deciding

A process is electing a parallel hash participant to decide on future batch growth.

IPC:Hash/GrowBatches/Electing

A process is electing a parallel hash participant to allocate more batches.

IPC:Hash/GrowBatches/Finishing

A process is waiting for an elected parallel hash participant to decide on future batch growth.

IPC:Hash/GrowBatches/Repartitioning

A process is waiting for other parallel hash participants to finish repartitioning.

IPC:Hash/GrowBuckets/Allocating

A process is waiting for an elected parallel hash participant to finish allocating more buckets.

IPC:Hash/GrowBuckets/Electing

A process is electing a parallel hash participant to allocate more buckets.

IPC:Hash/GrowBuckets/Reinserting

A process is waiting for other parallel hash participants to finish inserting tuples into new buckets.

IPC:HashBatchAllocate

A process is waiting for an elected parallel hash participant to allocate a hash table.

IPC:HashBatchElect

A process is waiting to elect a parallel hash participant to allocate a hash table.

IPC:HashBatchLoad

A process is waiting for other parallel hash participants to finish loading a hash table.

IPC:HashBuildAllocate

A process is waiting for an elected parallel hash participant to allocate the initial hash table.

IPC:HashBuildElect

A process is waiting to elect a parallel hash participant to allocate the initial hash table.

IPC:HashBuildHashInner

A process is waiting for other parallel hash participants to finish hashing the inner relation.

IPC:HashBuildHashOuter

A process is waiting for other parallel hash participants to finish partitioning the outer relation.

IPC:HashGrowBatchesAllocate

A process is waiting for an elected parallel hash participant to allocate more batches.

IPC:HashGrowBatchesDecide

A process is waiting to elect a parallel hash participant to decide on future batch growth.

IPC:HashGrowBatchesElect

A process is waiting to elect a parallel hash participant to allocate more batches.

IPC:HashGrowBatchesFinish

A process is waiting for an elected parallel hash participant to decide on future batch growth.

IPC:HashGrowBatchesRepartition

A process is waiting for other parallel hash participants to finish repartitioning.

IPC:HashGrowBucketsAllocate

A process is waiting for an elected parallel hash participant to finish allocating more buckets.

IPC:HashGrowBucketsElect

A process is waiting to elect a parallel hash participant to allocate more buckets.

IPC:HashGrowBucketsReinsert

A process is waiting for other parallel hash participants to finish inserting tuples into new buckets.

IPC:LogicalSyncData

A process is waiting for a logical replication remote server to send data for initial table synchronization.

IPC:LogicalSyncStateChange

A process is waiting for a logical replication remote server to change state.

IPC:MessageQueueInternal

A process is waiting for another process to be attached to a shared message queue.

IPC:MessageQueuePutMessage

A process is waiting to write a protocol message to a shared message queue.

IPC:MessageQueueReceive

A process is waiting to receive bytes from a shared message queue.

IPC:MessageQueueSend

A process is waiting to send bytes to a shared message queue.

IPC:ParallelBitmapScan

A process is waiting for a parallel bitmap scan to become initialized.

IPC:ParallelCreateIndexScan

A process is waiting for parallel CREATE INDEX workers to finish a heap scan.

IPC:ParallelFinish

A process is waiting for parallel workers to finish computing.

IPC:ProcArrayGroupUpdate

A process is waiting for the group leader to clear the transaction ID at the end of a parallel operation.

IPC:ProcSignalBarrier

A process is waiting for a barrier event to be processed by all backends.

IPC:Promote

A process is waiting for standby promotion.

IPC:RecoveryConflictSnapshot

A process is waiting for recovery conflict resolution for a vacuum cleanup.

IPC:RecoveryConflictTablespace

A process is waiting for recovery conflict resolution for dropping a tablespace.

IPC:RecoveryPause

A process is waiting for recovery to be resumed.

IPC:ReplicationOriginDrop

A process is waiting for a replication origin to become inactive so it can be dropped.

IPC:ReplicationSlotDrop

A process is waiting for a replication slot to become inactive so it can be dropped.

IPC:SafeSnapshot

A process is waiting to obtain a valid snapshot for a READ ONLY DEFERRABLE transaction.

IPC:SyncRep

A process is waiting for confirmation from a remote server during synchronous replication.

IPC:XactGroupUpdate

A process is waiting for the group leader to update the transaction status at the end of a parallel operation.

Lock:advisory

A backend process requested an advisory lock and is waiting for it. For more information, see [Lock:advisory \(p. 1403\)](#).

Lock:extend

A backend process is waiting for a lock to be released so that it can extend a relation. This lock is needed because only one backend process can extend a relation at a time. For more information, see [Lock:extend \(p. 1405\)](#).

Lock:frozenid

A process is waiting to update pg_database.datfrozenxid and pg_database.datminmxid.

Lock:object

A process is waiting to get a lock on a nonrelation database object.

Lock:page

A process is waiting to get a lock on a page of a relation.

Lock:Relation

A backend process is waiting to acquire a lock on a relation that is locked by another transaction. For more information, see [Lock:Relation \(p. 1407\)](#).

Lock:spectoken

A process is waiting to get a speculative insertion lock.

Lock:speculative token

A process is waiting to acquire a speculative insertion lock.

Lock:transactionid

A transaction is waiting for a row-level lock. For more information, see [Lock:transactionid \(p. 1410\)](#).

Lock:tuple

A backend process is waiting to acquire a lock on a tuple while another backend process holds a conflicting lock on the same tuple. For more information, see [Lock:tuple \(p. 1413\)](#).

Lock:userlock

A process is waiting to get a user lock.

Lock:virtualxid

A process is waiting to get a virtual transaction ID lock.

Lwlock:AddinShmemInit

A process is waiting to manage an extension's space allocation in shared memory.

Lwlock:AddinShmemInitLock

A process is waiting to manage space allocation in shared memory.

Lwlock:async

A process is waiting for I/O on an async (notify) buffer.

Lwlock:AsyncCtlLock

A process is waiting to read or update a shared notification state.

Lwlock:AsyncQueueLock

A process is waiting to read or update notification messages.

Lwlock:AutoFile

A process is waiting to update the `postgresql.auto.conf` file.

Lwlock:AutoFileLock

A process is waiting to update the `postgresql.auto.conf` file.

Lwlock:Autovacuum

A process is waiting to read or update the current state of autovacuum workers.

Lwlock:AutovacuumLock

An autovacuum worker or launcher is waiting to update or read the current state of autovacuum workers.

Lwlock:AutovacuumSchedule

A process is waiting to ensure that a table selected for autovacuum still needs vacuuming.

Lwlock:AutovacuumScheduleLock

A process is waiting to ensure that the table it has selected for a vacuum still needs vacuuming.

Lwlock:BackendRandomLock

A process is waiting to generate a random number.

Lwlock:BackgroundWorker

A process is waiting to read or update background worker state.

Lwlock:BackgroundWorkerLock

A process is waiting to read or update the background worker state.

Lwlock:BtreeVacuum

A process is waiting to read or update vacuum-related information for a B-tree index.

Lwlock:BtreeVacuumLock

A process is waiting to read or update vacuum-related information for a B-tree index.

LWLock:buffer_content

A backend process is waiting to acquire a lightweight lock on the contents of a shared memory buffer. For more information, see [lwlock:buffer_content \(BufferContent\) \(p. 1415\)](#).

LWLock:buffer_mapping

A backend process is waiting to associate a data block with a buffer in the shared buffer pool. For more information, see [LWLock:buffer_mapping \(p. 1417\)](#).

LWLock:BufferIO

A backend process wants to read a page into shared memory. The process is waiting for other processes to finish their I/O for the page. For more information, see [LWLock:BufferIO \(p. 1418\)](#).

Lwlock:Checkpoint

A process is waiting to begin a checkpoint.

Lwlock:CheckpointLock

A process is waiting to perform checkpoint.

Lwlock:CheckpointerComm

A process is waiting to manage `fsync` requests.

Lwlock:CheckpointerCommLock

A process is waiting to manage `fsync` requests.

Lwlock:clog

A process is waiting for I/O on a clog (transaction status) buffer.

Lwlock:CLogControlLock

A process is waiting to read or update transaction status.

Lwlock:CLogTruncationLock

A process is waiting to run `txid_status` or update the oldest transaction ID available to it.

Lwlock:commit_timestamp

A process is waiting for I/O on a commit timestamp buffer.

Lwlock:CommitTs

A process is waiting to read or update the last value set for a transaction commit timestamp.

Lwlock:CommitTsBuffer

A process is waiting for I/O on a segmented least-recently used (SLRU) buffer for a commit timestamp.

Lwlock:CommitTsControlLock

A process is waiting to read or update transaction commit timestamps.

Lwlock:CommitTsLock

A process is waiting to read or update the last value set for the transaction timestamp.

Lwlock:CommitTsSLRU

A process is waiting to access the segmented least-recently used (SLRU) cache for a commit timestamp.

Lwlock:ControlFile

A process is waiting to read or update the `pg_control` file or create a new write-ahead log (WAL) file.

Lwlock:ControlFileLock

A process is waiting to read or update the control file or creation of a new write-ahead log (WAL) file.

Lwlock:DynamicSharedMemoryControl

A process is waiting to read or update dynamic shared memory allocation information.

Lwlock:DynamicSharedMemoryControlLock

A process is waiting to read or update the dynamic shared memory state.

LWLock:lock_manager

A backend process is waiting to add or examine locks for backend processes. Or it's waiting to join or exit a locking group that is used by parallel query. For more information, see [LWLock:lock_manager \(p. 1420\)](#).

Lwlock:LockFastPath

A process is waiting to read or update a process's fast-path lock information.

Lwlock:LogicalRepWorker

A process is waiting to read or update the state of logical replication workers.

Lwlock:LogicalRepWorkerLock

A process is waiting for an action on a logical replication worker to finish.

Lwlock:multixact_member

A process is waiting for I/O on a multixact_member buffer.

Lwlock:multixact_offset

A process is waiting for I/O on a multixact offset buffer.

Lwlock:MultiXactGen

A process is waiting to read or update shared multixact state.

Lwlock:MultiXactGenLock

A process is waiting to read or update a shared multixact state.

Lwlock:MultiXactMemberBuffer

A process is waiting for I/O on a segmented least-recently used (SLRU) buffer for a multixact member.

Lwlock:MultiXactMemberControlLock

A process is waiting to read or update multixact member mappings.

Lwlock:MultiXactMemberSLRU

A process is waiting to access the segmented least-recently used (SLRU) cache for a multixact member.

Lwlock:MultiXactOffsetBuffer

A process is waiting for I/O on a segmented least-recently used (SLRU) buffer for a multixact offset.

Lwlock:MultiXactOffsetControlLock

A process is waiting to read or update multixact offset mappings.

Lwlock:MultiXactOffsetSLRU

A process is waiting to access the segmented least-recently used (SLRU) cache for a multixact offset.

Lwlock:MultiXactTruncation

A process is waiting to read or truncate multixact information.

Lwlock:MultiXactTruncationLock

A process is waiting to read or truncate multixact information.

Lwlock:NotifyBuffer

A process is waiting for I/O on the segmented least-recently used (SLRU) buffer for a NOTIFY message.

Lwlock:NotifyQueue

A process is waiting to read or update NOTIFY messages.

Lwlock:NotifyQueueTail

A process is waiting to update a limit on NOTIFY message storage.

Lwlock:NotifyQueueTailLock

A process is waiting to update limit on notification message storage.

Lwlock:NotifySLRU

A process is waiting to access the segmented least-recently used (SLRU) cache for a NOTIFY message.

Lwlock:OidGen

A process is waiting to allocate a new object ID (OID).

Lwlock:OidGenLock

A process is waiting to allocate or assign an object ID (OID).

Lwlock:oldserxid

A process is waiting for I/O on an oldserxid buffer.

Lwlock:OldSerXidLock

A process is waiting to read or record conflicting serializable transactions.

Lwlock:OldSnapshotTimeMap

A process is waiting to read or update old snapshot control information.

Lwlock:OldSnapshotTimeMapLock

A process is waiting to read or update old snapshot control information.

Lwlock:parallel_append

A process is waiting to choose the next subplan during parallel append plan execution.

Lwlock:parallel_hash_join

A process is waiting to allocate or exchange a chunk of memory or update counters during a parallel hash plan execution.

Lwlock:parallel_query_dsa

A process is waiting for a lock on dynamic shared memory allocation for a parallel query.

Lwlock:ParallelAppend

A process is waiting to choose the next subplan during parallel append plan execution.

Lwlock:ParallelHashJoin

A process is waiting to synchronize workers during plan execution for a parallel hash join.

Lwlock:ParallelQueryDSA

A process is waiting for dynamic shared memory allocation for a parallel query.

Lwlock:PerSessionDSA

A process is waiting for dynamic shared memory allocation for a parallel query.

Lwlock:PerSessionRecordType

A process is waiting to access a parallel query's information about composite types.

Lwlock:PerSessionRecordTypmod

A process is waiting to access a parallel query's information about type modifiers that identify anonymous record types.

Lwlock:PerXactPredicateList

A process is waiting to access the list of predicate locks held by the current serializable transaction during a parallel query.

Lwlock:predicate_lock_manager

A process is waiting to add or examine predicate lock information.

Lwlock:PredicateLockManager

A process is waiting to access predicate lock information used by serializable transactions.

Lwlock:proc

A process is waiting to read or update the fast-path lock information.

Lwlock:ProcArray

A process is waiting to access the shared per-process data structures (typically, to get a snapshot or report a session's transaction ID).

Lwlock:ProcArrayLock

A process is waiting to get a snapshot or clearing a transaction Id at a transaction's end.

Lwlock:RelationMapping

A process is waiting to read or update a `pg_filenode.map` file (used to track the file-node assignments of certain system catalogs).

Lwlock:RelationMappingLock

A process is waiting to update the relation map file used to store catalog-to-file-node mapping.

Lwlock:RelCacheInit

A process is waiting to read or update a `pg_internal.init` file (a relation cache initialization file).

Lwlock:RelCacheInitLock

A process is waiting to read or write a relation cache initialization file.

Lwlock:replication_origin

A process is waiting to read or update the replication progress.

Lwlock:replication_slot_io

A process is waiting for I/O on a replication slot.

Lwlock:ReplicationOrigin

A process is waiting to create, drop, or use a replication origin.

Lwlock:ReplicationOriginLock

A process is waiting to set up, drop, or use a replication origin.

Lwlock:ReplicationOriginState

A process is waiting to read or update the progress of one replication origin.

Lwlock:ReplicationSlotAllocation

A process is waiting to allocate or free a replication slot.

Lwlock:ReplicationSlotAllocationLock

A process is waiting to allocate or free a replication slot.

Lwlock:ReplicationSlotControl

A process is waiting to read or update a replication slot state.

Lwlock:ReplicationSlotControlLock

A process is waiting to read or update the replication slot state.

Lwlock:ReplicationSlotIO

A process is waiting for I/O on a replication slot.

Lwlock:SerialBuffer

A process is waiting for I/O on a segmented least-recently used (SLRU) buffer for a serializable transaction conflict.

Lwlock:SerializableFinishedList

A process is waiting to access the list of finished serializable transactions.

Lwlock:SerializableFinishedListLock

A process is waiting to access the list of finished serializable transactions.

Lwlock:SerializablePredicateList

A process is waiting to access the list of predicate locks held by serializable transactions.

Lwlock:SerializablePredicateLockListLock

A process is waiting to perform an operation on a list of locks held by serializable transactions.

Lwlock:SerializableXactHash

A process is waiting to read or update information about serializable transactions.

Lwlock:SerializableXactHashLock

A process is waiting to retrieve or store information about serializable transactions.

Lwlock:SerialSLRU

A process is waiting to access the segmented least-recently used (SLRU) cache for a serializable transaction conflict.

Lwlock:SharedTidBitmap

A process is waiting to access a shared tuple identifier (TID) bitmap during a parallel bitmap index scan.

Lwlock:SharedTupleStore

A process is waiting to access a shared tuple store during a parallel query.

Lwlock:ShmemIndex

A process is waiting to find or allocate space in shared memory.

Lwlock:ShmemIndexLock

A process is waiting to find or allocate space in shared memory.

Lwlock:SInvalRead

A process is waiting to retrieve messages from the shared catalog invalidation queue.

Lwlock:SInvalReadLock

A process is waiting to retrieve or remove messages from a shared invalidation queue.

Lwlock:SInvalWrite

A process is waiting to add a message to the shared catalog invalidation queue.

Lwlock:SInvalWriteLock

A process is waiting to add a message in a shared invalidation queue.

Lwlock:subtrans

A process is waiting for I/O on a subtransaction buffer.

Lwlock:SubtransBuffer

A process is waiting for I/O on a segmented least-recently used (SLRU) buffer for a subtransaction.

Lwlock:SubtransControlLock

A process is waiting to read or update subtransaction information.

Lwlock:SubtransSLRU

A process is waiting to access the segmented least-recently used (SLRU) cache for a subtransaction.

Lwlock:SyncRep

A process is waiting to read or update information about the state of synchronous replication.

Lwlock:SyncRepLock

A process is waiting to read or update information about synchronous replicas.

Lwlock:SyncScan

A process is waiting to select the starting location of a synchronized table scan.

Lwlock:SyncScanLock

A process is waiting to get the start location of a scan on a table for synchronized scans.

Lwlock:TablespaceCreate

A process is waiting to create or drop a tablespace.

Lwlock:TablespaceCreateLock

A process is waiting to create or drop the tablespace.

Lwlock:tbm

A process is waiting for a shared iterator lock on a tree bitmap (TBM).

Lwlock:TwoPhaseState

A process is waiting to read or update the state of prepared transactions.

Lwlock:TwoPhaseStateLock

A process is waiting to read or update the state of prepared transactions.

Lwlock:wal_insert

A process is waiting to insert the write-ahead log (WAL) into a memory buffer.

Lwlock:WALBufMapping

A process is waiting to replace a page in write-ahead log (WAL) buffers.

Lwlock:WALBufMappingLock

A process is waiting to replace a page in write-ahead log (WAL) buffers.

Lwlock:WALInsert

A process is waiting to insert write-ahead log (WAL) data into a memory buffer.

Lwlock:WALWrite

A process is waiting for write-ahead log (WAL) buffers to be written to disk.

Lwlock:WALWriteLock

A process is waiting for write-ahead log (WAL) buffers to be written to disk.

Lwlock:WrapLimitsVacuum

A process is waiting to update limits on transaction ID and multixact consumption.

Lwlock:WrapLimitsVacuumLock

A process is waiting to update limits on transaction ID and multixact consumption.

Lwlock:XactBuffer

A process is waiting for I/O on a segmented least-recently used (SLRU) buffer for a transaction status.

Lwlock:XactSLRU

A process is waiting to access the segmented least-recently used (SLRU) cache for a transaction status.

Lwlock:XactTruncation

A process is waiting to run pg_xact_status or update the oldest transaction ID available to it.

Lwlock:XidGen

A process is waiting to allocate a new transaction ID.

Lwlock:XidGenLock

A process is waiting to allocate or assign a transaction ID.

Timeout:BaseBackupThrottle

A process is waiting during base backup when throttling activity.

Timeout:PgSleep

A backend process has called the pg_sleep function and is waiting for the sleep timeout to expire. For more information, see [Timeout:PgSleep \(p. 1423\)](#).

Timeout:RecoveryApplyDelay

A process is waiting to apply write-ahead log (WAL) during recovery because of a delay setting.

Timeout:RecoveryRetrieveRetryInterval

A process is waiting during recovery when write-ahead log (WAL) data is not available from any source (pg_wal, archive, or stream).

Timeout:VacuumDelay

A process is waiting in a cost-based vacuum delay point.

For a complete list of PostgreSQL wait events, see [PostgreSQL wait-event table](#) in the PostgreSQL documentation.

Aurora PostgreSQL functions reference

Following, you can find a list of Aurora PostgreSQL functions that are available for your Aurora DB clusters that run the Aurora PostgreSQL-Compatible Edition DB engine. These Aurora PostgreSQL functions are in addition to the standard PostgreSQL functions. For more information about standard PostgreSQL functions, see [PostgreSQL–Functions and Operators](#).

Overview

You can use the following functions for Amazon RDS DB instances running Aurora PostgreSQL:

- [aurora_list_builtin](#) (p. 1588)
- [aurora_stat_dml_activity](#) (p. 1589)
- [aurora_stat_get_db_commit_latency](#) (p. 1592)
- [aurora_stat_system_waits](#) (p. 1594)
- [aurora_stat_wait_event](#) (p. 1595)
- [aurora_stat_wait_type](#) (p. 1597)

aurora_list_builtin

Lists all available Aurora PostgreSQL built-in functions, along with brief descriptions and function details.

Syntax

```
aurora_list_builtin()
```

Return type

SETOF record

Arguments

None

Examples

The following example shows results from calling the `aurora_list_builtin` function.

```
=> SELECT *
   FROM aurora_list_builtin();
```

types	Name	Result data type			Argument data	
	Description	Type	Volatility	Parallel	Security	
aurora_version	text				invoker	Amazon Aurora
PostgreSQL-Compatible Edition version string	func	stable	safe			
aurora_stat_wait_type	SETOF record		OUT type_id smallint, OUT type_name			
text	func	volatile	restricted	invoker	Lists all supported	wait types
aurora_stat_wait_event	SETOF record		OUT type_id smallint, OUT event_id			
integer, OUT event_na.	func	volatile	restricted	invoker	Lists all supported	wait events
aurora_list_builtin	SETOF record		.me text			
type" text, OUT "Argum.	func	stable	safe	invoker	Lists all Aurora	built-in functions
text, OUT "Volatility" .			.ent data types" text, OUT "Type"			
"Security" text, OUT "Des.			.text, OUT "Parallel" text, OUT			
		.cription" text				
.						
.						
.						
aurora_stat_file	SETOF record		OUT filename text, OUT			
allocated_bytes bigint, OUT used_.	func	stable	safe	invoker	Lists all	files present in Aurora storage
aurora_stat_get_db_commit_latency	bigint		.bytes bigint			
in microsecs	func	stable	restricted	invoker	oid	Per DB commit latency

[aurora_stat_dml_activity](#)

Reports cumulative activity for each type of data manipulation language (DML) operation on a database in an Aurora PostgreSQL cluster.

Syntax

```
aurora_stat_dml_activity(database_oid)
```

Return type

SETOF record

Arguments

database_oid

The object ID (OID) of the database in the Aurora PostgreSQL cluster.

Usage notes

The `aurora_stat_dml_activity` function is only available with Aurora PostgreSQL release 3.1 compatible with PostgreSQL engine 11.6 and later.

Use this function on Aurora PostgreSQL clusters with a large number of databases to identify which databases have more or slower DML activity, or both.

The `aurora_stat_dml_activity` function returns the number of times the operations ran and the cumulative latency in microseconds for SELECT, INSERT, UPDATE, and DELETE operations. The report includes only successful DML operations.

You can reset this statistic by using the PostgreSQL statistics access function `pg_stat_reset`. You can check the last time this statistic was reset by using the `pg_stat_get_db_stat_reset_time` function. For more information about PostgreSQL statistics access functions, see [The Statistics Collector](#) in the PostgreSQL documentation.

Examples

The following example shows how to report DML activity statistics for the connected database.

```
-- Define the oid variable from connected database by using \gset
=> SELECT oid,
        datname
   FROM pg_database
 WHERE datname=(select current_database()) \gset

=> SELECT *
      FROM aurora_stat_dml_activity(:oid);

 select_count | select_latency_microsecs | insert_count | insert_latency_microsecs |
 update_count | update_latency_microsecs | delete_count | delete_latency_microsecs
-----+-----+-----+-----+
-----+-----+-----+-----+
 178957 |           66684115 |     171065 |          28876649 |
 519538 |           1454579206167 |       1 |             53027

-- Showing the same results with expanded display on
=> SELECT *
      FROM aurora_stat_dml_activity(:oid);
-[ RECORD 1 ]-----+
select_count | 178957
select_latency_microsecs | 66684115
insert_count | 171065
insert_latency_microsecs | 28876649
update_count | 519538
update_latency_microsecs | 1454579206167
delete_count | 1
delete_latency_microsecs | 53027
```

The following example shows DML activity statistics for all databases in the Aurora PostgreSQL cluster. This cluster has two databases, `postgres` and `mydb`. The comma-separated list corresponds with the `select_count`, `select_latency_microsecs`, `insert_count`, `insert_latency_microsecs`, `update_count`, `update_latency_microsecs`, `delete_count`, and `delete_latency_microsecs` fields.

Aurora PostgreSQL creates and uses a system database named `rdsadmin` to support administrative operations such as backups, restores, health checks, replication, and so on. These DML operations have no impact on your Aurora PostgreSQL cluster.

```
=> SELECT oid,
       datname,
       aurora_stat_dml_activity(oid)
  FROM pg_database;
   oid | datname | aurora_stat_dml_activity
-----+-----+
 14006 | template0 | (,,,...)
 16384 | rdsadmin | (2346623,1211703821,4297518,817184554,0,0,0,0)
     1 | template1 | (,,,...)
 14007 | postgres  | (178961,66716329,171065,28876649,519538,1454579206167,1,53027)
 16401 | mydb     | (200246,64302436,200036,107101855,600000,83659417514,0,0)
```

The following example shows DML activity statistics for all databases, organized in columns for better readability.

```
SELECT db.datname,
       BTRIM(SPLIT_PART(db.asdmla::TEXT, ',', 1), '') AS select_count,
       BTRIM(SPLIT_PART(db.asdmla::TEXT, ',', 2), '') AS select_latency_microsecs,
       BTRIM(SPLIT_PART(db.asdmla::TEXT, ',', 3), '') AS insert_count,
       BTRIM(SPLIT_PART(db.asdmla::TEXT, ',', 4), '') AS insert_latency_microsecs,
       BTRIM(SPLIT_PART(db.asdmla::TEXT, ',', 5), '') AS update_count,
       BTRIM(SPLIT_PART(db.asdmla::TEXT, ',', 6), '') AS update_latency_microsecs,
       BTRIM(SPLIT_PART(db.asdmla::TEXT, ',', 7), '') AS delete_count,
       BTRIM(SPLIT_PART(db.asdmla::TEXT, ',', 8), '') AS delete_latency_microsecs
  FROM (SELECT datname,
              aurora_stat_dml_activity(oid) AS asdmla
         FROM pg_database
        ) AS db;

   datname | select_count | select_latency_microsecs | insert_count |
insert_latency_microsecs | update_count | update_latency_microsecs | delete_count |
delete_latency_microsecs
-----+-----+-----+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+-----+-----+
+-----+
template0 |           |           |           |           |           |           |
rdsadmin  | 4206523 | 2478812333 |           | 7009414 |           | 1338482258
| 0       | 0          |           |           | 0          |           | 0
template1 |           |           |           |           |           |           |
fault_test | 66       | 452099    |           | 0          |           | 0
| 0       | 0          |           |           | 0          |           | 0
db_access_test | 1       | 5982      |           | 0          |           | 0
| 0       | 0          |           |           | 0          |           | 0
postgres   | 42035    | 95179203 |           | 5752      |           | 2678832898
| 21157   | 441883182488 |           | 2          | 1520      |           | 0
mydb      | 71       | 453514    |           | 1          |           | 152
| 1       | 190       |           |           | 1          |           |
```

The following example shows the average cumulative latency (cumulative latency divided by count) for each DML operation for the database with the OID 16401.

```
=> SELECT select_count,
       select_latency_microsecs,
       select_latency_microsecs/NULLIF(select_count,0) select_latency_per_exec,
       insert_count,
       insert_latency_microsecs,
       insert_latency_microsecs/NULLIF(insert_count,0) insert_latency_per_exec,
       update_count,
       update_latency_microsecs,
       update_latency_microsecs/NULLIF(update_count,0) update_latency_per_exec,
```

```

    delete_count,
    delete_latency_microsecs,
    delete_latency_microsecs/NULLIF(delete_count,0) delete_latency_per_exec
  FROM aurora_stat_dml_activity(16401);

-[ RECORD 1 ]-----+
select_count | 451312
select_latency_microsecs | 80205857
select_latency_per_exec | 177
insert_count | 451001
insert_latency_microsecs | 123667646
insert_latency_per_exec | 274
update_count | 1353067
update_latency_microsecs | 200900695615
update_latency_per_exec | 148478
delete_count | 12
delete_latency_microsecs | 448
delete_latency_per_exec | 37

```

aurora_stat_get_db_commit_latency

Gets the cumulative commit latency in microseconds for Aurora PostgreSQL databases. *Commit latency* is measured as the time between when a client submits a commit request and when it receives the commit acknowledgement.

Syntax

```
aurora_stat_get_db_commit_latency(database_oid)
```

Return type

SETOF record

Arguments

database_oid

The object ID (OID) of the Aurora PostgreSQL database.

Usage notes

Amazon CloudWatch uses this function to calculate the average commit latency. For more information about Amazon CloudWatch metrics and how to troubleshoot high commit latency, see [Viewing metrics in the Amazon RDS console \(p. 563\)](#) and [Making better decisions about Amazon RDS with Amazon CloudWatch metrics](#).

You can reset this statistic by using the PostgreSQL statistics access function `pg_stat_reset`. You can check the last time this statistic was reset by using the `pg_stat_get_db_stat_reset_time` function. For more information about PostgreSQL statistics access functions, see [The Statistics Collector](#) in the PostgreSQL documentation.

Examples

The following example gets the cumulative commit latency for each database in the `pg_database` cluster.

```
=> SELECT oid,
```

```

        datname,
        aurora_stat_get_db_commit_latency(oid)
FROM pg_database;

    oid |      datname      | aurora_stat_get_db_commit_latency
-----+-----+-----+
 14006 | template0      | 0
 16384 | rdsadmin       | 654387789
     1 | template1       | 0
 16401 | mydb           | 229556
 69768 | postgres         | 22011

```

The following example gets the cumulative commit latency for the currently connected database. Before calling the `aurora_stat_get_db_commit_latency` function, the example first uses `\gset` to define a variable for the `oid` argument and sets its value from the connected database.

```

--Get the oid value from the connected database before calling
aurora_stat_get_db_commit_latency

=> SELECT oid
     FROM pg_database
     WHERE datname=(SELECT current_database()) \gset

=> SELECT *
     FROM aurora_stat_get_db_commit_latency(:oid);

aurora_stat_get_db_commit_latency
-----
1424279160

```

The following example gets the cumulative commit latency for the `mydb` database in the `pg_database` cluster. Then, it resets this statistic by using the `pg_stat_reset` function and shows the results. Finally, it uses the `pg_stat_get_db_stat_reset_time` function to check the last time this statistic was reset.

```

=> SELECT oid,
        datname,
        aurora_stat_get_db_commit_latency(oid)
     FROM pg_database
    WHERE datname = 'mydb';

    oid |      datname      | aurora_stat_get_db_commit_latency
-----+-----+-----+
 16427 | mydb           | 3320370

=> SELECT pg_stat_reset();

pg_stat_reset
-----

=> SELECT oid,
        datname,
        aurora_stat_get_db_commit_latency(oid)
     FROM pg_database
    WHERE datname = 'mydb';

    oid |      datname      | aurora_stat_get_db_commit_latency
-----+-----+-----+
 16427 | mydb           | 6

=> SELECT *

```

```
FROM pg_stat_get_db_stat_reset_time(16427);

pg_stat_get_db_stat_reset_time
-----
2021-04-29 21:36:15.707399+00
```

aurora_stat_system_waits

Reports wait event information for the Aurora PostgreSQL DB instance.

Syntax

```
aurora_stat_system_waits()
```

Return type

SETOF record

Arguments

None

Usage notes

This function returns the cumulative number of waits and cumulative wait time for each wait event generated by the DB instance that you're currently connected to.

The returned recordset includes the following fields:

- `type_id` – The ID of the type of wait event.
- `event_id` – The ID of the wait event.
- `waits` – The number of times the wait event occurred.
- `wait_time` – The total amount of time in microseconds spent waiting for this event.

Statistics returned by this function are reset when a DB instance restarts.

Examples

The following example shows results from calling the `aurora_stat_system_waits` function.

```
=> SELECT *
   FROM aurora_stat_system_waits();

 type_id | event_id |    waits    |   wait_time
-----+-----+-----+-----+
      1 | 16777219 |      11 |     12864
      1 | 16777220 |      501 |    174473
      1 | 16777270 |    53171 |  23641847
      1 | 16777271 |       23 |    319668
      1 | 16777274 |       60 |     12759
      .
      .
      .
      10 | 167772231 |  204596 | 790945212
      10 | 167772232 |        2 |      47729
```

10 167772234	1	888
10 167772235	2	64

The following example shows how you can use this function together with `aurora_stat_wait_event` and `aurora_stat_wait_type` to produce more readable results.

```
=> SELECT type_name,
           event_name,
           waits,
           wait_time
      FROM aurora_stat_system_waits()
NATURAL JOIN aurora_stat_wait_event()
NATURAL JOIN aurora_stat_wait_type();

   type_name |     event_name    |   waits   |   wait_time
-----+-----+-----+-----+
    LWLock  | XidGenLock      |      11 |      12864
    LWLock  | ProcArrayLock   |     501 |     174473
    LWLock  | buffer_content  |  53171 |  23641847
    LWLock  | rdsutils         |       2 |      12764
    Lock    | tuple            |  75686 | 2033956052
    Lock    | transactionid   | 1765147 | 47267583409
Activity | AutoVacuumMain | 136868 | 56305604538
Activity | BgWriterHibernate | 7486 | 55266949471
Activity | BgWriterMain    | 7487 | 1508909964
.
.
.
    IO      | SLRURead        |       3 |      11756
    IO      | WALWrite         | 52544463 | 388850428
    IO      | XactSync         | 187073 | 597041642
    IO      | ClogRead         |       2 |      47729
    IO      | OutboundCtrlRead |       1 |      888
    IO      | OutboundCtrlWrite|       2 |      64
```

aurora_stat_wait_event

Lists all supported wait events for Aurora PostgreSQL. For information about Aurora PostgreSQL wait events, see [Amazon Aurora PostgreSQL wait events \(p. 1570\)](#).

Syntax

```
aurora_stat_wait_event()
```

Return type

SETOF record

Arguments

None

Usage notes

To see event names with event types instead of IDs, use this function together with other functions such as `aurora_stat_wait_type` and `aurora_stat_system_waits`. Wait event names returned by this function are the same as those returned by the `aurora_wait_report` function.

Examples

The following example shows results from calling the `aurora_stat_wait_event` function.

```
=> SELECT *
   FROM aurora_stat_wait_event();

  type_id | event_id |           event_name
-----+-----+-----+
      1 | 16777216 | <unassigned:0>
      1 | 16777217 | ShmemIndexLock
      1 | 16777218 | OidGenLock
      1 | 16777219 | XidGenLock
      .
      .
      .
      9 | 150994945 | PgSleep
      9 | 150994946 | RecoveryApplyDelay
     10 | 167772160 | BufFileRead
     10 | 167772161 | BufFileWrite
     10 | 167772162 | ControlFileRead
      .
      .
      .
     10 | 167772226 | WALInitWrite
     10 | 167772227 | WALRead
     10 | 167772228 | WALSync
     10 | 167772229 | WALSyncMethodAssign
     10 | 167772230 | WALWrite
     10 | 167772231 | XactSync
      .
      .
      .
    11 | 184549377 | LsnAllocate
```

The following example joins `aurora_stat_wait_type` and `aurora_stat_wait_event` to return type names and event names for improved readability.

```
=> SELECT *
   FROM aurora_stat_wait_type() t
   JOIN aurora_stat_wait_event() e
     ON t.type_id = e.type_id;

  type_id | type_name | type_id | event_id |           event_name
-----+-----+-----+-----+
      1 | LWLock    |      1 | 16777216 | <unassigned:0>
      1 | LWLock    |      1 | 16777217 | ShmemIndexLock
      1 | LWLock    |      1 | 16777218 | OidGenLock
      1 | LWLock    |      1 | 16777219 | XidGenLock
      1 | LWLock    |      1 | 16777220 | ProcArrayLock
      .
      .
      .
      3 | Lock      |      3 | 50331648 | relation
      3 | Lock      |      3 | 50331649 | extend
      3 | Lock      |      3 | 50331650 | page
      3 | Lock      |      3 | 50331651 | tuple
      .
      .
      .
     10 | IO         |     10 | 167772214 | TimelineHistorySync
     10 | IO         |     10 | 167772215 | TimelineHistoryWrite
     10 | IO         |     10 | 167772216 | TwophaseFileRead
```

10 IO		10 167772217 TwophaseFileSync
.	.	.
11 LSN		11 184549376 LsnDurable

aurora_stat_wait_type

Lists all supported wait types for Aurora PostgreSQL.

Syntax

```
aurora_stat_wait_type()
```

Return type

SETOF record

Arguments

None

Usage notes

To see wait event names with wait event types instead of IDs, use this function together with other functions such as `aurora_stat_wait_event` and `aurora_stat_system_waits`. Wait type names returned by this function are the same as those returned by the `aurora_wait_report` function.

Examples

The following example shows results from calling the `aurora_stat_wait_type` function.

```
=> SELECT *
   FROM aurora_stat_wait_type();

 type_id | type_name
-----+-----
    1 | LWLock
    3 | Lock
    4 | BufferPin
    5 | Activity
    6 | Client
    7 | Extension
    8 | IPC
    9 | Timeout
   10 | IO
   11 | LSN
```

Amazon Aurora PostgreSQL updates

Following, you can find information about Amazon Aurora PostgreSQL engine version releases and updates. You can also find information about how to upgrade your Aurora PostgreSQL engine. For more information about Aurora releases in general, see [Amazon Aurora versions \(p. 5\)](#).

Topics

- [Identifying versions of Amazon Aurora PostgreSQL \(p. 1598\)](#)

- [Amazon Aurora PostgreSQL releases and engine versions \(p. 1599\)](#)
- [Extension versions for Amazon Aurora PostgreSQL \(p. 1668\)](#)
- [Upgrading the PostgreSQL DB engine for Aurora PostgreSQL \(p. 1681\)](#)
- [Aurora PostgreSQL long-term support \(LTS\) releases \(p. 1690\)](#)

Identifying versions of Amazon Aurora PostgreSQL

Amazon Aurora includes certain features that are general to Aurora and available to all Aurora DB clusters. Aurora includes other features that are specific to a particular database engine that Aurora supports. These features are available only to those Aurora DB clusters that use that database engine, such as Aurora PostgreSQL.

An Aurora database release typically has two version numbers, the database engine version number and the Aurora version number. If an Aurora PostgreSQL release has an Aurora version number, it's included after the engine version number in the [Amazon Aurora PostgreSQL releases and engine versions \(p. 1599\)](#) listing.

Aurora version number

Aurora version numbers use the `major.minor.patch` naming scheme. An Aurora patch version includes important bug fixes added to a minor version after its release. To learn more about Amazon Aurora major, minor, and patch releases, see [Amazon Aurora major versions \(p. 6\)](#), [Amazon Aurora minor versions \(p. 7\)](#), and [Amazon Aurora patch versions \(p. 7\)](#).

You can find out the Aurora version number of your Aurora PostgreSQL DB instance with the following SQL query:

```
pgres=> SELECT aurora_version();
```

Starting with the release of PostgreSQL versions 13.3, 12.8, 11.13, 10.18, and for all other later versions, Aurora version numbers align more closely to the PostgreSQL engine version. For example, querying an Aurora PostgreSQL 13.3 DB cluster returns the following:

```
aurora_version
-----
13.3.1
(1 row)
```

Prior releases, such as Aurora PostgreSQL 10.14 DB cluster, return version numbers similar to the following:

```
aurora_version
-----
2.7.3
(1 row)
```

PostgreSQL engine version numbers

Starting with PostgreSQL 10, PostgreSQL database engine versions use a `major.minor` numbering scheme for all releases. Some examples include PostgreSQL 10.18, PostgreSQL 12.7, and PostgreSQL 13.3.

Releases prior to PostgreSQL 10 use a `major.major.minor` numbering scheme in which the first two digits make up the major version number and a third digit denotes a minor version. For example, PostgreSQL 9.6 is a major version, with minor versions 9.6.19 or 9.6.21 indicated by the third digit.

Note

The PostgreSQL engine version 9.6 is no longer supported. To upgrade, see [Upgrading the PostgreSQL DB engine for Aurora PostgreSQL \(p. 1681\)](#).

You can find out the PostgreSQL database engine version number with the following SQL query:

```
pgres=> SELECT version();
```

For an Aurora PostgreSQL 13.3 DB cluster, the results are as follows:

```
version
-----
PostgreSQL 13.3 on x86_64-pc-linux-gnu, compiled by x86_64-pc-linux-gnu-gcc (GCC) 7.4.0,
64-bit
(1 row)
```

Amazon Aurora PostgreSQL releases and engine versions

Following, you can find information about versions of the Aurora PostgreSQL-Compatible Edition database engine that have been released for Amazon Aurora. Many of the listed releases include both a PostgreSQL version number and an Amazon Aurora version number. However, starting with the release of PostgreSQL versions 13.3, 12.8, 11.13, 10.18, and for all other later versions, Aurora version numbers aren't used. To determine the version numbers of your Aurora PostgreSQL database, see [Identifying versions of Amazon Aurora PostgreSQL \(p. 1598\)](#).

For information about extensions and modules, see [Extension versions for Amazon Aurora PostgreSQL \(p. 1668\)](#).

Amazon Aurora for PostgreSQL 1.X (compatible with PostgreSQL 9.6.XX) reaches end of life on January 31, 2022. For more information, see [Announcement: Amazon Aurora PostgreSQL 9.6 End-of-Life date is January 31, 2022](#). We recommend that you proactively upgrade your databases that are running Aurora PostgreSQL 9.6 to Amazon Aurora PostgreSQL 10 or higher at your convenience before January 31, 2022. To learn how, see [Upgrading the PostgreSQL DB engine for Aurora PostgreSQL \(p. 1681\)](#).

For more information about Amazon Aurora supported releases, policies, and timelines, see [How long Amazon Aurora major versions remain available \(p. 8\)](#). For more information about support and other policies for Amazon Aurora see [Amazon RDS FAQs](#).

To determine which Aurora PostgreSQL DB engine versions are available in an AWS Region, use the `describe-db-engine-versions` AWS CLI command. For example:

```
aws rds describe-db-engine-versions --engine aurora-postgresql --query '*[].[EngineVersion]' --output text --region aws-region
```

For a list of AWS Regions, see [Aurora PostgreSQL Region availability \(p. 14\)](#).

Topics

- [PostgreSQL 13.5 \(p. 1600\)](#)
- [PostgreSQL 13.4 \(p. 1601\)](#)
- [PostgreSQL 13.3 \(p. 1602\)](#)
- [PostgreSQL 12.9 \(p. 1604\)](#)
- [PostgreSQL 12.8 \(p. 1604\)](#)
- [PostgreSQL 12.7, Aurora PostgreSQL release 4.2 \(p. 1605\)](#)

- [PostgreSQL 12.6, Aurora PostgreSQL release 4.1 \(p. 1606\)](#)
- [PostgreSQL 12.4, Aurora PostgreSQL release 4.0 \(p. 1608\)](#)
- [PostgreSQL 11.14 \(p. 1610\)](#)
- [PostgreSQL 11.13 \(p. 1610\)](#)
- [PostgreSQL 11.12, Aurora PostgreSQL release 3.6 \(p. 1611\)](#)
- [PostgreSQL 11.11, Aurora PostgreSQL release 3.5 \(p. 1612\)](#)
- [PostgreSQL 11.9, Aurora PostgreSQL release 3.4 \(p. 1613\)](#)
- [PostgreSQL 11.8, Aurora PostgreSQL release 3.3 \(p. 1616\)](#)
- [PostgreSQL 11.7, Aurora PostgreSQL release 3.2 \(p. 1619\)](#)
- [PostgreSQL 11.6, Aurora PostgreSQL release 3.1 \(p. 1622\)](#)
- [PostgreSQL 11.4, Aurora PostgreSQL release 3.0 \(unsupported\) \(p. 1627\)](#)
- [PostgreSQL 10.19 \(p. 1628\)](#)
- [PostgreSQL 10.18 \(p. 1628\)](#)
- [PostgreSQL 10.17, Aurora PostgreSQL release 2.9 \(p. 1629\)](#)
- [PostgreSQL 10.16, Aurora PostgreSQL release 2.8 \(p. 1630\)](#)
- [PostgreSQL 10.14, Aurora PostgreSQL release 2.7 \(p. 1631\)](#)
- [PostgreSQL 10.13, Aurora PostgreSQL release 2.6 \(p. 1633\)](#)
- [PostgreSQL 10.12, Aurora PostgreSQL release 2.5 \(p. 1636\)](#)
- [PostgreSQL 10.11, Aurora PostgreSQL release 2.4 \(p. 1640\)](#)
- [PostgreSQL 10.7, Aurora PostgreSQL release 2.3 \(unsupported\) \(p. 1644\)](#)
- [PostgreSQL 10.6, Aurora PostgreSQL release 2.2 \(unsupported\) \(p. 1646\)](#)
- [PostgreSQL 10.5, Aurora PostgreSQL release 2.1 \(unsupported\) \(p. 1647\)](#)
- [PostgreSQL 10.4, Aurora PostgreSQL release 2.0 \(unsupported\) \(p. 1648\)](#)
- [PostgreSQL 9.6.22, Aurora PostgreSQL release 1.11 \(unsupported\) \(p. 1650\)](#)
- [PostgreSQL 9.6.21, Aurora PostgreSQL release 1.10 \(unsupported\) \(p. 1651\)](#)
- [PostgreSQL 9.6.19, Aurora PostgreSQL release 1.9 \(unsupported\) \(p. 1651\)](#)
- [PostgreSQL 9.6.18, Aurora PostgreSQL release 1.8 \(unsupported\) \(p. 1653\)](#)
- [PostgreSQL 9.6.17, Aurora PostgreSQL release 1.7 \(unsupported\) \(p. 1654\)](#)
- [PostgreSQL 9.6.16, Aurora PostgreSQL release 1.6 \(unsupported\) \(p. 1656\)](#)
- [PostgreSQL 9.6.12, Aurora PostgreSQL release 1.5 \(unsupported\) \(p. 1660\)](#)
- [PostgreSQL 9.6.11, Aurora PostgreSQL release 1.4 \(unsupported\) \(p. 1661\)](#)
- [PostgreSQL 9.6.9, Aurora PostgreSQL release 1.3 \(unsupported\) \(p. 1662\)](#)
- [PostgreSQL 9.6.8, Aurora PostgreSQL release 1.2 \(unsupported\) \(p. 1664\)](#)
- [PostgreSQL 9.6.6 Aurora PostgreSQL release 1.1 \(unsupported\) \(p. 1665\)](#)
- [PostgreSQL 9.6.3, Aurora PostgreSQL release 1.0 \(unsupported\) \(p. 1666\)](#)

PostgreSQL 13.5

This release of Aurora PostgreSQL is compatible with PostgreSQL 13.5. For more information about the improvements in PostgreSQL 13.5, see [PostgreSQL release 13.5](#).

Aurora PostgreSQL release 13.5

High priority stability enhancements

- Fixed a bug where logical replication may hang resulting in the replay falling behind on the read node. The instance may eventually restart.

Additional improvements and enhancements

- Added the `Buffers: shared hit` metric to the Explain output.
- Fixed a buffer cache bug that could cause brief periods of unavailability.
- Fixed a bug in the `apg_plan_mgmt` extension where an index based plan was not being enforced.
- Fixed a bug in the `pg_logical` extension that could cause brief periods of unavailability due to improper handling of NULL arguments.
- Fixed a bug that could cause brief periods of unavailability due to reading uninitialized pages.
- Fixed an issue where orphaned files caused major version upgrades to fail.
- Fixed incorrect Aurora Storage Daemon log write metrics.
- Fixed multiple bugs that could result in WAL replay falling behind and eventually causing the reader instances to restart.
- Improved the Aurora buffer cache page validation on reads.
- Improved the Aurora storage metadata validation.

This version also includes the following change:

- The `pg_cron` extension is updated to 1.4.1

For information about extensions and modules, see [Extensions supported for Aurora PostgreSQL 13.x \(p. 1668\)](#).

PostgreSQL 13.4

This release of Aurora PostgreSQL is compatible with PostgreSQL 13.4. For more information about the improvements in PostgreSQL 13.4, see [PostgreSQL release 13.4](#).

Aurora PostgreSQL release 13.4

New features

- This version supports Babelfish which extends your Amazon Aurora PostgreSQL database with the ability to accept database connections from Microsoft SQL Server clients. For more information, see [Working with Babelfish for Aurora PostgreSQL](#).

Critical stability enhancements

- Fixed an issue where, in rare circumstances, a data cache of a read node may be inconsistent following a restart of that node.

High priority stability enhancements

- Fixed an issue where queries may become unresponsive due to I/O resource exhaustion triggered by prefetch.
- Fixed an issue where Aurora may panic following a major version update with the message: "PANIC: could not access status of next transaction id xxxxxxxx".

Additional improvements and enhancements

- Fixed an issue where read nodes restart due to a replication origin cache lookup failure.
- Fixed an issue where read queries may time out on read nodes during the replay of lazy truncation triggered by vacuum on the write node.

- Fixed an issue that causes Performance Insights to incorrectly set the backend type of a database connection.
- Fixed an issue where the `aurora_postgres_replica_status()` function returned stale or lagging CPU stats.
- Fixed an issue where the role `rds_superuser` did not have permission to execute the `pg_stat_statements_reset()` function.
- Fixed an issue with the `apg_plan_mgmt` extension where the planning and execution times were reported as 0.
- Removed support for DES, 3DES and RC4 cipher suites.
- Updated the PostGIS extension to version 3.1.4.
- Updated the pgrouting extension to 3.1.3.
- Updated the pglogical extension to 2.4.0.
- Added support for the following SPI module extensions:
 - `autoinc` version 1.0
 - `insert_username` version 1.0
 - `moddatetime` version 1.0
 - `refint` version 1.0
- Fixed multiple issues in the Aurora storage daemon that could lead to brief periods of unavailability when specific network configurations are used.
- Fixed an out of memory crash issue with Aurora storage daemon that leads to writer node restart. This also reduces the overall system memory consumption.

PostgreSQL 13.3

This release of Aurora PostgreSQL is compatible with PostgreSQL 13.3. For more information about the improvements in PostgreSQL 13.3, see [PostgreSQL release 13.3](#).

Patch releases

- [Aurora PostgreSQL 13.3.1 \(p. 1602\)](#)
- [Aurora PostgreSQL release 13.3.0 \(p. 1603\)](#)

Aurora PostgreSQL 13.3.1

Critical stability enhancements

- Fixed an issue where, in rare circumstances, a data cache of a read node may be inconsistent following a restart of that node.

High priority stability enhancements

- Fixed an issue where queries may become unresponsive due to I/O resource exhaustion triggered by prefetch.
- Fixed an issue where Aurora may panic following a major version update with the message: "PANIC: could not access status of next transaction id xxxxxxxx".

Additional improvements and enhancements

- Fixed an issue where read nodes restart due to a replication origin cache lookup failure.
- Fixed an issue with the `apg_plan_mgmt` extension where the planning and execution times were reported as 0.

- Fixed an issue that causes Performance Insights to incorrectly set the backend type of a database connection.
- Fixed an issue with the `apg_plan_mgmt` extension where plan outline on a partitioned table did not enforce an index-based plan.
- Fixed an issue where orphaned files caused failed translations in read codepaths during or after a major version upgrade.
- Fixed multiple issues in the Aurora storage daemon that could lead to brief periods of unavailability when specific network configurations are used.
- Fixed an out of memory crash issue with Aurora storage daemon that leads to writer node restart. This also reduces the overall system memory consumption.

Aurora PostgreSQL release 13.3.0

New features

- Supports a major version upgrade from [PostgreSQL 12.4, Aurora PostgreSQL release 4.0 \(p. 1608\)](#) and later versions
- Supports `bool_plperl` version 1.0
- Supports `rds_tools` version 1.0

Critical stability enhancements

- Fixed an issue where, in rare circumstances, a data cache of a read node may be inconsistent following a restart of that node.

Additional improvements and enhancements

- Contains all of the fixes, features, and improvements present in [PostgreSQL 12.7, Aurora PostgreSQL release 4.2 \(p. 1605\)](#)
- Contains several improvements that were announced for PostgreSQL releases [13.0, 13.1, 13.2](#) and [13.3](#)
- Instance type R4 was deprecated.
- Updated the following extensions:
 - `hll` to version 2.15.
 - `hstore` to version 1.7.
 - `intarray` to version 1.3.
 - `log_fdw` to version 1.2.
 - `ltree` to version 1.2.
 - `pg_hint_plan` to version 1.3.7.
 - `pg_repack` to version 1.4.6.
 - `pg_stat_statements` to version 1.8.
 - `pg_trgm` to version 1.5.
 - `pgaudit` to version 1.5.
 - `pglogical` to version 2.3.3.
 - `pgrouting` to version 3.1.0
 - `plcoffee` to version 2.3.15.
 - `plls` to version 2.3.15.
 - `plv8` to version 2.3.15.

PostgreSQL 12.9

This release of Aurora PostgreSQL is compatible with PostgreSQL 12.9. For more information about the improvements in PostgreSQL 12.9, see [PostgreSQL release 12.9](#).

Aurora PostgreSQL release 12.9

Critical stability enhancements

- Fixed a bug where logical replication may hang resulting in the replay falling behind on the read node. The instance may eventually restart.

Additional improvements and enhancements

- Fixed a buffer cache bug that could cause brief periods of unavailability.
 - Fixed a bug in the `apg_plan_mgmt` extension where an index based plan was not being enforced.
 - Fixed a bug in the `pg_logical` extension that could cause brief periods of unavailability due to improper handling of NULL arguments.
 - Fixed an issue where orphaned files caused major version upgrades to fail.
 - Fixed incorrect Aurora Storage Daemon log write metrics.
 - Fixed multiple bugs that could result in WAL replay falling behind and eventually causing the reader instances to restart.
 - Improved the Aurora buffer cache page validation on reads.
 - Improved the Aurora storage metadata validation.
 - Updated the `pg_cron` extension to v1.4.
 - Updated the `pg_hint_plan` extension to v1.3.7.
-
- For information about extensions and modules, see [Extensions supported for Aurora PostgreSQL 12.x \(p. 1671\)](#).

PostgreSQL 12.8

This release of Aurora PostgreSQL is compatible with PostgreSQL 12.8. For more information about the improvements in PostgreSQL 12.8, see [PostgreSQL release 12.8](#).

Critical stability enhancements

- Fixed an issue where, in rare circumstances, a data cache of a read node may be inconsistent following a restart of that node.

High priority stability enhancements

- Fixed an issue where queries may become unresponsive due to I/O resource exhaustion triggered by prefetch.
- Fixed an issue where Aurora may panic following a major version update with the message: "PANIC: could not access status of next transaction id xxxxxxxx".

Additional improvements and enhancements

- Fixed an issue where read nodes restart due to a replication origin cache lookup failure.

- Fixed an issue where read queries may time out on read nodes during the replay of lazy truncation triggered by vacuum on the write node.
- Fixed an issue that causes Performance Insights to incorrectly set the backend type of a database connection.
- Fixed an issue where the `aurora_postgres_replica_status()` function returned stale or lagging CPU stats.
- Fixed an issue where the role `rds_superuser` did not have permission to execute the `pg_stat_statements_reset()` function.
- Fixed an issue with the `apg_plan_mgmt` extension where the planning and execution times were reported as 0.
- Removed support for DES, 3DES and RC4 cipher suites.
- Updated PostGIS extension to version 3.1.4.

PostgreSQL 12.7, Aurora PostgreSQL release 4.2

This release of Aurora PostgreSQL is compatible with PostgreSQL 12.7. For more information about the improvements in PostgreSQL 12.7, see [PostgreSQL release 12.7](#).

Patch releases

- [Aurora PostgreSQL 4.2.1 \(p. 1605\)](#)
- [Aurora PostgreSQL release 4.2.0 \(p. 1606\)](#)

Aurora PostgreSQL 4.2.1

Critical stability enhancements

- Fixed an issue where, in rare circumstances, a data cache of a read node may be inconsistent following a restart of that node.

High priority stability enhancements

- Fixed an issue where queries may become unresponsive due to I/O resource exhaustion triggered by prefetch.
- Fixed an issue where Aurora may panic following a major version update with the message: "PANIC: could not access status of next transaction id xxxxxxxx".

Additional improvements and enhancements

- Fixed an issue where read nodes restart due to a replication origin cache lookup failure.
- Fixed an issue with the `apg_plan_mgmt` extension where planning and execution time were reported as 0.
- Fixed an issue that causes Performance Insights to incorrectly set the backend type of a database connection.
- Fixed an issue with the `apg_plan_mgmt` extension where plan outline on a partitioned table did not enforce an index-based plan.
- Fixed an issue where orphaned files caused failed translations in read codepaths during or after major version upgrade.
- Fixed multiple issues in the Aurora storage daemon that could lead to brief periods of unavailability when specific network configurations are used.
- Fixed an out of memory crash issue with Aurora storage daemon that leads to writer node restart. This also reduces the overall system memory consumption.

Aurora PostgreSQL release 4.2.0

New features

- Added support for the `oracle_fdw` extension version 2.3.0.

High priority stability enhancements

- Fixed an issue where creating a database from an existing template database with tablespace resulted in an error with the message `ERROR: could not open file pg_tblspc/...: No such file or directory`.
- Fixed an issue where, in rare cases, an Aurora replica may be unable to start when a large number of PostgreSQL subtransactions (i.e. SQL savepoints) have been used.
- Fixed an issue where, in rare circumstances, read results may be inconsistent for repeated read requests on replica nodes.

Additional improvements and enhancements

- Upgraded OpenSSL to 1.1.1k.
- Reduced CPU usage and memory consumption of the WAL apply process on Aurora replicas for some workloads.
- Improved safety checks in the write path to detect incorrect writes to metadata.
- Improved security by removing 3DES and other older ciphers for SSL/TLS connections.
- Fixed an issue where a duplicate file entry can prevent the Aurora PostgreSQL engine from starting up.
- Fixed an issue that could cause temporary unavailability under heavy workloads.
- Added back ability to use a leading forward slash in the S3 path during S3 import.
- Added Graviton support for `oracle_fdw` extension version 2.3.0.
- Changed the following extensions:
 - Updated the `Orafcce` extension to version 3.16.
 - Updated the `pg_partman` extension to version 4.5.1.
 - Updated the `pg_cron` extension to version 1.3.1.
 - Updated the `postgis` extension to version 3.0.3.

PostgreSQL 12.6, Aurora PostgreSQL release 4.1

This release of Aurora PostgreSQL is compatible with PostgreSQL 12.6. For more information about the improvements in PostgreSQL 12.6, see [PostgreSQL release 12.6](#).

Aurora PostgreSQL release 4.1.0

New features

- Added support for the following extensions:
 - The `pg_proctab` extension version 0.0.9
 - The `pg_partman` extension version 4.4.0. For more information, see [Managing PostgreSQL partitions with the pg_partman extension \(p. 1530\)](#).
 - The `pg_cron` extension version 1.3.0. For more information, see [Scheduling maintenance with the PostgreSQL pg_cron extension \(p. 1370\)](#).
 - The `pg_bigm` extension version 1.2

High priority stability enhancements

- Fixed a bug in the `pglogical` extension that could lead to data inconsistency for inbound replication.
- Fixed a bug where in rare cases a reader had inconsistent results when it restarted while a transaction with more than 64 subtransactions was being processed.
- Backported fixes for the following PostgreSQL community security issues:
 - [CVE-2021-32027](#)
 - [CVE-2021-32028](#)
 - [CVE-2021-32029](#)

Additional improvements and enhancements

- Fixed a bug where the database could not be started when there were many relations in memory-constrained environments.
- Fixed a bug in the `apg_plan_mgmt` extension that could cause brief periods of unavailability due to an internal buffer overflow.
- Fixed a bug on reader nodes that could cause brief periods of unavailability during WAL replay.
- Fixed a bug in the `rds_activity_stream` extension that caused an error during startup when attempting to log audit events.
- Fixed bugs in the `aurora_replica_status` function where rows were sometimes partially populated and some values such as Replay Latency, and CPU usage were always 0.
- Fixed a bug where the database engine would attempt to create shared memory segments larger than the instance total memory and fail repeatedly. For example, attempts to create 128 GiB shared buffers on a db.r5.large instance would fail. With this change, requests for total shared memory allocations larger than the instance memory allow setting the instance to incompatible parameters.
- Added logic to clean up unnecessary `pg_wal` temporary files on a database startup.
- Fixed a bug that could lead to outbound replication synchronization errors after a major version upgrade.
- Fixed a bug that reported ERROR: `rds_activity_stream` stack item 2 not found on top - cannot pop when attempting to create the `rds_activity_stream` extension.
- Fixed a bug that could cause the error failed to build any 3-way joins in a correlated `IN` subquery under an `EXISTS` subquery.
- Backported the following performance improvement from the PostgreSQL community:
[pg_stat_statements: add missing check for pgss_enabled\(\)](#).
- Fixed a bug that could cause upgrades to Aurora PostgreSQL 12.x to fail due to the inability to open the `pg_control` file.
- Fixed a bug that could cause brief periods of unavailability due to running out of memory when creating the `postgis` extension with `pgAudit` enabled.
- Backported the following bug fix from the PostgreSQL community: [Fix use-after-free bug with AfterTriggersTableData.storeslot](#).
- Fixed a bug when using outbound logical replication to synchronize changes to another database that could fail with an error message like ERROR: could not map filenode "base/16395/228486645" to relation OID.
- Fixed a bug that could cause a brief period of unavailability when aborting a transaction.
- Fixed a bug that caused no ICU collations to be shown in the `pg_collation` catalog table after creating a new Aurora PostgreSQL 12.x instance. This issue does not affect upgrading from an older version.
- Fixed a bug where the `rds_ad` role wasn't created after upgrading from a version of Aurora PostgreSQL that doesn't support Microsoft Active Directory authentication.
- Added btree page checks to detect tuple metadata inconsistency.

- Fixed a bug in asynchronous buffer reads that could cause brief periods of unavailability on reader nodes during WAL replay.
- Fixed a bug where reading a TOAST value from disk could cause a brief period of unavailability.
- Fixed a bug that caused brief periods of unavailability when attempting to fetch a tuple from an index scan.

PostgreSQL 12.4, Aurora PostgreSQL release 4.0

This release of Aurora PostgreSQL is compatible with PostgreSQL 12.4. For more information about the improvements in PostgreSQL 12.4, see [PostgreSQL release 12.4](#).

Patch releases

- [Aurora PostgreSQL release 4.0.2 \(p. 1608\)](#)
- [Aurora PostgreSQL release 4.0.1 \(p. 1608\)](#)
- [Aurora PostgreSQL release 4.0.0 \(p. 1609\)](#)

Aurora PostgreSQL release 4.0.2

High priority stability enhancements

- Fixed a bug where a reader node might render an extra or missing row if the reader restarted while the writer node is processing a long transaction with more than 64 subtransactions.
- Fixed a bug that can cause vacuum to block on GiST indexes.
- Fixed a bug where after upgrade to PostgreSQL 12, vacuum can fail on the system table `pg_catalog.pg_shdescription` with the following error. ERROR: found xmin 484 from before relfrozenid

Additional improvements and enhancements

- Fixed a bug that could lead to intermittent unavailability due to a race condition when handling responses from storage nodes.
- Fixed a bug that could lead to intermittent unavailability due to the rotation of network encryption keys.
- Fixed a bug that could lead to intermittent unavailability due to heat management of the underlying storage segments.
- Fixed a bug where a large S3 import with thousands of clients can cause one or more of the import clients to stop responding.
- Removed a restriction that prevented setting configuration variable strings that contained `brazil`.
- Fixed a bug that could lead to intermittent unavailability if a reader node runs a query that access many tables while the writer node is acquiring exclusive locks on all of the same tables.

Aurora PostgreSQL release 4.0.1

New features

- This release adds support for the [Graviton2 db.r6g instance classes \(p. 54\)](#) to the PostgreSQL engine version 12.4.

Critical stability enhancements

- Fixed a bug that caused a read replica to unsuccessfully restart repeatedly in rare cases.

- Fixed a bug where a cluster became unavailable when attempting to create more than 16 read replicas or Aurora global database secondary AWS Regions. The cluster became available again when the new read replica or secondary AWS Region was removed.

Additional improvements and enhancements

- Fixed a bug that when under heavy load, snapshot import, COPY import, or Amazon S3 import stopped responding in rare cases.
- Fixed a bug where a read replica might not join the cluster when the writer was very busy with a write-intensive workload.
- Fixed a bug where a cluster could be unavailable briefly when a high-volume S3 import was running.
- Fixed a bug that caused a cluster to take several minutes to restart if a logical replication stream was terminated while handling many complex transactions.
- Fixed the Just-in-Time (JIT) compilation, which was incorrectly enabled by default in Aurora PostgreSQL release 4.0.0.
- Disallowed the use of both AWS Identity and Access Management (IAM) and Kerberos authentication for the same user.

Aurora PostgreSQL release 4.0.0

New features

- This version supports a major version upgrade from [PostgreSQL 11.7, Aurora PostgreSQL release 3.2 \(p. 1619\)](#) and later versions.

Additional improvements and enhancements

- Contains several improvements that were announced for PostgreSQL releases [12.0](#), [12.1](#), [12.2](#), [12.3](#), and [12.4](#).
- Contains all fixes, features, and improvements present in [PostgreSQL 11.9, Aurora PostgreSQL release 3.4 \(p. 1613\)](#).
- Backported fixes for the following PostgreSQL community security issues:
 - [CVE-2020-25694](#)
 - [CVE-2020-25695](#)
 - [CVE-2020-25696](#)
- Updated the following extensions:
 - `address_standardizer` to version 3.0.2
 - `address_standardizer_data_us` to version 3.0.2
 - `amcheck` to version 1.2
 - `citext` to version 1.6
 - `hll` to version 2.14
 - `hstore` to version 1.6
 - `ip4r` to version 2.4
 - `pg_repack` to version 1.4.5
 - `pg_stat_statements` to version 1.7
 - `pgaudit` to version 1.4
 - `pglogical` to version 2.3.2
 - `pgrouting` to version 3.0.3
 - `plv8` to version 2.3.14

- `postGIS` to version 3.0.2
- `postgis_tiger_geocoder` to version 3.0.2
- `postgis_topology` to version 3.0.2

PostgreSQL 11.14

This release of Aurora PostgreSQL is compatible with PostgreSQL 11.14. For more information about the improvements in PostgreSQL 11.14, see [PostgreSQL release 11.14](#).

Aurora PostgreSQL release 11.14

Critical stability enhancements

- Fixed a bug where logical replication may hang resulting in the replay falling behind on the read node. The instance may eventually restart.

Additional improvements and enhancements

- Fixed a buffer cache bug that could cause brief periods of unavailability.
 - Fixed a bug in the `apg_plan_mgmt` extension where an index based plan was not being enforced.
 - Fixed a bug in the `pg_logical` extension that could cause brief periods of unavailability due to improper handling of NULL arguments.
 - Fixed an issue where orphaned files caused major version upgrades to fail.
 - Fixed incorrect Aurora Storage Daemon log write metrics.
 - Fixed multiple bugs that could result in WAL replay falling behind and eventually causing the reader instances to restart.
 - Improved the Aurora buffer cache page validation on reads.
 - Improved the Aurora storage metadata validation.
 - Updated the `pg_hint_plan` extension to v1.3.7.
-
- For information about extensions and modules, see [Extensions supported for Aurora PostgreSQL 11.x \(p. 1674\)](#).

PostgreSQL 11.13

This release of Aurora PostgreSQL is compatible with PostgreSQL 11.13. For more information about the improvements in PostgreSQL 11.13, see [PostgreSQL release 11.13](#).

Critical stability enhancements

- Fixed an issue where, in rare circumstances, a data cache of a read node may be inconsistent following a restart of that node.

High priority stability enhancements

- Fixed an issue where queries may become unresponsive due to I/O resource exhaustion triggered by prefetch.
- Fixed an issue where Aurora may panic following a major version update with the message: "PANIC: could not access status of next transaction id xxxxxxxx".

Additional improvements and enhancements

- Fixed an issue where read nodes restart due to a replication origin cache lookup failure.
- Fixed an issue where read queries may time out on read nodes during the replay of lazy truncation triggered by vacuum on the write node.
- Fixed an issue that causes Performance Insights to incorrectly set the backend type of a database connection.
- Fixed an issue where the `aurora_postgres_replica_status()` function returned stale or lagging CPU stats.
- Fixed an issue where, in rare cases, an Aurora Global Database secondary mirror cluster may restart due to a stall in the log apply process.
- Fixed an issue with the `apg_plan_mgmt` extension where the planning and execution times were reported as 0.
- Removed support for DES, 3DES and RC4 cipher suites.
- Updated PostGIS extension to version 3.1.4.
- Added support for `postgis_raster` extension version 3.1.4.

PostgreSQL 11.12, Aurora PostgreSQL release 3.6

This release of Aurora PostgreSQL is compatible with PostgreSQL 11.12. For more information about the improvements in PostgreSQL 11.12, see [PostgreSQL release 11.12](#).

Patch releases

- [Aurora PostgreSQL 3.6.1 \(p. 1611\)](#)
- [Aurora PostgreSQL release 3.6.0 \(p. 1612\)](#)

Aurora PostgreSQL 3.6.1

Critical stability enhancements

- Fixed an issue where, in rare circumstances, a data cache of a read node may be inconsistent following a restart of that node.

High priority stability enhancements

- Fixed an issue where queries may become unresponsive due to I/O resource exhaustion triggered by prefetch.
- Fixed an issue where Aurora may panic following a major version update with the message: "PANIC: could not access status of next transaction id xxxxxxxx".
- Fixed multiple issues in the Aurora storage daemon that could lead to brief periods of unavailability when specific network configurations are used.
- Fixed an out of memory crash issue with Aurora storage daemon that leads to writer node restart. This also reduces the overall system memory consumption.

Additional improvements and enhancements

- Fixed an issue where read nodes restart due to a replication origin cache lookup failure.
- Fixed an issue with the `apg_plan_mgmt` extension where planning and execution time were reported as 0.
- Fixed an issue that causes Performance Insights to incorrectly set the backend type of a database connection.

- Fixed an issue where in rare cases, an Aurora Global Database secondary mirror cluster may restart due to a stall in the log apply process.
- Fixed an issue where orphaned files caused failed translations in read codepaths during or after major version upgrade.
- Fixed multiple issues in the Aurora storage daemon that could lead to brief periods of unavailability when specific network configurations are used.
- Fixed an out of memory crash issue with Aurora storage daemon that leads to writer node restart. This also reduces the overall system memory consumption.

Aurora PostgreSQL release 3.6.0

High priority stability enhancements

- Fixed an issue where creating a database from an existing template database with tablespace resulted in an error with the message `ERROR: could not open file pg_tblspc/...: No such file or directory.`
- Fixed an issue where, in rare cases, an Aurora replica may be unable to start when a large number of PostgreSQL subtransactions (i.e. SQL savepoints) have been used.
- Fixed an issue where, in rare circumstances, read results may be inconsistent for repeated read requests on replica nodes.

Additional improvements and enhancements

- Upgraded OpenSSL to 1.1.1k.
- Reduced CPU usage and memory consumption of the WAL apply process on Aurora replicas for some workloads.
- Improved metadata protection from accidental erasure.
- Improved safety checks in the write path to detect incorrect writes to metadata.
- Improved security by removing 3DES and other older ciphers for SSL/TLS connections.
- Fixed an issue where a duplicate file entry can prevent the Aurora PostgreSQL engine from starting up.
- Fixed an issue that could cause temporary unavailability under heavy workloads.
- Added back ability to use a leading forward slash in the S3 path during S3 import.
- Updated the `orafce` extension to version 3.16.

PostgreSQL 11.11, Aurora PostgreSQL release 3.5

This release of Aurora PostgreSQL is compatible with PostgreSQL 11.11. For more information about the improvements in PostgreSQL 11.11, see [PostgreSQL release 11.11](#).

Aurora PostgreSQL release 3.5.0

New features

- Added support for the following extensions:
 - The `pg_proctab` extension version 0.0.9
 - The `pg_bigm` extension version 1.2

High priority stability enhancements

- Fixed a bug where in rare cases a reader had inconsistent results when it restarted while a transaction with more than 64 subtransactions was being processed.

- Backported fixes for the following PostgreSQL community security issues:
 - [CVE-2021-32027](#)
 - [CVE-2021-32028](#)
 - [CVE-2021-32029](#)

Additional improvements and enhancements

- Fixed a bug where the database could not be started when there were many relations in memory-constrained environments.
- Fixed a bug in the `apg_plan_mgmt` extension that could cause brief periods of unavailability due to an internal buffer overflow.
- Fixed a bug on reader nodes that could cause brief periods of unavailability during WAL replay.
- Fixed a bug in the `rds_activity_stream` extension that caused an error during startup when attempting to log audit events.
- Fixed bugs in the `aurora_replica_status` function where rows were sometimes partially populated and some values such as Replay Latency, and CPU usage were always 0.
- Fixed a bug where the database engine would attempt to create shared memory segments larger than the instance total memory and fail repeatedly. For example, attempts to create 128 GiB shared buffers on a db.r5.large instance would fail. With this change, requests for total shared memory allocations larger than the instance memory allow setting the instance to incompatible parameters.
- Added logic to clean up unnecessary `pg_wal` temporary files on a database startup.
- Fixed a bug that reported ERROR: `rds_activity_stream` stack item 2 not found on top - cannot pop when attempting to create the `rds_activity_stream` extension.
- Fixed a bug that could cause the error failed to build any 3-way joins in a correlated `IN` subquery under an `EXISTS` subquery.
- Backported the following performance improvement from the PostgreSQL community: [pg_stat_statements: add missing check for pgss_enabled\(\)](#).
- Fixed a bug that could cause brief periods of unavailability due to running out of memory when creating the `postgis` extension with `pgAudit` enabled.
- Fixed a bug when using outbound logical replication to synchronize changes to another database that could fail with an error message like ERROR: could not map filenode "base/16395/228486645" to relation OID.
- Fixed a bug that could cause a brief period of unavailability when aborting a transaction.
- Fixed a bug where the `rds_ad` role wasn't created after upgrading from a version of Aurora PostgreSQL that doesn't support Microsoft Active Directory authentication.
- Added btree page checks to detect tuple metadata inconsistency.
- Fixed a bug in asynchronous buffer reads that could cause brief periods of unavailability on reader nodes during WAL replay.

PostgreSQL 11.9, Aurora PostgreSQL release 3.4

This release of Aurora PostgreSQL is compatible with PostgreSQL 11.9. For more information about the improvements in PostgreSQL 11.9, see [PostgreSQL release 11.9](#).

Patch releases

- [Aurora PostgreSQL release 3.4.3 \(p. 1614\)](#)
- [Aurora PostgreSQL release 3.4.2 \(p. 1614\)](#)
- [Aurora PostgreSQL release 3.4.1 \(p. 1614\)](#)

- [Aurora PostgreSQL release 3.4.0 \(p. 1615\)](#)

Aurora PostgreSQL release 3.4.3

High priority stability enhancements

- Provided a patch for PostgreSQL community security issues CVE-2021-32027, CVE-2021-32028 and CVE-2021-32029.

Additional improvements and enhancements

- Fixed a bug in the aws_s3 extension to allow import of objects with leading forward slashes in the object identifier.
- Fixed a bug in the rds_activity_stream extension that caused an error during startup when attempting to log audit events.
- Fixed a bug that returned an `ERROR` when attempting to create the `rds_activity_stream` extension.
- Fixed a bug that could cause brief periods of unavailability due to running out of memory when creating the `postgis` extension with `pgAudit` enabled.
- Fixed multiple issues in the Aurora storage daemon that could lead to brief periods of unavailability when specific network configurations are used.

Aurora PostgreSQL release 3.4.2

High priority stability enhancements

- Fixed a bug where in rare cases a reader had inconsistent results when it restarted while a transaction with more than 64 subtransactions was being processed.

Additional improvements and enhancements

- Fixed a bug that could lead to intermittent unavailability due to a race condition when handling responses from storage nodes.
- Fixed a bug that could lead to intermittent unavailability due to the rotation of network encryption keys.
- Fixed a bug that could lead to intermittent unavailability due to heat management of the underlying storage segments.
- Fixed a bug where a large S3 import with thousands of clients can cause one or more of the import clients to stop responding.
- Removed a restriction that prevented setting configuration variable strings that contained `brazil`.
- Fixed a bug that could lead to intermittent unavailability if a reader node runs a query that access many tables while the writer node is acquiring exclusive locks on all of the same tables.

Aurora PostgreSQL release 3.4.1

Critical stability enhancements

- Fixed a bug that caused a read replica to unsuccessfully restart repeatedly in rare cases.
- Fixed a bug where a cluster became unavailable when attempting to create more than 16 read replicas or Aurora global database secondary AWS Regions. The cluster became available again when the new read replica or secondary AWS Region was removed.

Additional improvements and enhancements

- Fixed a bug that when under heavy load, snapshot import, COPY import, or S3 import stopped responding in rare cases.
- Fixed a bug where a read replica might not join the cluster when the writer was very busy with a write-intensive workload.
- Fixed a bug where a cluster could be unavailable briefly when a high-volume S3 import was running.
- Fixed a bug that caused a cluster to take several minutes to restart if a logical replication stream was terminated while handling many complex transactions.
- Disallowed the use of both IAM and Kerberos authentication for the same user.

Aurora PostgreSQL release 3.4.0

New features

- Aurora PostgreSQL now supports invocation of AWS Lambda functions. This includes the new `aws_lambda` extension. For more information, see [Invoking an AWS Lambda function from an Aurora PostgreSQL DB cluster \(p. 1517\)](#).
- The `db.r6g` instance classes are now available in preview for Aurora. For more information, see [Aurora DB instance classes \(p. 54\)](#).

Critical stability enhancements

- None

High priority stability enhancements

- Fixed a bug in Aurora PostgreSQL replication that could result in the error message ERROR: MultiXactId nnnn has not been created yet -- apparent wraparound.
- Fixed a bug where in some cases, DB clusters that have logical replication enabled did not remove truncated WAL segment files from storage. This resulted in volume size growth.
- Backported fixes for the following PostgreSQL community security issues:
 - [CVE-2020-25694](#)
 - [CVE-2020-25695](#)
 - [CVE-2020-25696](#)
- Fixed a bug in the `pg_stat_statements` extension that caused excessive CPU consumption.

Additional improvements and enhancements

- You can now use `pg_replication_slot_advance` to advance a logical replication slot for the roles `rds_replication` and `rds_superuser`.
- Improved the asynchronous mode performance of database activity streams.
- Reduced the delay when publishing to CloudWatch the `rpo_lag_in_msec` metric for Aurora global database clusters.
- Aurora PostgreSQL no longer falls behind on a read node when the backend is blocked writing to the database client.
- Fixed a bug that in rare cases caused a brief period of unavailability on a read replica when the storage volume grew.
- Fixed a bug when creating a database that could return the following: ERROR: could not create directory on local disk

- Updated data grid files to fix errors or incorrect transformation results from the `ST_Transform` method of the PostGIS extension.
- Fixed a bug where in some cases replaying `XLOG_BTREE_REUSE_PAGE` records on Aurora reader instances caused unnecessary replay lag.
- Fixed a small memory leak in a b-tree index that could lead to an out of memory condition.
- Fixed a bug in the GiST index that could result in an out of memory condition after promoting an Aurora read replica.
- Fixed an S3 import bug that reported `ERROR: HTTP 403. Permission denied` when importing data from a file inside an S3 subfolder.
- Fixed a bug in the `aws_s3` extension for pre-signed URL handling that could result in the error message S3 bucket names with a period (.) are not supported.
- Fixed a bug in the `aws_s3` extension where an import might be blocked indefinitely if an exclusive lock was taken on the relation prior to beginning the operation.
- Fixed a bug related to replication when Aurora PostgreSQL is acting as a physical replica of an RDS PostgreSQL instance that uses GiST indexes. In rare cases, this bug caused a brief period of unavailability after promoting the Aurora cluster.
- Fixed a bug in database activity streams where customers were not notified of the end of an outage.
- Updated the `pg_audit` extension to version 1.3.1.

PostgreSQL 11.8, Aurora PostgreSQL release 3.3

This release of Aurora PostgreSQL is compatible with PostgreSQL 11.8. For more information about the improvements in PostgreSQL 11.8, see [PostgreSQL release 11.8](#).

Patch releases

- [Aurora PostgreSQL release 3.3.2 \(p. 1616\)](#)
- [Aurora PostgreSQL release 3.3.1 \(p. 1617\)](#)
- [Aurora PostgreSQL release 3.3.0 \(p. 1617\)](#)

Aurora PostgreSQL release 3.3.2

Critical stability enhancements

- None

High priority stability enhancements

- Fixed a bug in Aurora PostgreSQL replication that could result in the error message `ERROR: MultiXactId nnnn has not been created yet -- apparent wraparound`.
- Fixed a bug where in some cases, DB clusters that have logical replication enabled did not remove truncated WAL segment files from storage. This resulted in volume size growth.
- Fixed an issue with creating a global database cluster in a secondary region.
- Backported fixes for the following PostgreSQL community security issues:
 - [CVE-2020-25694](#)
 - [CVE-2020-25695](#)
 - [CVE-2020-25696](#)
- Fixed a bug in the `pg_stat_statements` extension that caused excessive CPU consumption.

Additional improvements and enhancements

- Aurora PostgreSQL no longer falls behind on a read node when the backend is blocked writing to the database client.
- Reduced the delay when publishing to CloudWatch the `rpo_lag_in_msec` metric for Aurora global database clusters.
- Fixed a bug where a `DROP DATABASE` statement didn't remove any relation files.
- Fixed a bug where in some cases replaying `XLOG_BTREE_REUSE_PAGE` records on Aurora reader instances caused unnecessary replay lag.
- Fixed a small memory leak in a b-tree index that could lead to an out of memory condition.
- Fixed a bug in the `aurora_replica_status()` function where the `server_id` field was sometimes truncated.
- Fixed a bug where a log record was incorrectly processed causing the Aurora replica to crash.
- Fixed an S3 import bug that reported ERROR: HTTP 403. Permission denied when importing data from a file inside an S3 subfolder.
- You can now use `pg_replication_slot_advance` to advance a logical replication slot for the roles `rds_replication` and `rds_superuser`.
- Improved performance of the asynchronous mode for database activity streams.
- Fixed a bug in the `aws_s3` extension that could result in the error message S3 bucket names with a period (.) are not supported.
- Fixed a race condition that caused valid imports to intermittently fail.
- Fixed a bug related to replication when Aurora PostgreSQL is acting as a physical replica of an RDS PostgreSQL instance that uses GiST indexes. In rare cases, this bug caused a brief period of unavailability after promoting the Aurora DB cluster.
- Fixed a bug in the `aws_s3` extension where an import may be blocked indefinitely if an exclusive lock was taken on the relation prior to beginning the operation.

Aurora PostgreSQL release 3.3.1

You can find the following improvements in this release.

Critical stability enhancements

1. Fixed a bug that appears when the `NOT EXISTS` operator incorrectly returns TRUE, which can only happen when the following unusual set of circumstances occurs:
 - A query is using the `NOT EXISTS` operator.
 - The column or columns being evaluated against the outer query in the `NOT EXISTS` subquery contain a NULL value.
 - There isn't another predicate in the subquery that removes the need for the evaluation of the NULL values.
 - The filter used in the subquery does not use an index seek for its execution.
 - The operator isn't converted to a join by the query optimizer.

Aurora PostgreSQL release 3.3.0

New features

- Added support for the RDKit extension version 3.8.

The RDKit extension provides modeling functions for cheminformatics. Cheminformatics is storing, indexing, searching, retrieving, and applying information about chemical compounds. For example,

with the RDKit extension you can construct models of molecules, search for molecular structures, and read or create molecules in various notations. You can also perform research on data loaded from the [ChEMBL website](#) or a SMILES file. The Simplified Molecular Input Line Entry System (SMILES) is a typographical notation for representing molecules and reactions. For more information, see [The RDKit database cartridge](#) in the RDKit documentation.

- Added support for a minimum TLS version

Support for a minimum Transport Layer Security (TLS) version is back ported from PostgreSQL 12. It allows the Aurora PostgreSQL server to constrain the TLS protocols with which a client is allowed to connect via two new PostgreSQL parameters. These parameters include `ssl_min_protocol_version` and `ssl_max_protocol_version`. For example, to limit client connections to the Aurora PostgreSQL server to at least TLS 1.2 protocol version, set the `ssl_min_protocol_version` to `TLSv1.2`.

- Added support for the `pglogical` extension version 2.2.2.

The `pglogical` extension is a logical streaming replication system that provides additional features beyond what's available in PostgreSQL native logical replication. Features include conflict handling, row filtering, DDL/sequence replication and delayed apply. You can use the `pglogical` extension to set up replication between Aurora PostgreSQL clusters, between RDS PostgreSQL and Aurora PostgreSQL, and with PostgreSQL databases running outside of RDS.

- Aurora dynamically resizes your cluster storage space. With dynamic resizing, the storage space for your Aurora DB cluster automatically decreases when you remove data from the DB cluster. For more information, see [Storage scaling \(p. 396\)](#).

Note

The dynamic resizing feature is being deployed in phases to the AWS Regions where Aurora is available. Depending on the Region where your cluster is, this feature might not be available yet. For more information, see [the What's New announcement](#).

Critical stability enhancements

- Fixed a bug related to heap page extend that in rare cases resulted in longer recovery time and impacted availability.

High priority stability enhancements

- Fixed a bug in Aurora Global Database that could cause delays in upgrading the database engine in a secondary AWS Region. For more information, see [Using Amazon Aurora global databases \(p. 225\)](#).
- Fixed a bug that in rare cases caused delays in upgrading a database to engine version 11.8.

Additional improvements and enhancements

- Fixed a bug where the Aurora replica crashed when workloads with heavy subtransactions are made on the writer instance.
- Fixed a bug where the writer instance crashed due to a memory leak and the depletion of memory used to track active transactions.
- Fixed a bug that lead to a crash due to improper initialization when there is no free memory available during PostgreSQL backend startup.
- Fixed a bug where an Aurora PostgreSQL Serverless DB cluster might return the following error after a scaling event: `ERROR: prepared statement "S_6" already exists.`
- Fixed an out-of-memory problem when issuing the `CREATE EXTENSION` command with PostGIS when Database Activity Streams enabled.
- Fixed a bug where a `SELECT` query might incorrectly return the error `Attempting to read past EOF of relation rrrr. blockno=bbb nblocks=nnn.`

- Fixed a bug where the database might be unavailable briefly due to error handling in database storage growth.
- Fixed a bug in Aurora PostgreSQL Serverless where queries that executed on previously idle connections got delayed until the scale operation completed.
- Fixed a bug where an Aurora PostgreSQL DB cluster with Database Activity Streams enabled might report the beginning of a potential loss window for activity records, but does not report the restoration of connectivity.
- Fixed a bug with the [aws_s3.table_import_from_s3](#) (p. 1446) function where a COPY from S3 failed with the error HTTP error code: 248.

PostgreSQL 11.7, Aurora PostgreSQL release 3.2

This release of Aurora PostgreSQL is compatible with PostgreSQL 11.7. For more information about the improvements in PostgreSQL 11.7, see [PostgreSQL release 11.7](#).

Patch releases

- [Aurora PostgreSQL release 3.2.7 \(p. 1619\)](#)
- [Aurora PostgreSQL release 3.2.6 \(p. 1619\)](#)
- [Aurora PostgreSQL release 3.2.4 \(p. 1620\)](#)
- [Aurora PostgreSQL release 3.2.3 \(p. 1620\)](#)
- [Aurora PostgreSQL release 3.2.2 \(p. 1621\)](#)
- [Aurora PostgreSQL release 3.2.1 \(p. 1621\)](#)

Aurora PostgreSQL release 3.2.7

You can find the following improvements in this release.

Critical stability enhancements

- None

High priority stability enhancements

- Backported fixes for the following PostgreSQL community security issues:
 - [CVE-2020-25694](#)
 - [CVE-2020-25695](#)
 - [CVE-2020-25696](#)

Additional improvements and enhancements

- None

Aurora PostgreSQL release 3.2.6

You can find the following improvements in this release.

Critical stability enhancements

- None

High priority stability enhancements

- Fixed a bug in Aurora PostgreSQL replication that might result in the error message, ERROR: MultiXactId nnnn has not been created yet -- apparent wraparound.

Additional improvements and enhancements

- Fixed a bug that in rare cases caused brief read replica unavailability when storage volume grew.
- Aurora PostgreSQL Serverless now supports execution of queries on all connections during a scale event.
- Fixed a bug in Aurora PostgreSQL Serverless where a leaked lock resulted in a prolonged scale event.
- Fixed a bug where the `aurora_replica_status` function showed truncated server identifiers.
- Fixed a bug in Aurora PostgreSQL Serverless where connections being migrated during a scale event disconnected with the message: "ERROR: could not open relation with OID"
- Fixed a small memory leak in a b-tree index that could lead to an out of memory condition.
- Fixed a bug in a GiST index that might result in an out of memory condition after promoting an Aurora Read Replica.
- Improved performance for Database Activity Streams.
- Fixed a bug in Database Activity Streams where customers were not notified when an outage ended.
- Fixed a bug in the `aws_s3` extension for pre-signed URL handling that could have resulted in the error message S3 bucket names with a period (.) are not supported.
- Fixed a bug in the `aws_s3` extension where incorrect error handling could lead to failures during the import process.
- Fixed a bug in the `aws_s3` extension where an import may be blocked indefinitely if an exclusive lock was taken on the relation prior to beginning the operation.

Aurora PostgreSQL release 3.2.4

You can find the following improvements in this release.

Critical stability enhancements

1. Fixed a bug that appears when the `NOT EXISTS` operator incorrectly returns TRUE, which can only happen when the following unusual set of circumstances occurs:
 - A query is using the `NOT EXISTS` operator.
 - The column or columns being evaluated against the outer query in the `NOT EXISTS` subquery contain a NULL value.
 - There isn't another predicate in the subquery that removes the need for the evaluation of the NULL values.
 - The filter used in the subquery does not use an index seek for its execution.
 - The operator isn't converted to a join by the query optimizer.

Aurora PostgreSQL release 3.2.3

You can find the following improvements in this release.

Critical stability enhancements

- None

High priority stability enhancements

- None

Additional improvements and enhancements

- Fixed a bug in Aurora PostgreSQL Serverless where queries that ran on previously idle connections got delayed until the scale operation completed.
- Fixed a bug that might cause brief unavailability for heavy subtransaction workloads when multiple reader instances restart or rejoin the cluster.

Aurora PostgreSQL release 3.2.2

You can find the following improvements in this release.

Critical stability enhancements

- Fixed a bug related to heap page extend that in rare cases resulted in longer recovery time and impacted availability.

High priority stability enhancements

- Fixed a bug in Aurora Global Database that could cause delays in upgrading the database engine in a secondary region. For more information, see [Using Amazon Aurora global databases \(p. 225\)](#).
- Fixed a bug that in rare cases caused delays in upgrading a database to engine version 11.7.

Additional improvements and enhancements

- Fixed a bug where the database might be unavailable briefly due to error handling in database storage growth.
- Fixed a bug where a SELECT query might incorrectly return the error, Attempting to read past EOF of relation rrrr. blockno=bbb nblocks=nnn.
- Fixed a bug where an Aurora PostgreSQL Serverless DB cluster might return the following error after a scaling event: ERROR: prepared statement "S_6" already exists.

Aurora PostgreSQL release 3.2.1

New features

- Added support for Amazon Aurora PostgreSQL Global Database. For more information, see [Using Amazon Aurora global databases \(p. 225\)](#).
- Added the ability to configure the recovery point objective (RPO) of a global database for Aurora PostgreSQL. For more information, see [Managing RPOs for Aurora PostgreSQL-based global databases \(p. 272\)](#).

You can find the following improvements in this release.

Critical stability enhancements

None.

High priority stability enhancements

- Improved performance and availability of read instances when applying DROP TABLE and TRUNCATE TABLE operations.
- Fixed a small but continuous memory leak in a diagnostic module that could lead to an out-of-memory condition on smaller DB instance types.
- Fixed a bug in the PostGIS extension which could lead to a database restart. This has been reported to the PostGIS community as <https://trac.osgeo.org/postgis/ticket/4646>.
- Fixed a bug where read requests might stop responding due to incorrect error handling in the storage engine.
- Fixed a bug that fails for some queries and results in the message ERROR: found xmin xxxxxxx from before relfrozenid yyyyyyy. This could occur following the promotion of a read instance to a write instance.
- Fixed a bug where an Aurora serverless DB cluster might crash while rolling back a scale attempt.

Additional improvements and enhancements

- Improved performance for queries that read many rows from storage.
- Improved performance and availability of reader DB instances during heavy read workload.
- Enabled correlated IN and NOT IN subqueries to be transformed to joins when possible.
- Improved the filtering estimation for enhanced semi-join filter pushdown by using multi-column statistics or indexes when available.
- Improved read performance of the pg_prewarm extension.
- Fixed a bug where an Aurora serverless DB cluster might report the message ERROR: incorrect binary data format in bind parameter ... following a scale event.
- Fixed a bug where a serverless DB cluster might report the message ERROR: insufficient data left in message following a scale event.
- Fixed a bug where an Aurora serverless DB cluster can experience prolonged or failed scale attempts.
- Fixed a bug that resulted in the message ERROR: could not create file "base/xxxxxxxx/yyyyyyy" as a previous version still exists on disk: Success. Please contact AWS customer support. This can occur during object creation after PostgreSQL's 32-bit object identifier has wrapped around.
- Fixed a bug where the write-ahead-log (WAL) segment files for PostgreSQL logical replication were not deleted when changing the wal_level value from logical to replica.
- Fixed a bug in the pg_hint_plan extension where a multi-statement query could lead to a crash when enable_hint_table is enabled. This is tracked in the PostgreSQL community as https://github.com/oss-c-db/pg_hint_plan/issues/25.
- Fixed a bug where JDBC clients might report the message java.io.IOException: Unexpected packet type: 75 following a scale event in an Aurora serverless DB cluster.
- Fixed a bug in PostgreSQL logical replication that resulted in the message ERROR: snapshot reference is not owned by resource owner TopTransaction.
- Changed the following extensions:
 - Updated orafce to version 3.8
 - Updated pgTAP to version 1.1
- Provided support for fault injection queries.

PostgreSQL 11.6, Aurora PostgreSQL release 3.1

This release of Aurora PostgreSQL is compatible with PostgreSQL 11.6. For more information about the improvements in PostgreSQL 11.6, see [PostgreSQL release 11.6](#).

This release contains multiple critical stability enhancements. Amazon highly recommends upgrading your Aurora PostgreSQL clusters that use older PostgreSQL 11 engines to this release.

Patch releases

- [Aurora PostgreSQL release 3.1.4 \(p. 1623\)](#)
- [Aurora PostgreSQL release 3.1.3 \(p. 1623\)](#)
- [Aurora PostgreSQL release 3.1.2 \(p. 1624\)](#)
- [Aurora PostgreSQL release 3.1.1 \(p. 1624\)](#)
- [Aurora PostgreSQL release 3.1.0 \(p. 1625\)](#)

[Aurora PostgreSQL release 3.1.4](#)

You can find the following improvements in this release.

Critical stability enhancements

- None

High priority stability enhancements

- Backported fixes for the following PostgreSQL community security issues:
 - [CVE-2020-25694](#)
 - [CVE-2020-25695](#)
 - [CVE-2020-25696](#)

Additional improvements and enhancements

- None

[Aurora PostgreSQL release 3.1.3](#)

New features

- Aurora PostgreSQL now supports the PostgreSQL `vacuum_truncate` storage parameter to manage vacuum truncation for specific tables. Set this [storage parameter](#) to false for a table to prevent the `VACUUM` SQL command from truncating the table's trailing empty pages.

Critical stability enhancements

- None

High priority stability enhancements

- Fixed a bug where reads from storage might stop responding due to incorrect error handling.

Additional improvements and enhancements

- None

Aurora PostgreSQL release 3.1.2

This release contains a critical stability enhancement. Amazon highly recommends updating your older Aurora PostgreSQL 11-compatible clusters to this release.

Critical stability enhancements

- Fixed a bug in which a reader DB instance might temporarily use stale data. This could lead to wrong results such as too few or too many rows. This error is not persisted on storage, and will clear when the database page containing the row has been evicted from cache. This can happen when the primary DB instance enters a transaction snapshot overflow due to having more than 64 subtransactions in a single transaction. Applications susceptible to this bug include those that use SQL savepoints or PostgreSQL exception handlers with more than 64 subtransactions in the top transaction.

High priority stability enhancements

- Fixed a bug that might cause a reader DB instance to crash causing unavailability while attempting to join the DB cluster. This can happen in some cases when the primary DB instance has a transaction snapshot overflow due to a high number of subtransactions. In this situation the reader DB instance will be unable to join until the snapshot overflow has cleared.

Additional improvements and enhancements

- Fixed a bug that prevented Performance Insights from determining the query ID of a running statement.

Aurora PostgreSQL release 3.1.1

You can find the following improvements in this release.

Critical stability enhancements

- Fixed a bug in which the DB instance might be briefly unavailable due to the self-healing function of the underlying storage.

High priority stability enhancements

- Fixed a bug in which the database engine might crash causing unavailability. This occurred while scanning an included, non-key column of a B-Tree index. This only applies to PostgreSQL 11 "included column" indexes.
- Fixed a bug that might cause the database engine to crash causing unavailability. This occurred if a newly established database connection encountered a resource exhaustion-related error during initialization after successful authentication.

Additional improvements and enhancements

- Provided a fix for the pg_hint_plan extension that could lead the database engine to crash causing unavailability. The open source issue can be tracked at https://github.com/ossc-db/pg_hint_plan/pull/45.
- Fixed a bug where SQL of the form `ALTER FUNCTION ... OWNER TO ...` incorrectly reported `ERROR: improper qualified name (too many dotted names)`.
- Improved the performance of GIN index vacuum via prefetching.

- Fixed a bug in open source PostgreSQL that could lead to a database engine crash causing unavailability. This occurred during parallel B-Tree index scans. This issue has been reported to the PostgreSQL community.
- Improved the performance of in-memory B-Tree index scans.

For information about extensions and modules, see [Extensions supported for Aurora PostgreSQL 11.x \(p. 1674\)](#).

Aurora PostgreSQL release 3.1.0

You can find the following new features and improvements in this engine version.

New features

1. Support for exporting data to Amazon S3. For more information, see [Exporting data from an Aurora PostgreSQL DB cluster to Amazon S3 \(p. 1450\)](#).
2. Support for Amazon Aurora Machine Learning. For more information, see [Using machine learning \(ML\) with Aurora PostgreSQL \(p. 1494\)](#).
3. SQL processing enhancements include:
 - Optimizations for NOT IN with the `apg_enable_not_in_transform` parameter.
 - Semi-join filter pushdown enhancements for hash joins with the `apg_enable_semijoin_push_down` parameter.
 - Optimizations for redundant inner join removal with the `apg_enable_remove_redundant_inner_joins` parameter.
 - Improved ANSI compatibility options with the `ansi_constraint_trigger_ordering`, `ansi_force_foreign_key_checks` and `ansi_qualified_update_set_target` parameters.

For more information, see [Amazon Aurora PostgreSQL parameters \(p. 1547\)](#).

4. New and updated PostgreSQL extensions include:
 - The new `aws_ml` extension. For more information, see [Using machine learning \(ML\) with Aurora PostgreSQL \(p. 1494\)](#).
 - The new `aws_s3` extension. For more information, see [Exporting data from an Aurora PostgreSQL DB cluster to Amazon S3 \(p. 1450\)](#).
 - Updates to the `apg_plan_mgmt` extension. For more information, see [Managing query execution plans for Aurora PostgreSQL \(p. 1460\)](#)

Critical stability enhancements

1. Fixed a bug related to creating B-tree indexes on temporary tables that in rare cases might result in longer recovery time, and impact availability.
2. Fixed a bug related to replication when Aurora PostgreSQL is acting as a physical replica of an RDS PostgreSQL instance. In rare cases, this bug causes a log write failure that might result in longer recovery time, and impact availability.
3. Fixed a bug related to handling of reads with high I/O latency that in rare cases might result in longer recovery time, and impact availability.

High priority stability enhancements

1. Fixed a bug related to logical replication in which `wal` segments are not properly removed from storage. This can result in storage bloat. To monitor this, view the `TransactionLogDiskUsage` parameter.
2. Fixed multiple bugs, which cause Aurora to crash during prefetch operations on Btree indexes.

3. Fixed a bug in which an Aurora restart might time out when logical replication is used.
4. Enhanced the validation checks performed on data blocks in the buffer cache. This improves Aurora's detection of inconsistency.

Additional improvements and enhancements

1. The query plan management extension `apg_plan_mgmt` has an improved algorithm for managing plan generation for highly partitioned tables.
2. Reduced startup time on instances with large caches via improvements in the buffer cache recovery algorithm.
3. Improved the performance of the read-node-apply process under high transaction rate workloads by using changes to PostgreSQL `LWLock` prioritization. These changes prevent starvation of the read-node-apply process while the PostgreSQL `ProcArray` is under heavy contention.
4. Improved handling of batch reads during vacuum, table scans, and index scans. This results in greater throughput and lower CPU consumption.
5. Fixed a bug in which a read node might crash during the replay of a PostgreSQL `SIRU-truncate` operation.
6. Fixed a bug where in rare cases, database writes might stall following an error returned by one of the six copies of an Aurora log record.
7. Fixed a bug related to logical replication where an individual transaction larger than 1 GB in size might result in an engine crash.
8. Fixed a memory leak on read nodes when cluster cache management is enabled.
9. Fixed a bug in which importing an RDS PostgreSQL snapshot might stop responding if the source snapshot contains a large number of unlogged relations.
10. Fixed a bug in which the Aurora storage daemon might crash under heavy I/O load.
11. Fixed a bug related to `hot_standby_feedback` for read nodes in which the read node might report the wrong transaction id epoch to the write node. This can cause the write node to ignore the `hot_standby_feedback` and invalidate snapshots on the read node.
12. Fixed a bug in which storage errors that occur during `CREATE DATABASE` statements are not properly handled. The bug left the resulting database inaccessible. The correct behavior is to fail the database creation and return the appropriate error to the user.
13. Improved handling of PostgreSQL snapshot overflow when a read node attempts to connect to a write node. Prior to this change, if the write node was in a snapshot overflow state, the read node was unable to join. A message appeared in the PostgreSQL log file in the form `DEBUG: recovery snapshot waiting for non-overflowed snapshot or until oldest active xid on standby is at least xxxxxxxx (now yyyyyyyy)`. A snapshot overflow occurs when an individual transaction has created over 64 subtransactions.
14. Fixed a bug related to common table expressions in which an error is incorrectly raised when a NOT IN class exists in a CTE. The error is `CTE with NOT IN fails with ERROR: could not find CTE CTE-Name`.
15. Fixed a bug related to an incorrect `last_error_timestamp` value in the `aurora_replica_status` table.
16. Fixed a bug to avoid populating shared buffers with blocks belonging to temporary objects. These blocks correctly reside in PostgreSQL backend local buffers.
17. Changed the following extensions:
 - Updated `pg_hint_plan` to version 1.3.4.
 - Added `plprofiler` version 4.1.
 - Added `pgTAP` version 1.0.0.

PostgreSQL 11.4, Aurora PostgreSQL release 3.0 (unsupported)

Note

The PostgreSQL engine version 11.4 with the Aurora PostgreSQL release 3.0 is no longer supported. To upgrade, see [Upgrading the PostgreSQL DB engine for Aurora PostgreSQL \(p. 1681\)](#).

This release of Aurora PostgreSQL is compatible with PostgreSQL 11.4. For more information about the improvements in PostgreSQL 11.4, see [PostgreSQL release 11.4](#).

You can find the following improvements in this release.

Improvements

1. This release contains all fixes, features, and improvements present in [Aurora PostgreSQL release 2.3.5 \(p. 1644\)](#).
2. Partitioning – Partitioning improvements include support for hash partitioning, enabling creation of a default partition, and dynamic row movement to another partition based on the key column update.
3. Performance – Performance improvements include parallelism while creating indexes, materialized views, hash joins, and sequential scans to make the operations perform better.
4. Stored procedures – SQL stored procedures now added support for embedded transactions.
5. Autovacuum improvements – To provide valuable logging, the parameter `rds.force_autovacuum_logging` is ON by default in conjunction with the `log_autovacuum_min_duration` parameter set to 10 seconds. To increase autovacuum effectiveness, the values for the `autovacuum_max_workers` and `autovacuum_vacuum_cost_limit` parameters are computed based on host memory capacity to provide larger default values.
6. Improved transaction timeout – The parameter `idle_in_transaction_session_timeout` is set to 24 hours. Any session that has been idle more than 24 hours is terminated.
7. The `tsearch2` module is no longer supported – If your application uses `tsearch2` functions, update it to use the equivalent functions provided by the core PostgreSQL engine. For more information about the `tsearch2` module, see [PostgreSQL tsearch2](#).
8. The `chkpass` module is no longer supported – For more information about the `chkpass` module, see [PostgreSQL chkpass](#).
9. Updated the following extensions:
 - `address_standardizer` to version 2.5.1
 - `address_standardizer_data_us` to version 2.5.1
 - `btree_gin` to version 1.3
 - `citext` to version 1.5
 - `cube` to version 1.4
 - `hstore` to version 1.5
 - `ip4r` to version 2.2
 - `isn` to version 1.2
 - `orafce` to version 3.7
 - `pg_hint_plan` to version 1.3.4
 - `pg_prewarm` to version 1.2
 - `pg_repack` to version 1.4.4
 - `pg_trgm` to version 1.4
 - `pgaudit` to version 1.3
 - `pgrouting` to version 2.6.1
 - `pgtap` to version 1.0.0
 - `plcoffee` to version 2.3.8

- `plls` to version 2.3.8
- `plv8` to version 2.3.8
- `postgis` to version 2.5.1
- `postgis_tiger_geocoder` to version 2.5.1
- `postgis_topology` to version 2.5.1
- `rds_activity_stream` to version 1.3

PostgreSQL 10.19

This release of Aurora PostgreSQL is compatible with PostgreSQL 10.19. For more information about the improvements in PostgreSQL 10.19, see [PostgreSQL release 10.19](#).

Aurora PostgreSQL release 10.19

Critical stability enhancements

- Fixed a bug where logical replication may hang resulting in the replay falling behind on the read node. The instance may eventually restart.

Additional improvements and enhancements

- Fixed a buffer cache bug that could cause brief periods of unavailability.
 - Fixed a bug in the `apg_plan_mgmt` extension where an index based plan was not being enforced.
 - Fixed a bug in the `pg_logical` extension that could cause brief periods of unavailability due to improper handling of NULL arguments.
 - Fixed an issue where orphaned files caused major version upgrades to fail.
 - Fixed incorrect Aurora Storage Daemon log write metrics.
 - Fixed multiple bugs that could result in WAL replay falling behind and eventually causing the reader instances to restart.
 - Improved the Aurora buffer cache page validation on reads.
 - Improved the Aurora storage metadata validation.
 - Updated the `pg_hint_plan` extension to v1.3.6.
-
- For information about extensions and modules, see [Extensions supported for Aurora PostgreSQL 10.x \(p. 1676\)](#).

PostgreSQL 10.18

This release of Aurora PostgreSQL is compatible with PostgreSQL 10.18. For more information about the improvements in PostgreSQL 10.18, see [PostgreSQL release 10.18](#).

Critical stability enhancements

- Fixed an issue where, in rare circumstances, a data cache of a read node may be inconsistent following a restart of that node.

High priority stability enhancements

- Fixed an issue where queries may become unresponsive due to I/O resource exhaustion triggered by prefetch.

- Fixed an issue where Aurora may panic following a major version update with the message: "PANIC: could not access status of next transaction id xxxxxxxx".

Additional improvements and enhancements

- Fixed an issue where read nodes restart due to a replication origin cache lookup failure.
- Fixed an issue where read queries may time out on read nodes during the replay of lazy truncation triggered by vacuum on the write node.
- Fixed an issue that causes Performance Insights to incorrectly set the backend type of a database connection.
- Fixed an issue where the `aurora_postgres_replica_status()` function returned stale or lagging CPU stats.
- Fixed an issue where, in rare cases, an Aurora Global Database secondary mirror cluster may restart due to a stall in the log apply process.
- Removed support for DES, 3DES and RC4 cipher suites.
- Updated PostGIS extension to version 3.1.4.
- Added support for `postgis_raster` extension version 3.1.4.

PostgreSQL 10.17, Aurora PostgreSQL release 2.9

This release of Aurora PostgreSQL is compatible with PostgreSQL 10.17. For more information about the improvements in PostgreSQL 10.17, see [PostgreSQL release 10.17](#).

Patch releases

- [Aurora PostgreSQL 2.9.1 \(p. 1629\)](#)
- [Aurora PostgreSQL release 2.9 \(p. 1630\)](#)

[Aurora PostgreSQL 2.9.1](#)

Critical stability enhancements

- Fixed an issue where, in rare circumstances, a data cache of a read node may be inconsistent following a restart of that node.

High priority stability enhancements

- Fixed an issue where queries may become unresponsive due to I/O resource exhaustion triggered by prefetch.
- Fixed an issue where Aurora may panic following a major version update with the message: "PANIC: could not access status of next transaction id xxxxxxxx".

Additional improvements and enhancements

- Fixed an issue where read nodes restart due to a replication origin cache lookup failure.
- Fixed an issue where in rare cases, an Aurora Global Database secondary mirror cluster may restart due to a stall in the log apply process.
- Fixed an issue that causes Performance Insights to incorrectly set the backend type of a database connection.
- Fixed an issue where orphaned files caused failed translations in read codepaths during or after major version upgrade.

- Fixed multiple issues in the Aurora storage daemon that could lead to brief periods of unavailability when specific network configurations are used.
- Fixed an out of memory crash issue with Aurora storage daemon that leads to writer node restart. This also reduces the overall system memory consumption.

Aurora PostgreSQL release 2.9

High priority stability enhancements

1. Fixed an issue where creating a database from an existing template database with tablespace resulted in an error with the message `ERROR: could not open file pg_tblspc/...: No such file or directory.`
2. Fixed an issue where, in rare cases, an Aurora replica may be unable to start when a large number of PostgreSQL subtransactions (i.e. SQL savepoints) have been used.
3. Fixed an issue where, in rare circumstances, read results may be inconsistent for repeated read requests on replica nodes.

Additional improvements and enhancements

1. Upgraded OpenSSL to 1.1.1k.
2. Reduced CPU usage and memory consumption of the WAL apply process on Aurora replicas for some workloads.
3. Improved safety checks in the write path to detect incorrect writes to metadata.
4. Improved security by removing 3DES and other older ciphers for SSL/TLS connections.
5. Fixed an issue where a duplicate file entry can prevent the Aurora PostgreSQL engine from starting up.
6. Fixed an issue that could cause temporary unavailability under heavy workloads.
7. Added back ability to use a leading forward slash in the S3 path during S3 import.
8. Updated the `orafce` extension to version 3.16.
9. Updated the `PostGIS` extension to version 2.4.7.

PostgreSQL 10.16, Aurora PostgreSQL release 2.8

This release of Aurora PostgreSQL is compatible with PostgreSQL 10.16. For more information about the improvements in PostgreSQL 10.16, see [PostgreSQL release 10.16](#).

Aurora PostgreSQL release 2.8.0

High priority stability enhancements

1. Fixed a bug where in rare cases a reader had inconsistent results when it restarted while a transaction with more than 64 subtransactions was being processed.
2. Backported fixes for the following PostgreSQL community security issues:
 - [CVE-2021-32027](#)
 - [CVE-2021-32028](#)
 - [CVE-2021-32029](#)

Additional improvements and enhancements

1. Fixed a bug where the database could not be started when there were many relations in memory-constrained environments.

2. Fixed a bug in the `apg_plan_mgmt` extension that could cause brief periods of unavailability due to an internal buffer overflow.
3. Fixed a bug on reader nodes that could cause brief periods of unavailability during WAL replay.
4. Fixed a bug in the `rds_activity_stream` extension that caused an error during startup when attempting to log audit events.
5. Fixed a bug that prevented minor version updates of an Aurora global database cluster.
6. Fixed bugs in the `aurora_replica_status` function where rows were sometimes partially populated and some values such as Replay Latency, and CPU usage were always 0.
7. Fixed a bug where the database engine would attempt to create shared memory segments larger than the instance total memory and fail repeatedly. For example, attempts to create 128 GiB shared buffers on a db.r5.large instance would fail. With this change, requests for total shared memory allocations larger than the instance memory allow setting the instance to incompatible parameters.
8. Added logic to clean up unnecessary `pg_wal` temporary files on a database startup.
9. Fixed a bug that reported ERROR: `rds_activity_stream` stack item 2 not found on top - cannot pop when attempting to create the `rds_activity_stream` extension.
10. Fixed a bug that could cause the error failed to build any 3-way joins in a correlated `IN` subquery under an `EXISTS` subquery.
11. Fixed a bug that could cause brief periods of unavailability due to running out of memory when creating the `postgis` extension with `pgAudit` enabled.
12. Fixed a bug when using outbound logical replication to synchronize changes to another database that could fail with an error message like ERROR: could not map filenode "base/16395/228486645" to relation OID.
13. Fixed a bug where the `rds_ad` role wasn't created after upgrading from a version of Aurora PostgreSQL that doesn't support Microsoft Active Directory authentication.
14. Added btree page checks to detect tuple metadata inconsistency.
15. Fixed a bug in asynchronous buffer reads that could cause brief periods of unavailability on reader nodes during WAL replay.

PostgreSQL 10.14, Aurora PostgreSQL release 2.7

This release of Aurora PostgreSQL is compatible with PostgreSQL 10.14. For more information about the improvements in PostgreSQL 10.14, see [PostgreSQL release 10.14](#).

Patch releases

- [Aurora PostgreSQL release 2.7.3 \(p. 1631\)](#)
- [Aurora PostgreSQL release 2.7.2 \(p. 1632\)](#)
- [Aurora PostgreSQL release 2.7.1 \(p. 1632\)](#)
- [Aurora PostgreSQL release 2.7.0 \(p. 1632\)](#)

Aurora PostgreSQL release 2.7.3

High priority stability enhancements

1. Provided a patch for PostgreSQL community security issues CVE-2021-32027, CVE-2021-32028 and CVE-2021-32029.

Additional improvements and enhancements

1. Fixed a bug in the `aws_s3` extension to allow import of objects with leading forward slashes in the object identifier.

2. Fixed a bug in the `rds_activity_stream` extension that caused an error during startup when attempting to log audit events.
3. Fixed a bug that returned an `ERROR` when attempting to create the `rds_activity_stream` extension.
4. Fixed a bug that could cause brief periods of unavailability due to running out of memory when creating the `postgis` extension with `pgAudit` enabled.
5. Fixed multiple issues in the Aurora storage daemon that could lead to brief periods of unavailability when specific network configurations are used.

Aurora PostgreSQL release 2.7.2

High priority stability enhancements

1. Fixed a bug where a reader node might render an extra or missing row if the reader restarted while the writer node is processing a long transaction with more than 64 subtransactions.

Additional improvements and enhancements

1. Fixed a bug that could lead to intermittent unavailability due to the rotation of network encryption keys.
2. Fixed a bug where a large S3 import with thousands of clients can cause one or more of the import clients to stop responding.

Aurora PostgreSQL release 2.7.1

Critical stability enhancements

1. Fixed a bug that caused a read replica to unsuccessfully restart repeatedly in rare cases.
2. Fixed a bug where a cluster became unavailable when attempting to create more than 16 read replicas or Aurora global database secondary AWS Regions. The cluster became available again when the new read replica or secondary AWS Region was removed.

Additional improvements and enhancements

1. Fixed a bug that when under heavy load, snapshot import, COPY import, or S3 import stopped responding in rare cases.
2. Fixed a bug where a read replica might not join the cluster when the writer was very busy with a write-intensive workload.
3. Fixed a bug that caused a cluster to take several minutes to restart if a logical replication stream was terminated while handling many complex transactions.
4. Disallowed the use of both IAM and Kerberos authentication for the same user.

Aurora PostgreSQL release 2.7.0

Critical stability enhancements

- None

High priority stability enhancements

1. Backported fixes for the following PostgreSQL community security issues:

- [CVE-2020-25694](#)
 - [CVE-2020-25695](#)
 - [CVE-2020-25696](#)
2. Fixed a bug in Aurora PostgreSQL replication that could result in the error message ERROR: MultiXactId nnnn has not been created yet -- apparent wraparound.
 3. Fixed a bug where in some cases, DB clusters that have logical replication enabled did not remove truncated WAL segment files from storage. This resulted in volume size growth.
 4. Fixed a bug in the pg_stat_statements extension that caused excessive CPU consumption.

Additional improvements and enhancements

1. Improved the asynchronous mode performance of database activity streams.
2. Aurora Serverless v1 for PostgreSQL now supports query execution on all connections during a scale event.
3. Reduced the delay when publishing to CloudWatch the `xpo_lag_in_msec` metric for Aurora global database clusters.
4. Fixed a bug in Serverless clusters where transaction processing was unnecessarily suspended for long periods when creating a scale point.
5. Fixed a bug in Aurora Serverless v1 for PostgreSQL where a leaked lock resulted in a prolonged scale event.
6. Fixed a bug in Aurora Serverless v1 for PostgreSQL where connections being migrated during a scale event was disconnected with the following message: ERROR: could not open relation with OID ...
7. Aurora PostgreSQL no longer falls behind on a read node when the backend is blocked writing to the database client.
8. Fixed a bug that in rare cases caused a brief period of unavailability on a read replica when the storage volume grew.
9. Fixed a bug when creating a database that could return the following error: ERROR: could not create directory on local disk
- 10Fixed a bug where in some cases replaying `XLOG_BTREE_REUSE_PAGE` records on Aurora reader instances caused unnecessary replay lag.
- 11Fixed a bug in the `GIST` index that could result in an out of memory condition after promoting an Aurora read replica.
- 12Fixed a bug where the `aurora_replica_status` function showed truncated server identifiers.
- 13Fixed an S3 import bug that reported ERROR: HTTP 403. Permission denied when importing data from a file inside an S3 subfolder.
- 14Fixed a bug in the `aws_s3` extension for pre-signed URL handling that could result in the error message S3 bucket names with a period (.) are not supported.
- 15Fixed a bug in the `aws_s3` extension where an import might be blocked indefinitely if an exclusive lock was taken on the relation prior to beginning the operation.
- 16Fixed a bug related to replication when Aurora PostgreSQL is acting as a physical replica of an RDS PostgreSQL instance that uses `GIST` indexes. In rare cases, this bug caused a brief period of unavailability after promoting the Aurora cluster.
- 17Fixed a bug in database activity streams where customers were not notified of the end of an outage.

PostgreSQL 10.13, Aurora PostgreSQL release 2.6

This release of Aurora PostgreSQL is compatible with PostgreSQL 10.13. For more information about the improvements in PostgreSQL 10.13, see [PostgreSQL release 10.13](#).

Patch releases

- [Aurora PostgreSQL release 2.6.2 \(p. 1634\)](#)
- [Aurora PostgreSQL release 2.6.1 \(p. 1635\)](#)
- [Aurora PostgreSQL release 2.6.0 \(p. 1635\)](#)

Aurora PostgreSQL release 2.6.2

Critical stability enhancements

1. None

High priority stability enhancements

1. Fixed a bug in Aurora PostgreSQL replication that could result in the error message ERROR: MultiXactId nnnn has not been created yet -- apparent wraparound.
2. Fixed a bug where in some cases, DB clusters that have logical replication enabled did not remove truncated WAL segment files from storage. This resulted in volume size growth.
3. Fixed an issue with creating a global database cluster in a secondary region.
4. Backported fixes for the following PostgreSQL community security issues:
 - [CVE-2020-25694](#)
 - [CVE-2020-25695](#)
 - [CVE-2020-25696](#)
5. Fixed a bug in the pg_stat_statements extension that caused excessive CPU consumption.

Additional improvements and enhancements

1. Aurora PostgreSQL no longer falls behind on a read node when the backend is blocked writing to the database client.
2. Reduced the delay when publishing to CloudWatch the `rpo_lag_in_msec` metric for Aurora global database clusters.
3. Fixed a bug where a `DROP DATABASE` statement didn't remove any relation files.
4. Fixed a bug where in some cases replaying `XLOG_BTREE_REUSE_PAGE` records on Aurora reader instances caused unnecessary replay lag.
5. Fixed a small memory leak in a b-tree index that could lead to an out of memory condition.
6. Fixed a bug in the `aurora_replica_status()` function where the `server_id` field was sometimes truncated.
7. Fixed a bug where a log record was incorrectly processed causing the Aurora replica to crash.
8. Fixed an S3 import bug that reported ERROR: HTTP 403. Permission denied when importing data from a file inside an S3 subfolder.
9. Improved performance of the asynchronous mode for database activity streams.
10. Fixed a bug in the `aws_s3` extension that could result in the error message S3 bucket names with a period (.) are not supported.
11. Fixed a race condition that caused valid imports to intermittently fail.
12. Fixed a bug related to replication when Aurora PostgreSQL is acting as a physical replica of an RDS PostgreSQL instance that uses GiST indexes. In rare cases, this bug caused a brief period of unavailability after promoting the Aurora DB cluster.
13. Fixed a bug in the `aws_s3` extension where an import may be blocked indefinitely if an exclusive lock was taken on the relation prior to beginning the operation.

Aurora PostgreSQL release 2.6.1

You can find the following improvements in this release.

Critical stability enhancements

1. Fixed a bug that appears when the `NOT EXISTS` operator incorrectly returns TRUE, which can only happen when the following unusual set of circumstances occurs:
 - A query is using the `NOT EXISTS` operator.
 - The column or columns being evaluated against the outer query in the `NOT EXISTS` subquery contain a NULL value.
 - There isn't another predicate in the subquery that removes the need for the evaluation of the NULL values.
 - The filter used in the subquery does not use an index seek for its execution.
 - The operator isn't converted to a join by the query optimizer.

Aurora PostgreSQL release 2.6.0

You can find the following improvements in this release.

New features

1. Added support for the RDKit extension version 3.8.

The RDKit extension provides modeling functions for cheminformatics. Cheminformatics is storing, indexing, searching, retrieving, and applying information about chemical compounds. For example, with the RDKit extension you can construct models of molecules, search for molecular structures, and read or create molecules in various notations. You can also perform research on data loaded from the [ChEMBL website](#) or a SMILES file. The Simplified Molecular Input Line Entry System (SMILES) is a typographical notation for representing molecules and reactions. For more information, see [The RDKit database cartridge](#) in the RDKit documentation.

2. Added support for the pglogical extension version 2.2.2.

The pglogical extension is a logical streaming replication system that provides additional features beyond what's available in PostgreSQL native logical replication. Features include conflict handling, row filtering, DDL/sequence replication and delayed apply. You can use the pglogical extension to set up replication between Aurora PostgreSQL clusters, between RDS PostgreSQL and Aurora PostgreSQL, and with PostgreSQL databases running outside of RDS.

3. Aurora dynamically resizes your cluster storage space. With dynamic resizing, the storage space for your Aurora DB cluster automatically decreases when you remove data from the DB cluster. For more information, see [Storage scaling \(p. 396\)](#).

Note

The dynamic resizing feature is being deployed in phases to the AWS Regions where Aurora is available. Depending on the Region where your cluster is, this feature might not be available yet. For more information, see [the What's New announcement](#).

Critical stability enhancements

1. Fixed a bug related to heap page extend that in rare cases resulted in longer recovery time and impacted availability.

High priority stability enhancements

1. Fixed a bug when upgrading Aurora Global Database clusters from 10.11.
2. Fixed a bug in Aurora Global Database that could cause delays in upgrading the database engine in a secondary AWS Region. For more information, see [Using Amazon Aurora global databases \(p. 225\)](#).
3. Fixed a bug that in rare cases caused delays in upgrading a database to engine version 10.13.

Additional improvements and enhancements

1. Fixed a bug where the Aurora replica crashed when workloads with heavy subtransactions are made on the writer instance.
2. Fixed a bug where the writer instance crashed due to a memory leak and the depletion of memory used to track active transactions.
3. Fixed a bug that lead to a crash due to improper initialization when there is no free memory available during PostgreSQL backend startup.
4. Fixed a bug where an Aurora PostgreSQL Serverless DB cluster might return the following error after a scaling event: ERROR: prepared statement "S_6" already exists.
5. Fixed an out-of-memory problem when issuing the `CREATE EXTENSION` command with PostGIS when Database Activity Streams enabled.
6. Fixed a bug where a `SELECT` query might incorrectly return the error Attempting to read past EOF of relation rrrr. blockno=bbb nblocks=nnn.
7. Fixed a bug where the database might be unavailable briefly due to error handling in database storage growth.
8. Fixed a bug in Aurora PostgreSQL Serverless where queries that executed on previously idle connections got delayed until the scale operation completed.
9. Fixed a bug where an Aurora PostgreSQL DB cluster with Database Activity Streams enabled might report the beginning of a potential loss window for activity records, but does not report the restoration of connectivity.

PostgreSQL 10.12, Aurora PostgreSQL release 2.5

This release of Aurora PostgreSQL is compatible with PostgreSQL 10.12. For more information about the improvements in PostgreSQL 10.12, see [PostgreSQL release 10.12](#).

Patch releases

- [Aurora PostgreSQL release 2.5.7 \(p. 1636\)](#)
- [Aurora PostgreSQL release 2.5.6 \(p. 1637\)](#)
- [Aurora PostgreSQL release 2.5.4 \(p. 1637\)](#)
- [Aurora PostgreSQL release 2.5.3 \(p. 1638\)](#)
- [Aurora PostgreSQL release 2.5.2 \(p. 1638\)](#)
- [Aurora PostgreSQL release 2.5.1 \(p. 1639\)](#)

Aurora PostgreSQL release 2.5.7

You can find the following improvements in this release.

Critical stability enhancements

- None

High priority stability enhancements

1. Backported fixes for the following PostgreSQL community security issues:
 - [CVE-2020-25694](#)
 - [CVE-2020-25695](#)
 - [CVE-2020-25696](#)

Additional improvements and enhancements

- None

Aurora PostgreSQL release 2.5.6

You can find the following improvements in this release.

Critical stability enhancements

- None

High priority stability enhancements

1. Fixed a bug in Aurora PostgreSQL replication that might result in the error message, ERROR: MultiXactId nnnn has not been created yet -- apparent wraparound.

Additional improvements and enhancements

1. Fixed a bug that in rare cases caused brief read replica unavailability when storage volume grew.
2. Aurora PostgreSQL Serverless now supports execution of queries on all connections during a scale event.
3. Fixed a bug in Aurora PostgreSQL Serverless where a leaked lock resulted in a prolonged scale event.
4. Fixed a bug where the `aurora_replica_status` function showed truncated server identifiers.
5. Fixed a bug in Aurora PostgreSQL Serverless where connections being migrated during a scale event disconnected with the message: "ERROR: could not open relation with OID"
6. Fixed a bug in a GiST index that might result in an out of memory condition after promoting an Aurora Read Replica.
7. Improved performance for Database Activity Streams.
8. Fixed a bug in Database Activity Streams where customers were not notified when an outage ended.
9. Fixed a bug in the `aws_s3` extension for pre-signed URL handling that could have resulted in the error message S3 bucket names with a period (.) are not supported.
10. Fixed a bug in the `aws_s3` extension where incorrect error handling could lead to failures during the import process.
11. Fixed a bug in the `aws_s3` extension where an import may be blocked indefinitely if an exclusive lock was taken on the relation prior to beginning the operation.

Aurora PostgreSQL release 2.5.4

You can find the following improvements in this release.

Critical stability enhancements

1. Fixed a bug that appears when the `NOT EXISTS` operator incorrectly returns TRUE, which can only happen when the following unusual set of circumstances occurs:
 - A query is using the `NOT EXISTS` operator.
 - The column or columns being evaluated against the outer query in the `NOT EXISTS` subquery contain a NULL value.
 - There isn't another predicate in the subquery that removes the need for the evaluation of the NULL values.
 - The filter used in the subquery does not use an index seek for its execution.
 - The operator isn't converted to a join by the query optimizer.

Aurora PostgreSQL release 2.5.3

You can find the following improvements in this release.

Critical stability enhancements

- None

High priority stability enhancements

- None

Additional improvements and enhancements

1. Fixed a bug in Aurora PostgreSQL Serverless where queries that ran on previously idle connections got delayed until the scale operation completed.
2. Fixed a bug that might cause brief unavailability for heavy subtransaction workloads when multiple reader instances restart or rejoin the cluster.
3. Fixed a bug in Aurora PostgreSQL Global Database where upgrading a secondary cluster might result in failure due to incorrect checksum versioning. This might have required re-creating the secondary clusters.

Aurora PostgreSQL release 2.5.2

You can find the following improvements in this release.

Critical stability enhancements

1. Fixed a bug related to heap page extend that in rare cases resulted in longer recovery time and impacted availability.

High priority stability enhancements

1. Fixed a bug in Aurora Global Database that could cause delays in upgrading the database engine in a secondary region. For more information, see [Using Amazon Aurora global databases \(p. 225\)](#).
2. Fixed a bug that in rare cases caused delays in upgrading a database to engine version 10.12.

Additional improvements and enhancements

1. Fixed a bug where the database might be unavailable briefly due to error handling in database storage growth.
2. Fixed a bug where a SELECT query might incorrectly return the error, Attempting to read past EOF of relation rrrr. blockno=bbb nblocks=nnn.
3. Fixed a bug where an Aurora PostgreSQL Serverless DB cluster might return the following error after a scaling event: ERROR: prepared statement "S_6" already exists.

Aurora PostgreSQL release 2.5.1

New features

1. Added support for Amazon Aurora PostgreSQL Global Database. For more information, see [Using Amazon Aurora global databases \(p. 225\)](#).
2. Added the ability to configure the recovery point objective (RPO) of a global database for Aurora PostgreSQL. For more information, see [Managing RPOs for Aurora PostgreSQL-based global databases \(p. 272\)](#).

You can find the following improvements in this release.

Critical stability enhancements

None.

High priority stability enhancements

1. Improved performance and availability of read instances when applying DROP TABLE and TRUNCATE TABLE operations.
2. Fixed a small but continuous memory leak in a diagnostic module that could lead to an out-of-memory condition on smaller DB instance types.
3. Fixed a bug in the PostGIS extension which could lead to a database restart. This has been reported to the PostGIS community as <https://trac.osgeo.org/postgis/ticket/4646>.
4. Fixed a bug where read requests might stop responding due to incorrect error handling in the storage engine.
5. Fixed a bug that fails for some queries and results in the message ERROR: found xmin xxxxxx from before relfrozenid yyyyyyy. This could occur following the promotion of a read instance to a write instance.
6. Fixed a bug where an Aurora serverless DB cluster might crash while rolling back a scale attempt.

Additional improvements and enhancements

1. Improved performance for queries that read many rows from storage.
2. Improved performance and availability of reader DB instances during heavy read workload.
3. Enabled correlated IN and NOT IN subqueries to be transformed to joins when possible.
4. Improved read performance of the pg_prewarm extension.
5. Fixed a bug where an Aurora serverless DB cluster might report the message ERROR: incorrect binary data format in bind parameter ... following a scale event.
6. Fixed a bug where a serverless DB cluster might report the message ERROR: insufficient data left in message following a scale event.

7. Fixed a bug where an Aurora serverless DB cluster may experience prolonged or failed scale attempts.
8. Fixed a bug that resulted in the message ERROR: could not create file "base/xxxxxx/yyyyyyy" as a previous version still exists on disk: Success. Please contact AWS customer support. This can occur during object creation after PostgreSQL's 32-bit object identifier has wrapped around.
9. Fixed a bug where the write-ahead-log (WAL) segment files for PostgreSQL logical replication were not deleted when changing the wal_level value from logical to replica.
10. Fixed a bug in the pg_hint_plan extension where a multi-statement query could lead to a crash when enable_hint_table is enabled. This is tracked in the PostgreSQL community as https://github.com/osscc-db/pg_hint_plan/issues/25.
11. Fixed a bug where JDBC clients might report the message java.io.IOException: Unexpected packet type: 75 following a scale event in an Aurora serverless DB cluster.
12. Fixed a bug in PostgreSQL logical replication that resulted in the message ERROR: snapshot reference is not owned by resource owner TopTransaction.
13. Changed the following extensions:
 - Updated orafce to version 3.8

PostgreSQL 10.11, Aurora PostgreSQL release 2.4

This release of Aurora PostgreSQL is compatible with PostgreSQL 10.11. For more information about the improvements in PostgreSQL 10.11, see [PostgreSQL release 10.11](#).

This release contains multiple critical stability enhancements. Amazon highly recommends upgrading your Aurora PostgreSQL clusters that use older PostgreSQL 10 engines to this release.

Patch releases

- [Aurora PostgreSQL release 2.4.4 \(p. 1640\)](#)
- [Aurora PostgreSQL release 2.4.3 \(p. 1641\)](#)
- [Aurora PostgreSQL release 2.4.2 \(p. 1641\)](#)
- [Aurora PostgreSQL release 2.4.1 \(p. 1641\)](#)
- [Aurora PostgreSQL release 2.4.0 \(p. 1642\)](#)

Aurora PostgreSQL release 2.4.4

You can find the following improvements in this release.

Critical stability enhancements

- None

High priority stability enhancements

1. Backported fixes for the following PostgreSQL community security issues:
 - [CVE-2020-25694](#)
 - [CVE-2020-25695](#)
 - [CVE-2020-25696](#)

Additional improvements and enhancements

- None

Aurora PostgreSQL release 2.4.3

New features

1. Aurora PostgreSQL now supports the PostgreSQL `vacuum_truncate` storage parameter to manage vacuum truncation for specific tables. Set this [storage parameter](#) to false for a table to prevent the `VACUUM` SQL command from truncating the table's trailing empty pages.

Critical stability enhancements

- None

High priority stability enhancements

1. Fixed a bug where reads from storage might stop responding due to incorrect error handling.

Additional improvements and enhancements

- None

Aurora PostgreSQL release 2.4.2

You can find the following improvements in this release.

Critical stability enhancements

1. Fixed a bug in which a reader DB instance might temporarily use stale data. This could lead to wrong results such as too few or too many rows. This error is not persisted on storage, and will clear when the database page containing the row has been evicted from cache. This can happen when the primary DB instance enters a transaction snapshot overflow due to having more than 64 subtransactions in a single transaction. Applications susceptible to this bug include those that use SQL savepoints or PostgreSQL exception handlers with more than 64 subtransactions in the top transaction.

High priority stability enhancements

1. Fixed a bug that may cause a reader DB instance to crash causing unavailability while attempting to join the DB cluster. This can happen in some cases when the primary DB instance has a transaction snapshot overflow due to a high number of subtransactions. In this situation the reader DB instance will be unable to join until the snapshot overflow has cleared.

Additional improvements and enhancements

1. Fixed a bug that prevented Performance Insights from determining the query ID of a running statement.

Aurora PostgreSQL release 2.4.1

You can find the following improvements in this release.

Critical stability enhancements

1. Fixed a bug in which the DB instance might be briefly unavailable due to the self-healing function of the underlying storage.

High priority stability enhancements

1. Fixed a bug that might cause the database engine to crash causing unavailability. This occurred if a newly established database connection encountered a resource exhaustion-related error during initialization after successful authentication.

Additional improvements and enhancements

1. Provided a fix for the `pg_hint_plan` extension that could lead the database engine to crash causing unavailability. The open source issue can be tracked at https://github.com/ossc-db/pg_hint_plan/pull/45.
2. Fixed a bug where SQL of the form `ALTER FUNCTION ... OWNER TO ...` incorrectly reported `ERROR: improper qualified name (too many dotted names)`.
3. Improved the performance of GIN index vacuum via prefetching.
4. Fixed a bug in open source PostgreSQL that could lead to a database engine crash causing unavailability. This occurred during parallel B-Tree index scans. This issue has been reported to the PostgreSQL community.
5. Improved the performance of in-memory B-Tree index scans.
6. Additional general improvements to the stability and availability of Aurora PostgreSQL.

Aurora PostgreSQL release 2.4.0

You can find the following new features and improvements in this engine version.

New features

1. Support for exporting data to Amazon S3. For more information, see [Exporting data from an Aurora PostgreSQL DB cluster to Amazon S3 \(p. 1450\)](#).
2. Support for Amazon Aurora Machine Learning. For more information, see [Using machine learning \(ML\) with Aurora PostgreSQL \(p. 1494\)](#).
3. SQL processing enhancements include:
 - Optimizations for `NOT IN` with the `apg_enable_not_in_transform` parameter.
 - Semi-join filter pushdown enhancements for hash joins with the `apg_enable_semijoin_push_down` parameter.
 - Optimizations for redundant inner join removal with the `apg_enable_remove_redundant_inner_joins` parameter.
 - Improved ANSI compatibility options with the `ansi_constraint_trigger_ordering`, `ansi_force_foreign_key_checks` and `ansi_qualified_update_set_target` parameters.

For more information, see [Amazon Aurora PostgreSQL parameters \(p. 1547\)](#).

4. New and updated PostgreSQL extensions include:
 - The new `aws_ml` extension. For more information, see [Using machine learning \(ML\) with Aurora PostgreSQL \(p. 1494\)](#).
 - The new `aws_s3` extension. For more information, see [Exporting data from an Aurora PostgreSQL DB cluster to Amazon S3 \(p. 1450\)](#).
 - Updates to the `apg_plan_mgmt` extension. For more information, see [Managing query execution plans for Aurora PostgreSQL \(p. 1460\)](#)

Critical stability enhancements

1. Fixed a bug related to creating B-tree indexes on temporary tables that in rare cases may result in longer recovery time, and impact availability.
2. Fixed a bug related to replication when Aurora PostgreSQL is acting as a physical replica of an RDS PostgreSQL instance. In rare cases, this bug causes a log write failure that may result in longer recovery time, and impact availability.
3. Fixed a bug related to handling of reads with high I/O latency that in rare cases may result in longer recovery time, and impact availability.

High priority stability enhancements

1. Fixed a bug related to logical replication in which wal segments are not properly removed from storage. This can result in storage bloat. To monitor this, view the `TransactionLogDiskUsage` parameter.
2. Fixed multiple bugs, which cause Aurora to crash during prefetch operations on Btree indexes.
3. Fixed a bug in which an Aurora restart may timeout when logical replication is used.
4. Enhanced the validation checks performed on data blocks in the buffer cache. This improves Aurora's detection of inconsistency.

Additional improvements and enhancements

1. The query plan management extension `apg_plan_mgmt` has an improved algorithm for managing plan generation for highly partitioned tables.
2. Reduced startup time on instances with large caches via improvements in the buffer cache recovery algorithm.
3. Improved the performance of the read-node-apply process under high transaction rate workloads by using changes to PostgreSQL `IWLCK` prioritization. These changes prevent starvation of the read-node-apply process while the PostgreSQL `ProcArray` is under heavy contention.
4. Improved handling of batch reads during vacuum, table scans, and index scans. This results in greater throughput and lower CPU consumption.
5. Fixed a bug in which a read node may crash during the replay of a PostgreSQL `SRLRU-truncate` operation.
6. Fixed a bug where in rare cases, database writes may stall following an error returned by one of the six copies of an Aurora log record.
7. Fixed a bug related to logical replication where an individual transaction larger than 1 GB in size may result in an engine crash.
8. Fixed a memory leak on read nodes when cluster cache management is enabled.
9. Fixed a bug in which importing an RDS PostgreSQL snapshot might stop responding if the source snapshot contains a large number of unlogged relations.
10. Fixed a bug in which the Aurora storage daemon may crash under heavy I/O load.
11. Fixed a bug related to `hot_standby_feedback` for read nodes in which the read node may report the wrong transaction id epoch to the write node. This can cause the write node to ignore the `hot_standby_feedback` and invalidate snapshots on the read node.
12. Fixed a bug in which storage errors that occur during `CREATE DATABASE` statements are not properly handled. The bug left the resulting database inaccessible. The correct behavior is to fail the database creation and return the appropriate error to the user.
13. Improved handling of PostgreSQL snapshot overflow when a read node attempts to connect to a write node. Prior to this change, if the write node was in a snapshot overflow state, the read node was unable to join. A message appeared in the PostgreSQL log file in the form `DEBUG: recovery`

snapshot waiting for non-overflowed snapshot or until oldest active xid on standby is at least `xxxxxxxx` (now `yyyyyyyy`). A snapshot overflow occurs when an individual transaction has created over 64 subtransactions.

14Fixed a bug related to common table expressions in which an error is incorrectly raised when a NOT IN clause exists in a CTE. The error is CTE with NOT IN fails with ERROR: could not find CTE `CTE-Name`.

15Fixed a bug related to an incorrect `last_error_timestamp` value in the `aurora_replica_status` table.

16Fixed a bug to avoid populating shared buffers with blocks belonging to temporary objects. These blocks correctly reside in PostgreSQL backend local buffers.

17Improved the performance of vacuum cleanup on GIN indexes.

18Fixed a bug where in rare cases Aurora may exhibit 100% CPU utilization while acting as a replica of an RDS PostgreSQL instance even when the replication stream is idle.

19Backported a change from PostgreSQL 11 which improves the cleanup of orphaned temporary tables. Without this change, it is possible that in rare cases orphaned temporary tables can lead to transaction ID wraparound. For more information, see this [PostgreSQL community commit](#).

20Fixed a bug where a Writer instance may accept replication registration requests from Reader instances while having an uninitialized startup process.

21Changed the following extensions:

- Updated `pg_hint_plan` to version 1.3.3.
- Added `plprofiler` version 4.1.

For information about extensions and modules, see [Extensions supported for Aurora PostgreSQL 10.x \(p. 1676\)](#).

PostgreSQL 10.7, Aurora PostgreSQL release 2.3 (unsupported)

Note

The PostgreSQL engine version 10.7 with the Aurora PostgreSQL release 2.3 is no longer supported. To upgrade, see [Upgrading the PostgreSQL DB engine for Aurora PostgreSQL \(p. 1681\)](#).

This release of Aurora PostgreSQL is compatible with PostgreSQL 10.7. For more information about the improvements in PostgreSQL 10.7, see [PostgreSQL release 10.7](#).

Patch releases

- [Aurora PostgreSQL release 2.3.5 \(p. 1644\)](#)
- [Aurora PostgreSQL release 2.3.3 \(p. 1645\)](#)
- [Aurora PostgreSQL release 2.3.1 \(p. 1645\)](#)
- [Aurora PostgreSQL release 2.3.0 \(p. 1645\)](#)

Aurora PostgreSQL release 2.3.5

You can find the following improvements in this release.

Improvements

1. Fixed a bug that could cause DB instance restarts.
2. Fixed a bug that could cause a crash when the PostgreSQL backend exits while using logical replication.
3. Fixed a bug that could cause a restart when reads occurred during failovers.

4. Fixed a bug with the `wal2json` module for logical replication.
5. Fixed a bug that could result in inconsistent metadata.

Aurora PostgreSQL release 2.3.3

You can find the following improvements in this release.

Improvements

1. Provided a backport fix for the PostgreSQL community security issue CVE-2019-10130.
2. Provided a backport fix for the PostgreSQL community security issue CVE-2019-10164.
3. Fixed a bug in which data activity streaming could consume excessive CPU time.
4. Fixed a bug in which parallel threads scanning a B-tree index might stop responding following a disk read.
5. Fixed a bug where use of the `not in` predicate against a common table expression (CTE) could return the following error: "ERROR: bad levelsup for CTE".
6. Fixed a bug in which the read node replay process might stop responding while applying a modification to a generalized search tree (GiST) index.
7. Fixed a bug in which visibility map pages could contain incorrect freeze bits following a failover to a read node.
8. Optimized log traffic between the write node and read nodes during index maintenance.
9. Fixed a bug in which queries on read nodes may crash while performing a B-tree index scan.
10. Fixed a bug in which a query that has been optimized for redundant inner join removal could crash.
11. The function `aurora_stat_memctx_usage` now reports the number of instances of a given context name.
12. Fixed a bug in which the function `aurora_stat_memctx_usage` reported incorrect results.
13. Fixed a bug in which the read node replay process could wait to stop conflicting queries beyond the configured `max_standby_streaming_delay` value.
14. Additional information is now logged on read nodes when active connections conflict with the relay process.
15. Provided a backport fix for the PostgreSQL community bug #15677, where a crash could occur while deleting from a partitioned table.

Aurora PostgreSQL release 2.3.1

You can find the following improvements in this release.

Improvements

1. Fixed multiple bugs related to I/O prefetching that caused engine crashes.

Aurora PostgreSQL release 2.3.0

You can find the following improvements in this release.

New features

1. Aurora PostgreSQL now performs I/O prefetching while scanning B-tree indexes. This results in significantly improved performance for B-tree scans over uncached data.

Improvements

1. Fixed a bug in which read nodes may fail with the error "too many LWLocks taken".
2. Addressed numerous issues that caused read nodes to fail to startup while the cluster is under heavy write workload.
3. Fixed a bug in which usage of the `aurora_stat_memctx_usage()` function could lead to a crash.
4. Improved the cache replacement strategy used by table scans to minimize thrashing of the buffer cache.

PostgreSQL 10.6, Aurora PostgreSQL release 2.2 (unsupported)

Note

The PostgreSQL engine version 10.6 with the Aurora PostgreSQL release 2.2 is no longer supported. To upgrade, see [Upgrading the PostgreSQL DB engine for Aurora PostgreSQL \(p. 1681\)](#).

This release of Aurora PostgreSQL is compatible with PostgreSQL 10.6. For more information about the improvements in PostgreSQL 10.6, see [PostgreSQL release 10.6](#).

Patch releases

- [Aurora PostgreSQL release 2.2.1 \(p. 1646\)](#)
- [Aurora PostgreSQL release 2.2.0 \(p. 1646\)](#)

Aurora PostgreSQL release 2.2.1

You can find the following improvements in this release.

Improvements

1. Improved stability of logical replication.
2. Fixed a bug which could cause an error running queries. The message reported was of the form "CLOG segment 123 does not exist: No such file or directory".
3. Increased the supported size of IAM passwords to 8KB.
4. Improved consistency of performance under high throughput write workloads.
5. Fixed a bug which could cause a read replica to crash during a restart.
6. Fixed a bug which could cause an error running queries. The message reported was of the form "SQL ERROR: Attempting to read past EOF of relation".
7. Fixed a bug which could cause an increase in memory usage after a restart.
8. Fixed a bug which could cause a transaction with a large number of subtransactions to fail.
9. Merged a patch from community PostgreSQL which addresses potential failures when using GIN indexes. For more information see <https://git.postgresql.org/gitweb/?p=postgresql.git;a=commit;h=f9e66f2fbbb49a493045c8d8086a9b15d95b8f18>.
10. Fixed a bug which could cause a snapshot import from RDS for PostgreSQL to fail.

Aurora PostgreSQL release 2.2.0

You can find the following improvements in this release.

New features

1. Added the restricted password management feature. Restricted password management enables you to restrict who can manage user passwords and password expiration changes by using the parameter

`rds.restrict_password_commands` and the role `rds_password`. For more information, see [Restricting password management \(p. 1277\)](#).

PostgreSQL 10.5, Aurora PostgreSQL release 2.1 (unsupported)

Note

The PostgreSQL engine version 10.5 with the Aurora PostgreSQL release 2.1 is no longer supported. To upgrade, see [Upgrading the PostgreSQL DB engine for Aurora PostgreSQL \(p. 1681\)](#).

This release of Aurora PostgreSQL is compatible with PostgreSQL 10.5. For more information about the improvements in PostgreSQL 10.5, see [PostgreSQL release 10.5](#).

Patch releases

- [Aurora PostgreSQL release 2.1.1 \(p. 1647\)](#)
- [Aurora PostgreSQL release 2.1.0 \(p. 1647\)](#)

Aurora PostgreSQL release 2.1.1

You can find the following improvements in this release.

Improvements

1. Fixed a bug which could cause an error running queries. The message reported was of the form "CLOG segment 123 does not exist: No such file or directory".
2. Increased the supported size of IAM passwords to 8KB.
3. Improved consistency of performance under high throughput write workloads.
4. Fixed a bug which could cause a read replica to crash during a restart.
5. Fixed a bug which could cause an error running queries. The message reported was of the form "SQL ERROR: Attempting to read past EOF of relation".
6. Fixed a bug which could cause an increase in memory usage after a restart.
7. Fixed a bug which could cause a transaction with a large number of subtransactions to fail.
8. Merged a patch from community PostgreSQL which addresses potential failures when using GIN indexes. For more information see <https://git.postgresql.org/gitweb/?p=postgresql.git;a=commit;h=f9e66f2fbbb49a493045c8d8086a9b15d95b8f18>.
9. Fixed a bug which could cause a snapshot import from RDS for PostgreSQL to fail.

For information about extensions and modules, see [Extensions supported for Aurora PostgreSQL 10.x \(p. 1676\)](#).

Aurora PostgreSQL release 2.1.0

You can find the following improvements in this release.

New features

1. General availability of Aurora Query Plan Management, which enables customers to track and manage any or all query plans used by their applications, to control query optimizer plan selection, and to ensure high and stable application performance. For more information, see [Managing query execution plans for Aurora PostgreSQL \(p. 1460\)](#).
2. Updated the `libprotobuf` extension to version 1.3.0. This is used by the PostGIS extension.
3. Updated the `pg_similarity` extension to version 1.0.
4. Updated the `log_fdw` extension to version 1.1.

5. Updated the `pg_hint_plan` extension to version 1.3.1.

Improvements

1. Network traffic between the writer and reader nodes is now compressed to reduce network utilization. This reduces the chance of read node unavailability due to network saturation.
2. Implemented a high performance, scalable subsystem for PostgreSQL subtransactions. This improves the performance of applications which make extensive use of savepoints and PL/pgSQL exception handlers.
3. The `rds_superuser` role can now set the following parameters on a per-session, database, or role level:
 - `log_duration`
 - `log_error_verbosity`
 - `log_executor_stats`
 - `log_lock_waits`
 - `log_min_duration_statement`
 - `log_min_error_statement`
 - `log_min_messages`
 - `log_parser_stats`
 - `log_planner_stats`
 - `log_replication_commands`
 - `log_statement_stats`
 - `log_temp_files`
4. Fixed a bug in which the SQL command "ALTER FUNCTION ... OWNER TO ..." might fail with error "improper qualified name (too many dotted names)".
5. Fixed a bug in which a crash could occur while committing a transaction with more than two million subtransactions.
6. Fixed a bug in community PostgreSQL code related to GIN indexes which can cause the Aurora Storage volume to become unavailable.
7. Fixed a bug in which an Aurora PostgreSQL replica of an RDS for PostgreSQL instance might fail to start, reporting error: "PANIC: could not locate a valid checkpoint record".
8. Fixed a bug in which passing an invalid parameter to the `aurora_stat_backend_waits` function could cause a crash.

Known issues

1. The `pageinspect` extension is not supported in Aurora PostgreSQL.

PostgreSQL 10.4, Aurora PostgreSQL release 2.0 (unsupported)

Note

The PostgreSQL engine version 10.4 with the Aurora PostgreSQL release 2.0 is no longer supported. To upgrade, see [Upgrading the PostgreSQL DB engine for Aurora PostgreSQL \(p. 1681\)](#).

This release of Aurora PostgreSQL is compatible with PostgreSQL 10.4. For more information about the improvements in PostgreSQL 10.4, see [PostgreSQL release 10.4](#).

Patch releases

- [Aurora PostgreSQL release 2.0.1 \(p. 1649\)](#)

- [Aurora PostgreSQL release 2.0.0 \(p. 1649\)](#)

Aurora PostgreSQL release 2.0.1

You can find the following improvements in this release.

Improvements

1. Fixed a bug which could cause an error running queries. The message reported was of the form "CLOG segment 123 does not exist: No such file or directory".
2. Increased the supported size of IAM passwords to 8KB.
3. Improved consistency of performance under high throughput write workloads.
4. Fixed a bug which could cause a read replica to crash during a restart.
5. Fixed a bug which could cause an error running queries. The message reported was of the form "SQL ERROR: Attempting to read past EOF of relation".
6. Fixed a bug which could cause an increase in memory usage after a restart.
7. Fixed a bug which could cause a transaction with a large number of subtransactions to fail.
8. Merged a patch from community PostgreSQL which addresses potential failures when using GIN indexes. For more information see <https://git.postgresql.org/gitweb/?p=postgresql.git;a=commit;h=f9e66f2fbbb49a493045c8d8086a9b15d95b8f18>.
9. Fixed a bug which could cause a snapshot import from RDS for PostgreSQL to fail.

For information about extensions and modules, see [Extensions supported for Aurora PostgreSQL 10.x \(p. 1676\)](#).

Aurora PostgreSQL release 2.0.0

You can find the following improvements in this release.

Improvements

1. This release contains all fixes, features, and improvements present in [PostgreSQL 9.6.9, Aurora PostgreSQL release 1.3 \(unsupported\) \(p. 1662\)](#).
2. The temporary file size limitation is user-configurable. You require the `rds_superuser` role to modify the `temp_file_limit` parameter.
3. Updated the GDAL library, which is used by the PostGIS extension.
4. Updated the ip4r extension to version 2.1.1.
5. Updated the pg_repack extension to version 1.4.3.
6. Updated the plv8 extension to version 2.1.2.
7. Parallel queries – When you create a new Aurora PostgreSQL version 2.0 instance, parallel queries are enabled for the `default.postgres10` parameter group. The parameter `max_parallel_workers_per_gather` is set to 2 by default, but you can modify it to support your specific workload requirements.
8. Fixed a bug in which read nodes may crash following a specific type of free space change from the write node.

PostgreSQL 9.6.22, Aurora PostgreSQL release 1.11 (unsupported)

Note

The PostgreSQL engine version 9.6.22 and Aurora PostgreSQL release 1.10 are no longer supported. To upgrade, see [Upgrading the PostgreSQL DB engine for Aurora PostgreSQL \(p. 1681\)](#).

This release of Aurora PostgreSQL is compatible with PostgreSQL 9.6.22. For more information about the improvements in PostgreSQL 9.6.22, see [PostgreSQL release 9.6.22](#).

Patch releases

- [Aurora PostgreSQL 1.11.1 \(p. 1650\)](#)
- [Aurora PostgreSQL release 1.11 \(p. 1650\)](#)

Aurora PostgreSQL 1.11.1

High priority stability enhancements

- Fixed an issue where queries may become unresponsive due to I/O resource exhaustion triggered by prefetch.

Additional improvements and stability enhancements

- Fixed multiple issues in the Aurora storage daemon that could lead to brief periods of unavailability when specific network configurations are used.

Aurora PostgreSQL release 1.11

High priority stability enhancements

1. Fixed an issue where creating a database from an existing template database with tablespace resulted in an error with the message `ERROR: could not open file pg_tblspc/...: No such file or directory`.
2. Fixed an issue where, in rare cases, an Aurora replica may be unable to start when a large number of PostgreSQL subtransactions (i.e. SQL savepoints) have been used.
3. Fixed an issue where, in rare circumstances, read results may be inconsistent for repeated read requests on replica nodes.

Additional improvements and enhancements

1. Upgraded OpenSSL to 1.1.1k.
2. Reduced CPU usage and memory consumption of the WAL apply process on Aurora replicas for some workloads.
3. Improve safety checks in the write path to detect incorrect writes to metadata.
4. Fixed an issue where a duplicate file entry can prevent the Aurora PostgreSQL engine from starting up.
5. Fixed an issue that could cause temporary unavailability under heavy workloads.
6. Added back ability to use a leading forward slash in the S3 path during S3 import.
7. Updated the PostGIS extension to version 2.4.7.

8. Updated the `Orafce` extension to version 3.16.

PostgreSQL 9.6.21, Aurora PostgreSQL release 1.10 (unsupported)

Note

The PostgreSQL engine version 9.6.21 and Aurora PostgreSQL release 1.10 are no longer supported. To upgrade, see [Upgrading the PostgreSQL DB engine for Aurora PostgreSQL \(p. 1681\)](#).

This release of Aurora PostgreSQL is compatible with PostgreSQL 9.6.21. For more information about the improvements in PostgreSQL 9.6.21, see [PostgreSQL release 9.6.21](#).

Aurora PostgreSQL release 1.10.0

High priority stability enhancements

1. Fixed a bug where in rare cases a reader had inconsistent results when it restarted while a transaction with more than 64 subtransactions was being processed.
2. Backported fixes for the following PostgreSQL community security issues:
 - [CVE-2021-32027](#)
 - [CVE-2021-32028](#)
 - [CVE-2021-32029](#)

Additional improvements and enhancements

1. Fixed a bug where the database could not be started when there were many relations in memory-constrained environments.
2. Fixed a bug in the `apg_plan_mgmt` extension that could cause brief periods of unavailability due to an internal buffer overflow.
3. Fixed a bug where the database engine would attempt to create shared memory segments larger than the instance total memory and fail repeatedly. For example, attempts to create 128 GiB shared buffers on a db.r5.large instance would fail. With this change, requests for total shared memory allocations larger than the instance memory allow setting the instance to incompatible parameters.
4. Added logic to clean up unnecessary `pg_wal` temporary files on a database startup.
5. Fixed a bug in Aurora PostgreSQL 9.6 that sometimes prevented read/write nodes from starting up when inbound replication is used.
6. Fixed a bug that could cause brief periods of unavailability due to running out of memory when creating the `postgis` extension with `pgAudit` enabled.
7. Added btree page checks to detect tuple metadata inconsistency.

PostgreSQL 9.6.19, Aurora PostgreSQL release 1.9 (unsupported)

Note

The PostgreSQL engine version 9.6.19 and Aurora PostgreSQL release 1.9 are no longer supported. To upgrade, see [Upgrading the PostgreSQL DB engine for Aurora PostgreSQL \(p. 1681\)](#).

This release of Aurora PostgreSQL is compatible with PostgreSQL 9.6.19. For more information about the improvements in PostgreSQL 9.6.19, see [PostgreSQL release 9.6.19](#).

Patch releases

- [Aurora PostgreSQL release 1.9.2 \(p. 1652\)](#)
- [Aurora PostgreSQL release 1.9.1 \(p. 1652\)](#)
- [Aurora PostgreSQL release 1.9.0 \(p. 1652\)](#)

Aurora PostgreSQL release 1.9.2

High priority stability enhancements

1. Fixed a bug where a reader node might render an extra or missing row if the reader restarted while the writer node is processing a long transaction with more than 64 subtransactions.

Additional improvements and enhancements

1. Fixed a bug where a large S3 import with thousands of clients can cause one or more of the import clients to stop responding.

Aurora PostgreSQL release 1.9.1

Critical stability enhancements

1. Fixed a bug that caused a read replica to unsuccessfully restart repeatedly in rare cases.

Additional improvements and enhancements

1. Fixed a bug that when under heavy load, snapshot import, COPY import, or S3 import stopped responding in rare cases.
2. Fixed a bug where a read replica might not join the cluster when the writer was very busy with a write-intensive workload.

Aurora PostgreSQL release 1.9.0

Critical stability enhancements

- None

High priority stability enhancements

1. Backported a fix for the PostgreSQL community security issues CVE-2020-25694, CVE-2020-25695, and CVE-2020-25696.
2. Fixed a bug in Aurora PostgreSQL replication that might result in the following error message: ERROR: MultiXactId nnnn has not been created yet -- apparent wraparound

Additional improvements and enhancements

1. Aurora PostgreSQL no longer falls behind on a read node when the backend is blocked writing to the database client.
2. Fixed a bug that in rare cases caused a brief period of unavailability on a read replica when the storage volume grew.

3. Fixed a bug when creating a database that could return the following error: ERROR: could not create directory on local disk
4. Fixed a bug in the GiST index that could result in an out of memory condition after promoting an Aurora read replica.
5. Fixed a bug related to replication when Aurora PostgreSQL is acting as a physical replica of an RDS PostgreSQL instance that uses GiST indexes. In rare cases, this bug caused a brief period of unavailability after promoting the Aurora cluster.

PostgreSQL 9.6.18, Aurora PostgreSQL release 1.8 (unsupported)

Note

The PostgreSQL engine version 9.6.18 and Aurora PostgreSQL release 1.8 are no longer supported. To upgrade, see [Upgrading the PostgreSQL DB engine for Aurora PostgreSQL \(p. 1681\)](#).

This release of Aurora PostgreSQL is compatible with PostgreSQL 9.6.18. For more information about the improvements in PostgreSQL 9.6.18, see [PostgreSQL release 9.6.18](#).

Patch releases

- [Aurora PostgreSQL release 1.8.2 \(p. 1653\)](#)
- [Aurora PostgreSQL release 1.8.0 \(p. 1654\)](#)

There is no version 1.8.1.

Aurora PostgreSQL release 1.8.2

Critical stability enhancements

1. None

High priority stability enhancements

1. Fixed a bug in Aurora PostgreSQL replication that could result in the error message ERROR: MultiXactId nnnn has not been created yet -- apparent wraparound.
2. Backported fixes for the following PostgreSQL community security issues:
 - [CVE-2020-25694](#)
 - [CVE-2020-25695](#)
 - [CVE-2020-25696](#)

Additional improvements and enhancements

1. Aurora PostgreSQL no longer falls behind on a read node when the backend is blocked writing to the database client.
2. Fixed a bug where a `DROP DATABASE` statement didn't remove any relation files.
3. Fixed a small memory leak in a b-tree index that could lead to an out of memory condition.
4. Fixed a bug in the `aurora_replica_status()` function where the `server_id` field was sometimes truncated.
5. Fixed a bug related to replication when Aurora PostgreSQL is acting as a physical replica of an RDS PostgreSQL instance that uses GiST indexes. In rare cases, this bug caused a brief period of unavailability after promoting the Aurora DB cluster.

Aurora PostgreSQL release 1.8.0

You can find the following improvements in this release.

Critical stability enhancements

1. Fixed a bug related to heap page extend that in rare cases resulted in longer recovery time and impacted availability.

Additional improvements and enhancements

1. Fixed a bug where the Aurora replica crashed when workloads with heavy subtransactions are made on the writer instance.
2. Fixed a bug where the writer instance crashed due to a memory leak and the depletion of memory used to track active transactions.
3. Fixed a bug that lead to a crash due to improper initialization when there is no free memory available during PostgreSQL backend startup.
4. Fixed a crash during a BTREE prefetch that occurred under certain conditions that depended on the shape and data contained in the index.
5. Fixed a bug where a `SELECT` query might incorrectly return the error Attempting to read past EOF of relation rrrr. blockno=bbb nblocks=nnn.
6. Fixed a bug where the database might be unavailable briefly due to error handling in database storage growth.

PostgreSQL 9.6.17, Aurora PostgreSQL release 1.7 (unsupported)

This release of Aurora PostgreSQL is compatible with PostgreSQL 9.6.17. For more information about the improvements in PostgreSQL 9.6.17, see [PostgreSQL release 9.6.17](#).

Patch releases

- [Aurora PostgreSQL release 1.7.7 \(p. 1654\)](#)
- [Aurora PostgreSQL release 1.7.6 \(p. 1655\)](#)
- [Aurora PostgreSQL release 1.7.3 \(p. 1655\)](#)
- [Aurora PostgreSQL release 1.7.2 \(p. 1655\)](#)
- [Aurora PostgreSQL release 1.7.1 \(p. 1656\)](#)

Aurora PostgreSQL release 1.7.7

You can find the following improvements in this release.

Critical stability enhancements

- None

High priority stability enhancements

1. Backported fixes for the following PostgreSQL community security issues:
 - [CVE-2020-25694](#)
 - [CVE-2020-25695](#)
 - [CVE-2020-25696](#)

Additional improvements and enhancements

- None

[Aurora PostgreSQL release 1.7.6](#)

You can find the following improvements in this release.

Critical stability enhancements

- None

High priority stability enhancements

1. Fixed a bug in Aurora PostgreSQL replication that might result in the error message, ERROR: MultiXactId nnnn has not been created yet -- apparent wraparound.

Additional improvements and enhancements

1. Fixed a bug that in rare cases caused brief read replica unavailability when storage volume grew.
2. Fixed a bug in a b-tree index read optimization that might have caused a brief period of unavailability.
3. Fixed a bug in a GiST index that might result in an out of memory condition after promoting an Aurora Read Replica.

[Aurora PostgreSQL release 1.7.3](#)

You can find the following improvements in this release.

Critical stability enhancements

- None

High priority stability enhancements

- None

Additional improvements and enhancements

1. Fixed a bug that might cause brief unavailability for heavy subtransaction workloads when multiple reader instances restart or rejoin the cluster.

[Aurora PostgreSQL release 1.7.2](#)

You can find the following improvements in this release.

Critical stability enhancements

1. Fixed a bug related to heap page extend that in rare cases resulted in longer recovery time and impacted availability.

High Priority Stability Enhancements

None

Additional improvements and enhancements

1. Fixed a bug where the database might be unavailable briefly due to error handling in database storage growth.
2. Fixed a bug where a SELECT query might incorrectly return the error, Attempting to read past EOF of relation rrrr. blockno=bbb nblocks=nnn.
3. Fixed an issue with the internal metrics collector that could result in erratic CPU spikes on database instances.

Aurora PostgreSQL release 1.7.1

You can find the following improvements in this release.

Critical stability enhancements

None.

High priority stability enhancements

1. Improved performance and availability of read instances when applying DROP TABLE and TRUNCATE TABLE operations.
2. Fixed a small but continuous memory leak in a diagnostic module that could lead to an out-of-memory condition on smaller DB instance types.
3. Fixed a bug in the PostGIS extension which could lead to a database restart. This has been reported to the PostGIS community as <https://trac.osgeo.org/postgis/ticket/4646>.
4. Fixed a bug where read requests might stop responding due to incorrect error handling in the storage engine.
5. Fixed a bug that fails for some queries and results in the message ERROR: found xmin xxxxxx from before refrozenid yyyyyyy. This could occur following the promotion of a read instance to a write instance.

Additional improvements and enhancements

1. Improved performance for queries that read many rows from storage.
2. Improved performance and availability of reader DB instances during heavy read workload.
3. Fixed a bug that resulted in the message ERROR: could not create file "base/xxxxxx/yyyyyyy" as a previous version still exists on disk: Success. Please contact AWS customer support. This can occur during object creation after PostgreSQL's 32-bit object identifier has wrapped around.
4. Fixed a bug in the pg_hint_plan extension where a multi-statement query could lead to a crash when enable_hint_table is enabled. This is tracked in the PostgreSQL community as https://github.com/ossc-db/pg_hint_plan/issues/25.
5. Changed the following extensions:
 - Updated orafce to version 3.8

PostgreSQL 9.6.16, Aurora PostgreSQL release 1.6 (unsupported)

This version of Aurora PostgreSQL is compatible with PostgreSQL 9.6.16. For more information about the improvements in release 9.6.16, see [PostgreSQL release 9.6.16](#).

This release contains multiple critical stability enhancements. Amazon highly recommends upgrading your Aurora PostgreSQL clusters that use older PostgreSQL 9.6 engines to this release.

Patch versions

- [Aurora PostgreSQL release 1.6.4 \(p. 1657\)](#)
- [Aurora PostgreSQL release 1.6.3 \(p. 1657\)](#)
- [Aurora PostgreSQL release 1.6.2 \(p. 1657\)](#)
- [Aurora PostgreSQL release 1.6.1 \(p. 1658\)](#)
- [Aurora PostgreSQL release 1.6.0 \(p. 1658\)](#)

Aurora PostgreSQL release 1.6.4

You can find the following improvements in this release.

Critical stability enhancements

- None

High priority stability enhancements

1. Backported fixes for the following PostgreSQL community security issues:
 - [CVE-2020-25694](#)
 - [CVE-2020-25695](#)
 - [CVE-2020-25696](#)

Additional improvements and enhancements

- None

Aurora PostgreSQL release 1.6.3

New features

1. Aurora PostgreSQL now supports the PostgreSQL `vacuum_truncate` storage parameter to manage vacuum truncation for specific tables. Set this [storage parameter](#) to false when creating or altering a table to prevent the `VACUUM` SQL command from truncating the table's trailing empty pages.

Critical stability enhancements

- None

High priority stability enhancements

1. Fixed a bug where reads from storage might stop responding due to incorrect error handling.

Additional improvements and enhancements

- None

Aurora PostgreSQL release 1.6.2

You can find the following improvements in this engine update.

Critical stability enhancements

1. Fixed a bug in which a reader DB instance might temporarily use stale data. This could lead to wrong results such as too few or too many rows. This error is not persisted on storage, and will clear when the database page containing the row has been evicted from cache. This can happen when the primary DB instance enters a transaction snapshot overflow due to having more than 64 subtransactions in a single transaction. Applications susceptible to this bug include those that use SQL savepoints or PostgreSQL exception handlers with more than 64 subtransactions in the top transaction.

High priority stability enhancements

1. Fixed a bug that may cause a reader DB instance to crash causing unavailability while attempting to join the DB cluster. This can happen in some cases when the primary DB instance has a transaction snapshot overflow due to a high number of subtransactions. In this situation the reader DB instance will be unable to join until the snapshot overflow has cleared.

Additional improvements and enhancements

1. Fixed a bug that prevented Performance Insights from determining the query ID of a running statement.

Aurora PostgreSQL release 1.6.1

You can find the following improvements in this engine update.

Critical stability enhancements

1. None

High priority stability enhancements

1. Fixed a bug that might cause the database engine to crash causing unavailability. This occurred if a newly established database connection encountered a resource exhaustion-related error during initialization after successful authentication.

Additional improvements and enhancements

1. Provided general improvements to the stability and availability of Aurora PostgreSQL.

Aurora PostgreSQL release 1.6.0

You can find the following new features and improvements in this engine version.

New features

1. Updates to the `apg_plan_mgmt` extension. For more information, see [Managing query execution plans for Aurora PostgreSQL \(p. 1460\)](#)

Critical stability enhancements

1. Fixed a bug related to creating B-tree indexes on temporary tables that in rare cases may result in longer recovery time, and impact availability.

2. Fixed a bug related to replication when Aurora PostgreSQL is acting as a physical replica of an RDS PostgreSQL instance. In rare cases, this bug causes a log write failure that may result in longer recovery time, and impact availability.
3. Fixed a bug related to handling of reads with high I/O latency that in rare cases may result in longer recovery time, and impact availability.

High priority stability enhancements

1. Fixed multiple bugs, which cause Aurora to crash during prefetch operations on Btree indexes.
2. Enhanced the validation checks performed on data blocks in the buffer cache. This improves Aurora's detection of inconsistency.

Additional improvements and enhancements

1. The query plan management extension `apg_plan_mgmt` has an improved algorithm for managing plan generation for highly partitioned tables.
2. Reduced startup time on instances with large caches via improvements in the buffer cache recovery algorithm.
3. Improved the performance of the read-node-apply process under high transaction rate workloads by using changes to PostgreSQL LWLock prioritization. These changes prevent starvation of the read-node-apply process while the PostgreSQL `ProcArray` is under heavy contention.
4. Fixed a bug in which a read node may crash during the replay of a PostgreSQL SLRU-truncate operation.
5. Fixed a bug where in rare cases, database writes might stall following an error returned by one of the six copies of an Aurora log record.
6. Fixed a memory leak on read nodes when cluster cache management is enabled.
7. Fixed a bug in which importing an RDS PostgreSQL snapshot might stop responding if the source snapshot contains a large number of unlogged relations.
8. Fixed a bug related to `hot_standby_feedback` for read nodes in which the read node may report the wrong transaction id epoch to the write node. This can cause the write node to ignore the `hot_standby_feedback` and invalidate snapshots on the read node.
9. Fixed a bug in which storage errors that occur during `CREATE DATABASE` statements are not properly handled. The bug left the resulting database inaccessible. The correct behavior is to fail the database creation and return the appropriate error to the user.
10. Improved handling of PostgreSQL snapshot overflow when a read node attempts to connect to a write node. Prior to this change, if the write node was in a snapshot overflow state, the read node was unable to join. A message appears in the PostgreSQL log file in the form `DEBUG: recovery snapshot waiting for non-overflowed snapshot or until oldest active xid on standby is at least xxxxxx (now yyyyyyy)`. A snapshot overflow occurs when an individual transaction has created over 64 subtransactions.
11. Fixed a bug related to common table expressions in which an error is incorrectly raised when a NOT IN clause exists in a CTE. The error is `CTE with NOT IN fails with ERROR: could not find CTE CTE-Name`.
12. Fixed a bug related to an incorrect `last_error_timestamp` value in the `aurora_replica_status` table.
13. Fixed a bug to avoid populating shared buffers with blocks belonging to temporary objects. These blocks correctly reside in PostgreSQL backend local buffers.
14. Fixed a bug where in rare cases Aurora may exhibit 100% CPU utilization while acting as a replica of an RDS PostgreSQL instance even when the replication stream is idle.
15. Backported a change from PostgreSQL 11 which improves the cleanup of orphaned temporary tables. Without this change, it is possible that in rare cases orphaned temporary tables can lead to transaction ID wraparound. For more information, see this [PostgreSQL community commit](#).

16Fixed a bug where a Writer instance may accept replication registration requests from Reader instances while having an uninitialized startup process.

17Changed the following extensions:

- Updated `pg_hint_plan` to version 1.2.5.

PostgreSQL 9.6.12, Aurora PostgreSQL release 1.5 (unsupported)

Note

The PostgreSQL engine version 9.6.12 with the Aurora PostgreSQL release 1.5 is no longer supported. To upgrade, see [Upgrading the PostgreSQL DB engine for Aurora PostgreSQL \(p. 1681\)](#).

This release of Aurora PostgreSQL is compatible with PostgreSQL 9.6.12. For more information about the improvements in PostgreSQL 9.6.12, see [PostgreSQL release 9.6.12](#).

Patch releases

- [Aurora PostgreSQL release 1.5.3 \(p. 1660\)](#)
- [Aurora PostgreSQL release 1.5.2 \(p. 1660\)](#)
- [Aurora PostgreSQL release 1.5.1 \(p. 1661\)](#)
- [Aurora PostgreSQL release 1.5.0 \(p. 1661\)](#)

Aurora PostgreSQL release 1.5.3

You can find the following improvements in this release.

Improvements

1. Fixed a bug that could cause DB instance restarts.
2. Fixed a bug that could cause a restart when reads occurred during failovers.
3. Fixed a bug that could result in inconsistent metadata.

Aurora PostgreSQL release 1.5.2

You can find the following improvements in this release.

Improvements

1. Provided a backport fix for the PostgreSQL community security issue CVE-2019-10130.
2. Fixed a bug in which the read node replay process might stop responding while applying a modification to a generalized search tree (GiST) index.
3. Fixed a bug in which visibility map pages may contain incorrect freeze bits following a failover to a read node.
4. Fixed a bug in which the error "relation `relation-name` does not exist" is incorrectly reported.
5. Optimized log traffic between the write node and read nodes during index maintenance.
6. Fixed a bug in which queries on read nodes may crash while performing a B-tree index scan.
7. The function `aurora_stat_memctx_usage` now reports the number of instances of a given context name.
8. Fixed a bug in which the function `aurora_stat_memctx_usage` reported incorrect results.
9. Fixed a bug in which the read node replay process may wait to stop conflicting queries beyond the configured `max_standby_streaming_delay`.

10Additional information is now logged on read nodes when active connections conflict with the relay process.

Aurora PostgreSQL release 1.5.1

You can find the following improvements in this release.

Improvements

1. Fixed multiple bugs related to I/O prefetching, which caused engine crashes.

Aurora PostgreSQL release 1.5.0

You can find the following improvements in this release.

New features

1. Aurora PostgreSQL now performs I/O prefetching while scanning B-tree indexes. This results in significantly improved performance for B-tree scans over uncached data.

Improvements

1. Addressed numerous issues that caused read nodes to fail to startup while the cluster is under heavy write workload.
2. Fixed a bug in which usage of the `aurora_stat_memctx_usage()` function could lead to a crash.
3. Improved the cache replacement strategy used by table scans to minimize thrashing of the buffer cache.

PostgreSQL 9.6.11, Aurora PostgreSQL release 1.4 (unsupported)

Note

The PostgreSQL engine version 9.6.11 with the Aurora PostgreSQL release 1.4 is no longer supported. To upgrade, see [Upgrading the PostgreSQL DB engine for Aurora PostgreSQL \(p. 1681\)](#).

This release of Aurora PostgreSQL is compatible with PostgreSQL 9.6.11. For more information about the improvements in PostgreSQL 9.6.11, see [PostgreSQL release 9.6.11](#).

You can find the following improvements in this release.

New features

1. Support is added for the `pg_similarity` extension version 1.0.
2. Aurora PostgreSQL now supports the PostgreSQL `vacuum_truncate` storage parameter to manage vacuum truncation for specific tables. Set this `storage parameter` to false when creating or altering a table to prevent the `VACUUM` SQL command from truncating the table's trailing empty pages.

Improvements

1. This release contains all fixes, features, and improvements present in [PostgreSQL 9.6.9, Aurora PostgreSQL release 1.3 \(unsupported\) \(p. 1662\)](#).

2. Network traffic between the writer and reader nodes is now compressed to reduce network utilization. This reduces the chance of read node unavailability due to network saturation.
3. Performance of subtransactions has improved under high concurrency workloads.
4. An update for the `pg_hint_plan` extension to version 1.2.3.
5. Fixed an issue where on a busy system, a commit with millions of subtransactions (and sometimes with commit timestamps enabled) can cause Aurora to crash.
6. Fixed an issue where an `INSERT` statement with `VALUES` could fail with the message "Attempting to read past EOF of relation".
7. An upgrade of the `apg_plan_mgmt` extension to version 1.0.1. For details, see [Version 1.0.1 of the Aurora PostgreSQL `apg_plan_mgmt` extension \(p. 1680\)](#).

The `apg_plan_mgmt` extension is used with query plan management. For more about how to install, upgrade, and use the `apg_plan_mgmt` extension, see [Managing query execution plans for Aurora PostgreSQL \(p. 1460\)](#).

PostgreSQL 9.6.9, Aurora PostgreSQL release 1.3 (unsupported)

Note

The PostgreSQL engine version 9.6.9 with the Aurora PostgreSQL release 1.3 is no longer supported. To upgrade, see [Upgrading the PostgreSQL DB engine for Aurora PostgreSQL \(p. 1681\)](#).

This release of Aurora PostgreSQL is compatible with PostgreSQL 9.6.9. For more information about the improvements in PostgreSQL 9.6.9, see [PostgreSQL release 9.6.9](#).

Patch releases

- [Aurora PostgreSQL release 1.3.2 \(p. 1662\)](#)
- [Aurora PostgreSQL release 1.3.0 \(p. 1663\)](#)

Aurora PostgreSQL release 1.3.2

You can find the following improvements in this release.

New features

1. Added the `ProcArrayGroupUpdate` wait event.

Improvements

1. Fixed a bug which could cause an error running queries. The message reported was of the form "CLOG segment 123 does not exist: No such file or directory".
2. Increased the supported size of IAM passwords to 8KB.
3. Improved consistency of performance under high throughput write workloads.
4. Fixed a bug which could cause a read replica to crash during a restart.
5. Fixed a bug which could cause an error running queries. The message reported was of the form "SQL ERROR: Attempting to read past EOF of relation".
6. Fixed a bug which could cause an increase in memory usage after a restart.
7. Fixed a bug which could cause a transaction with a large number of subtransactions to fail.
8. Merged a patch from community PostgreSQL which addresses potential failures when using GIN indexes. For more information see <https://git.postgresql.org/gitweb/?p=postgresql.git;a=commit;h=f9e66f2fbbb49a493045c8d8086a9b15d95b8f18>.
9. Fixed a bug which could cause a snapshot import from RDS for PostgreSQL to fail.

Aurora PostgreSQL release 1.3.0

You can find the following improvements in this release.

Improvements

1. This release contains all fixes, features, and improvements present in [PostgreSQL 9.6.8, Aurora PostgreSQL release 1.2 \(unsupported\) \(p. 1664\)](#).
2. Updated the GDAL library, which is used by the PostGIS extension.
3. Updated the following PostgreSQL extensions:
 - ip4r updated to version 2.1.1.
 - pgaudit updated to version 1.1.1.
 - pg_repack updated to version 1.4.3.
 - plv8 updated to version 2.1.2.
4. Fixed an issue in the monitoring system that could incorrectly cause a failover when local disk usage is high.
5. Fixed a bug in which Aurora PostgreSQL can repeatedly crash, reporting:

```
PANIC: new_record_total_len (8201) must be less than BLCKSZ (8192), rmid (6), info (32)
```

6. Fixed a bug in which an Aurora PostgreSQL read node might be unable to rejoin a cluster due to recovery of a large buffer cache. This issue is unlikely to occur on instances other than **r4.16xlarge**.
7. Fixed a bug in which inserting into an empty GIN index leaf page imported from pre-9.4 engine versions can cause the Aurora storage volume to become unavailable.
8. Fixed a bug in which, in rare circumstances, a crash during transaction commit could result in the loss of CommitTs data for the committing transaction. The actual durability of the transaction was not impacted by this bug.
9. Fixed a bug in the PostGIS extension in which PostGIS can crash in the function gserialized_gist_picksplit_2d().
- 10Improved the stability of read-only nodes during heavy write traffic on instances smaller than **r4.8xL**. This specifically addresses a situation where the network bandwidth between the writer and the reader is constrained.
- 11Fixed a bug in which an Aurora PostgreSQL instance acting as a replication target of an RDS for PostgreSQL instance crashed with the following error:

```
FATAL: could not open file "base/16411/680897_vm": No such file or directory
during "xlog redo at 782/3122D540 for Storage/TRUNCATE"
```

- 12Fixed a memory leak on read-only nodes in which the heap size for the "aurora wal replay process" will continue to grow. This is observable via Enhanced Monitoring.

- 13Fixed a bug in which Aurora PostgreSQL can fail to start, with the following message reported in the PostgreSQL log:

```
FATAL: Storage initialization failed.
```

- 14Fixed a performance limitation on heavy write workloads that caused waits on the LWLock:buffer_content and IO:ControlFileSyncUpdate events.

- 15Fixed a bug in which read nodes could crash following a specific type of free space change from the write node.

PostgreSQL 9.6.8, Aurora PostgreSQL release 1.2 (unsupported)

Note

The PostgreSQL engine version 9.6.8 with the Aurora PostgreSQL release 1.2 is no longer supported. To upgrade, see [Upgrading the PostgreSQL DB engine for Aurora PostgreSQL \(p. 1681\)](#).

For more information about PostgreSQL 9.6.8, see [PostgreSQL release 9.6.8](#).

Patch releases

- [Aurora PostgreSQL release 1.2.2 \(p. 1664\)](#)
- [Aurora PostgreSQL release 1.2.0 \(p. 1664\)](#)

Aurora PostgreSQL release 1.2.2

You can find the following improvements in this release.

New features

1. Added the `ProcArrayGroupUpdate` wait event.

Improvements

1. Fixed a bug which could cause an error running queries. The message reported was of the form "CLOG segment 123 does not exist: No such file or directory".
2. Increased the supported size of IAM passwords to 8KB.
3. Improved consistency of performance under high throughput write workloads.
4. Fixed a bug which could cause a read replica to crash during a restart.
5. Fixed a bug which could cause an error running queries. The message reported was of the form "SQL ERROR: Attempting to read past EOF of relation".
6. Fixed a bug which could cause an increase in memory usage after a restart.
7. Fixed a bug which could cause a transaction with a large number of subtransactions to fail.
8. Merged a patch from community PostgreSQL which addresses potential failures when using GIN indexes. For more information see <https://git.postgresql.org/gitweb/?p=postgresql.git;a=commit;h=f9e66f2fbbb49a493045c8d8086a9b15d95b8f18>.
9. Fixed a bug which could cause a snapshot import from RDS for PostgreSQL to fail.

Aurora PostgreSQL release 1.2.0

You can find the following improvements in this release.

New features

1. Introduced the `aurora_stat_memctx_usage()` function. This function reports internal memory context usage for each PostgreSQL backend. You can use this function to help determine why certain backends are consuming large amounts of memory.

Improvements

1. This release contains all fixes, features, and improvements present in [PostgreSQL 9.6.6 Aurora PostgreSQL release 1.1 \(unsupported\) \(p. 1665\)](#).
2. Updates the following PostgreSQL extensions:
 - `pg_hint_plan` updated to version 1.2.2

- plv8 updated to version 2.1.0
3. Improves efficiency of traffic between writer and reader nodes.
 4. Improves connection establishment performance.
 5. Improve the diagnostic data provided in the PostgreSQL error log when an out-of-memory error is encountered.
 6. Multiple fixes to improve the reliability and performance of snapshot import from Amazon RDS for PostgreSQL to Aurora PostgreSQL-Compatible Edition.
 7. Multiple fixes to improve the reliability and performance of Aurora PostgreSQL read nodes.
 8. Fixes a bug in which an otherwise idle instance can generate unnecessary read traffic on an Aurora storage volume.
 9. Fixes a bug in which duplicate sequence values can be encountered during insert. The problem only occurs when migrating a snapshot from RDS for PostgreSQL to Aurora PostgreSQL. The fix prevents the problem from being introduced when performing the migration. Instances migrated before this release can still encounter duplicate key errors.
 - 10 Fixes a bug in which an RDS for PostgreSQL instance migrated to Aurora PostgreSQL using replication can run out of memory doing insert/update of GIST indexes, or cause other issues with GIST indexes.
 - 11 Fixes a bug in which vacuum can fail to update the corresponding pg_database.datfrozenxid value for a database.
 - 12 Fixes a bug in which a crash while creating a new MultiXact (contended row level lock) can cause Aurora PostgreSQL to stop responding indefinitely on the first access to the same relation after the engine restarts.
 - 13 Fixes a bug in which a PostgreSQL backend can't be terminated or canceled while invoking an fdw call.
 - 14 Fixes a bug in which one vCPU is fully utilized at all times by the Aurora storage daemon. This issue is especially noticeable on smaller instance classes, such as r4.large, where it can lead to 25–50 percent CPU usage when idle.
 - 15 Fixes a bug in which an Aurora PostgreSQL writer node can fail over spuriously.
 - 16 Fixes a bug in which, in a rare scenario, an Aurora PostgreSQL read node can report:

"FATAL: lock buffer_io is not held"
 - 17 Fixes a bug in which stale relcache entries can halt vacuuming of relations and push the system close to transaction ID wraparound. The fix is a port of a PostgreSQL community patch scheduled to be released in a future minor version.
 - 18 Fixes a bug in which a failure while extending a relation can cause Aurora to crash while scanning the partially extended relation.

PostgreSQL 9.6.6 Aurora PostgreSQL release 1.1 (unsupported)

Note

The PostgreSQL engine version 9.6.6 with the Aurora PostgreSQL release 1.1 is no longer supported. To upgrade, see [Upgrading the PostgreSQL DB engine for Aurora PostgreSQL \(p. 1681\)](#).

For more information about PostgreSQL 9.6.6 see, [PostgreSQL release 9.6.6](#).

You can find the following improvements in this engine update:

New features

1. Introduced the aurora_stat_utils extension. This extension includes two functions:
 - aurora_wait_report() function for wait event monitoring
 - aurora_log_report() for log record write monitoring
2. Added support for the following extensions:

- orafce 3.6.1
- pgrouting 2.4.2
- postgresql-hll 2.10.2
- prefix 1.2.6

Improvements

1. This release contains all fixes, features, and improvements present in [Aurora PostgreSQL release 1.0.11 \(p. 1667\)](#)
2. Updates for the following PostgreSQL extensions:
 - postgis extension updated to version 2.3.4
 - geos library updated to version 3.6.2
 - pg_repack updated to version 1.4.2
3. Access to the pg_statistic relation enabled.
4. Disabled the 'effective_ioConcurrency' guc parameter, as it does not apply to Aurora storage.
5. Changed the 'hot_standby_feedback' guc parameter to not-modifiable and set the value to '1'.
6. Improved heap page read performance during a vacuum operation.
7. Improved performance of snapshot conflict resolution on read nodes.
8. Improved performance of transaction snapshot acquisition on read nodes.
9. Improved write performance for GIN meta page updates.
10. Improved buffer cache recovery performance during startup.
11. Fixes a bug that caused a database engine crash at startup while recovering prepared transactions.
12. Fixes a bug that could result in the inability to start a read node when there are a large number of prepared transactions.
13. Fixes a bug that could cause a read node to report:

```
ERROR: could not access status of transaction 6080077  
DETAIL: * *Could not open file "pg_subtrans/005C": No such file or directory.
```
14. Fixes a bug that could cause the error below when replicating from RDS PostgreSQL to Aurora PostgreSQL:

```
FATAL: lock buffer_content is not held  
CONTEXT: xlog redo at 46E/F1330870 for Storage/TRUNCATE: base/13322/8058750 to 0 blocks  
flags 7
```
15. Fixes a bug that could cause Aurora PostgreSQL to stop responding while replaying a multixact WAL record when replicating from RDS PostgreSQL to Aurora PostgreSQL.
16. Multiple improvements to the reliability of importing snapshots from RDS PostgreSQL to Aurora PostgreSQL.

PostgreSQL 9.6.3, Aurora PostgreSQL release 1.0 (unsupported)

Note

The PostgreSQL engine version 9.6.3 with the Aurora PostgreSQL release 1.0 is no longer supported. To upgrade, see [Upgrading the PostgreSQL DB engine for Aurora PostgreSQL \(p. 1681\)](#).

For more information about PostgreSQL 9.6.3 see, [PostgreSQL release 9.6.3](#).

This version includes the following patch releases:

Patch releases

- [Aurora PostgreSQL release 1.0.11 \(p. 1667\)](#)
- [Aurora PostgreSQL release 1.0.10 \(p. 1667\)](#)
- [Aurora PostgreSQL release 1.0.9 \(p. 1667\)](#)
- [Aurora PostgreSQL release 1.0.8 \(p. 1667\)](#)
- [Aurora PostgreSQL release 1.0.7 \(p. 1668\)](#)

Aurora PostgreSQL release 1.0.11

You can find the following improvements in this engine update:

1. Fixes an issue with parallel query processing that can lead to incorrect results.
2. Fixes an issue with visibility map handling during replication from Amazon RDS for PostgreSQL that can cause the Aurora storage volume to become unavailable.
3. Corrects the pg-repack extension.
4. Implements improvements to maintain fresh nodes.
5. Fixes issues that can lead to an engine crash.

Aurora PostgreSQL release 1.0.10

This update includes a new feature. You can now replicate an Amazon RDS PostgreSQL DB instance to Aurora PostgreSQL. For more information, see [Replication with Amazon Aurora PostgreSQL \(p. 1431\)](#).

You can find the following improvements in this engine update:

1. Adds error logging when a cache exists and a parameter change results in a mismatched buffer cache, storage format, or size.
2. Fixes an issue that causes an engine reboot if there is an incompatible parameter value for huge pages.
3. Improves handling of multiple truncate table statements during a replay of a write ahead log (WAL) on a read node.
4. Reduces static memory overhead to reduce out-of-memory errors.
5. Fixes an issue that can lead to out-of-memory errors while performing an insert with a GiST index.
6. Improves snapshot import from RDS PostgreSQL, removing the requirement that a vacuum be performed on uninitialized pages.
7. Fixes an issue that causes prepared transactions to return to the previous state following an engine crash.
8. Implements improvements to prevent read nodes from becoming stale.
9. Implements improvements to reduce downtime with an engine restart.
10. Fixes issues that can cause an engine crash.

Aurora PostgreSQL release 1.0.9

In this engine update, we fix an issue that can cause the Aurora storage volume to become unavailable when importing a snapshot from RDS PostgreSQL that contained uninitialized pages.

Aurora PostgreSQL release 1.0.8

You can find the following improvements in this engine update:

1. Fixes an issue that prevented the engine from starting if the `shared_preload_libraries` instance parameter contained `pg_hint_plan`.

2. Fixes the error "Attempt to fetch heap block XXX is beyond end of heap (YYY blocks)," which can occur during parallel scans.
3. Improves the effectiveness of prefetching on reads for a vacuum.
4. Fixes issues with snapshot import from RDS PostgreSQL, which can fail if there are incompatible pg_internal.init files in the source snapshot.
5. Fixes an issue that can cause a read node to crash with the message "aurora wal replay process (PID XXX) was terminated by signal 11: Segmentation fault". This issue occurs when the reader applied a visibility map change for an uncached visibility map page.

Aurora PostgreSQL release 1.0.7

This is the first generally available release of Amazon Aurora PostgreSQL-Compatible Edition.

Extension versions for Amazon Aurora PostgreSQL

Topics

- [Extensions supported for Aurora PostgreSQL 13.x \(p. 1668\)](#)
- [Extensions supported for Aurora PostgreSQL 12.x \(p. 1671\)](#)
- [Extensions supported for Aurora PostgreSQL 11.x \(p. 1674\)](#)
- [Extensions supported for Aurora PostgreSQL 10.x \(p. 1676\)](#)
- [Aurora PostgreSQL apg_plan_mgmt extension versions \(p. 1679\)](#)

To upgrade a PostgreSQL extension, see [Upgrading PostgreSQL extensions \(p. 1689\)](#).

Extensions supported for Aurora PostgreSQL 13.x

The following table shows the PostgreSQL extension versions that are currently supported on Aurora PostgreSQL versions 13.x. "NA" indicates that the extension isn't available for that PostgreSQL version. For more information about PostgreSQL extensions, see [Packaging Related Objects into an Extension](#) in the PostgreSQL documentation.

Extension	13.3	13.4	13.5
address_standardizer	3.0.3	3.1.4	3.1.4
address_standardizer_data	3.0.3	3.1.4	3.1.4
amcheck	1.2	1.2	1.2
apg_plan_mgmt (p. 1679)	2.1	2.1	2.1
aurora_stat_utils	1.0	1.0	1.0
autoinc (contrib-spi)	N/A	1.0	1.0
aws_commons	1.2	1.2	1.2
aws_lambda	1.0	1.0	1.0
aws_ml	1.0	1.0	1.0
aws_s3	1.1	1.1	1.1
bloom	1.0	1.0	1.0

Extension	13.3	13.4	13.5
bool_plperl	1.3	1.3	1.3
btree_gin	1.3	1.3	1.3
btree_gist	1.5	1.5	1.5
citext	1.6	1.6	1.6
cube	1.4	1.4	1.4
dblink	1.2	1.2	1.2
dict_int	1.0	1.0	1.0
dict_xsyn	1.0	1.0	1.0
earthdistance	1.1	1.1	1.1
fuzzystrmatch	1.1	1.1	1.1
hll	2.15	2.15	2.15
hstore	1.7	1.7	1.7
hstore_plperl	1.0	1.0	1.0
insert_username (contrib-spi)	N/A	1.0	1.0
intagg	1.1	1.1	1.1
intarray	1.3	1.3	1.3
ip4r	2.4	2.4	2.4
isn	1.2	1.2	1.2
jsonb_plperl	1.0	1.0	1.0
log_fdw	1.2	1.2	1.2
ltree	1.2	1.2	1.2
moddatetime (contrib-spi)	N/A	1.0	1.0
oracle_fdw	2.3.0	2.3.0	2.3.0
orafce	3.16	3.16	3.16
pg_bigm	1.2	1.2	1.2
pg_buffercache	1.3	1.3	1.3
pg_cron	1.3	1.3	1.4.1
pg_freespacemap	1.2	1.2	1.2
pg_hint_plan	1.3.7	1.3.7	1.3.7
pg_partman	4.5.1	4.5.1	4.5.1

Extension	13.3	13.4	13.5
pg_prewarm	1.2	1.2	1.2
pg_proctab	0.0.9	0.0.9	0.0.9
pg_repack	1.4.6	1.4.6	1.4.6
pg_similarity	1.0	1.0	1.0
pg_stat_statements	1.8	1.8	1.8
pg_trgm	1.5	1.5	1.5
pg_visibility	1.2	1.2	1.2
pgaudit	1.5	1.5	1.5
pgcrypto	1.3	1.3	1.3
pglogical	2.3.3	2.4.0	2.4.0
pglogical_origin	1.0.0	1.0.0	1.0.0
pgrouting	3.1.0	3.1.3	3.1.3
pgrowlocks	1.2	1.2	1.2
pgstattuple	1.5	1.5	1.5
pgtap	1.1.0	1.1.0	1.1.0
plcoffee	2.3.15	2.3.15	2.3.15
plls	2.3.15	2.3.15	2.3.15
plperl	1.0	1.0	1.0
plpgsql	1.0	1.0	1.0
plprofiler	4.1	4.1	4.1
pltcl	1.0	1.0	1.0
plv8	2.3.15	2.3.15	2.3.15
PostGIS	3.0.3	3.1.4	3.1.4
postgis_raster	3.0.3	3.1.4	3.1.4
postgis_tiger_geocoder	3.0.3	3.1.4	3.1.4
postgis_topology	3.0.3	3.1.4	3.1.4
postgres_fdw	1.0	1.0	1.0
prefix	1.2.0	1.2.0	1.2.0
rdkit	3.8	3.8	3.8
rds_activity_stream	1.3	1.3	1.3
rds_tools	1.0	1.0	1.0

Extension	13.3	13.4	13.5
refint (contrib-spi)	N/A	1.0	1.0
sslinfo	1.2	1.2	1.2
tablefunc	1.0	1.0	1.0
test_parser	1.0	1.0	1.0
tsm_system_rows	1.0	1.0	1.0
tsm_system_time	1.0	1.0	1.0
unaccent	1.1	1.1	1.1
uuid-ossp	1.1	1.1	1.1
wal2json	2.3	2.3	2.3

Extensions supported for Aurora PostgreSQL 12.x

The following table shows the PostgreSQL extension versions that are currently supported on Aurora PostgreSQL versions 12.x. "NA" indicates that the extension isn't available for that PostgreSQL version. For more information about PostgreSQL extensions, see [Packaging Related Objects into an Extension](#) in the PostgreSQL documentation.

Extension	12.4	12.6	12.7	12.8	12.9
address_standardizer	3.0.2	3.0.2	3.0.2	3.0.2	3.0.2
address_standardizer_data_us	3.0.2	3.0.2	3.0.2	3.0.2	3.0.2
amcheck	1.2	1.2	1.2	1.2	1.2
apg_plan_mgmt (p.21079)	2.0	2.0	2.0	2.0	2.0
aurora_stat_utils	1.0	1.0	1.0	1.0	1.0
aws_commons	1.2	1.2	1.2	1.2	1.2
aws_lambda	1.0	1.0	1.0	1.0	1.0
aws_ml	1.0	1.0	1.0	1.0	1.0
aws_s3	1.1	1.1	1.1	1.1	1.1
bloom	1.0	1.0	1.0	1.0	1.0
btree_gin	1.3	1.3	1.3	1.3	1.3
btree_gist	1.5	1.5	1.5	1.5	1.5
citext	1.6	1.6	1.6	1.6	1.6
cube	1.4	1.4	1.4	1.4	1.4
dblink	1.2	1.2	1.2	1.2	1.2
dict_int	1.0	1.0	1.0	1.0	1.0

Extension	12.4	12.6	12.7	12.8	12.9
dict_xsyn	1.0	1.0	1.0	1.0	1.0
earthdistance	1.1	1.1	1.1	1.1	1.1
fuzzystrmatch	1.1	1.1	1.1	1.1	1.1
hll	2.14	2.14	2.14	2.14	2.14
hstore	1.6	1.6	1.6	1.6	1.6
hstore_plperl	1.0	1.0	1.0	1.0	1.0
intagg	1.1	1.1	1.1	1.1	1.1
intarray	1.2	1.2	1.2	1.2	1.2
ip4r	2.4	2.4	2.4	2.4	2.4
isn	1.2	1.2	1.2	1.2	1.2
jsonb_plperl	1.0	1.0	1.0	1.0	1.0
log_fdw	1.1	1.1	1.1	1.1	1.1
ltree	1.1	1.1	1.1	1.1	1.1
oracle_fdw	NA	NA	2.3.0	2.3.0	2.3.0
orafce	3.8	3.8	3.16	3.16	3.16
pg_bigm	NA	1.2	1.2	1.2	1.2
pg_buffercache	1.3	1.3	1.3	1.3	1.3
pg_cron	NA	1.3	1.3.1	1.3.1	1.4
pg_freespacemap	1.2	1.2	1.2	1.2	1.2
pg_hint_plan	1.3.5	1.3.5	1.3.5	1.3.5	1.3.7
pg_partman	NA	4.4.0	4.5.1	4.5.1	4.5.1
pg_prewarm	1.2	1.2	1.2	1.2	1.2
pg_proctab	NA	0.0.9	0.0.9	0.0.9	0.0.9
pg_repack	1.4.5	1.4.5	1.4.5	1.4.5	1.4.5
pg_similarity	1.0	1.0	1.0	1.0	1.0
pg_stat_statements	1.7	1.7	1.7	1.7	1.7
pg_trgm	1.4	1.4	1.4	1.4	1.4
pg_visibility	1.2	1.2	1.2	1.2	1.2
pgaudit	1.4	1.4	1.4	1.4	1.4
pgcrypto	1.3	1.3	1.3	1.3	1.3
pglogical	2.3.2	2.3.2	2.3.2	2.4.0	2.4.0

Extension	12.4	12.6	12.7	12.8	12.9
pglogical_origin	1.0.0	1.0.0	1.0.0	1.0.0	1.0.0
pgrouting	3.0.3	3.0.3	3.0.3	3.0.3	3.0.3
pgrowlocks	1.2	1.2	1.2	1.2	1.2
pgstattuple	1.5	1.5	1.5	1.5	1.5
pgTAP	1.1.0	1.1.0	1.1.0	1.1.0	1.1.0
plcoffee	2.3.14	2.3.14	2.3.14	2.3.14	2.3.14
plls	2.3.14	2.3.14	2.3.14	2.3.14	2.3.14
plperl	1.0	1.0	1.0	1.0	1.0
plpgsql	1.0	1.0	1.0	1.0	1.0
plprofiler	4.1	4.1	4.1	4.1	4.1
pltcl	1.0	1.0	1.0	1.0	1.0
plv8	2.3.14	2.3.14	2.3.14	2.3.14	2.3.14
PostGIS	3.0.2	3.0.2	3.0.3	3.1	3.1
postgis_raster	3.0.2	3.0.2	3.0.3	3.1	3.1
postgis_tiger_geocoder	3.0.2	3.0.2	3.0.3	3.1	3.1
postgis_topology	3.0.2	3.0.2	3.0.3	3.1	3.1
postgres_fdw	1.0	1.0	1.0	1.0	1.0
prefix	1.2.0	1.2.0	1.2.0	1.2.0	1.2.0
RDKit	3.8	3.8	3.8	3.8	3.8
rds_activity_stream	1.3	1.3	1.3	1.3	1.3
sslinfo	1.2	1.2	1.2	1.2	1.2
tablefunc	1.0	1.0	1.0	1.0	1.0
test_parser	1.0	1.0	1.0	1.0	1.0
tsm_system_rows	1.0	1.0	1.0	1.0	1.0
tsm_system_time	1.0	1.0	1.0	1.0	1.0
unaccent	1.1	1.1	1.1	1.1	1.1
uuid-ossp	1.1	1.1	1.1	1.1	1.1
wal2json	2.3	2.3	2.3	2.3	2.3

Extensions supported for Aurora PostgreSQL 11.x

The following table shows PostgreSQL extension versions currently supported on Aurora PostgreSQL versions 11.x. "NA" indicates that the extension isn't available for that PostgreSQL version. For more information about PostgreSQL extensions, see [Packaging Related Objects into an Extension](#).

Extension	11.4	11.6	11.7	11.8	11.9	11.11	11.12	11.13	11.14
address_standardizer	2.5.1	2.5.2	2.5.2	2.5.2	2.5.2	2.5.2	2.5.2	2.5.2	2.5.2
address_standardizer	2.5.1a_us	2.5.2	2.5.2	2.5.2	2.5.2	2.5.2	2.5.2	2.5.2	2.5.2
amcheck	1.1	1.1	1.1	1.1	1.1	1.1	1.1	1.1	1.1
apg_plan_mgmt (p. 1609)	1.0	2.0	2.0	2.0	2.0	2.0	2.0	2.0	2.0
aurora_stat_utils	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
aws_commons	1.0	1.1	1.1	1.1	1.2	1.2	1.2	1.2	1.2
aws_lambda	NA	NA	NA	NA	1.0	1.0	1.0	1.0	1.0
aws_ml	NA	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
aws_s3	1.0	1.1	1.1	1.1	1.1	1.1	1.1	1.1	1.1
bloom	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
btree_gin	1.3	1.3	1.3	1.3	1.3	1.3	1.3	1.3	1.3
btree_gist	1.5	1.5	1.5	1.5	1.5	1.5	1.5	1.5	1.5
citext	1.5	1.5	1.5	1.5	1.5	1.5	1.5	1.5	1.5
cube	1.4	1.4	1.4	1.4	1.4	1.4	1.4	1.4	1.4
dblink	1.2	1.2	1.2	1.2	1.2	1.2	1.2	1.2	1.2
dict_int	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
dict_xsyn	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
earthdistance	1.1	1.1	1.1	1.1	1.1	1.1	1.1	1.1	1.1
fuzzystrmatch	1.1	1.1	1.1	1.1	1.1	1.1	1.1	1.1	1.1
hll	2.11	2.11	2.11	2.11	2.11	2.11	2.11	2.11	2.11
hstore	1.5	1.5	1.5	1.5	1.5	1.5	1.5	1.5	1.5
hstore_plperl	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
intagg	1.1	1.1	1.1	1.1	1.1	1.1	1.1	1.1	1.1
intarray	1.2	1.2	1.2	1.2	1.2	1.2	1.2	1.2	1.2
ip4r	2.2	2.2	2.2	2.2	2.2	2.2	2.2	2.2	2.2
isn	1.2	1.2	1.2	1.2	1.2	1.2	1.2	1.2	1.2
jsonb_plperl	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0

Extension	11.4	11.6	11.7	11.8	11.9	11.11	11.12	11.13	11.14
log_fdw	1.1	1.1	1.1	1.1	1.1	1.1	1.1	1.1	1.1
ltree	1.1	1.1	1.1	1.1	1.1	1.1	1.1	1.1	1.1
orafce	3.7	3.7	3.8	3.8	3.8	3.8	3.16	3.16	3.16
pg_bigm	NA	NA	NA	NA	NA	1.2	1.2	1.2	1.2
pg_buffercache	1.3	1.3	1.3	1.3	1.3	1.3	1.3	1.3	1.3
pg_freespacemap	1.2	1.2	1.2	1.2	1.2	1.2	1.2	1.2	1.2
pg_hint_plan	1.3.4	1.3.4	1.3.4	1.3.5	1.3.5	1.3.5	1.3.5	1.3.5	1.3.7
pg_prewarm	1.2	1.2	1.2	1.2	1.2	1.2	1.2	1.2	1.2
pg_proctab	NA	NA	NA	NA	NA	0.0.9	0.0.9	0.0.9	0.0.9
pg_repack	1.4.4	1.4.4	1.4.4	1.4.4	1.4.4	1.4.4	1.4.4	1.4.4	1.4.4
pg_similarity	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
pg_stat_statements	1.6	1.6	1.6	1.6	1.6	1.6	1.6	1.6	1.6
pg_trgm	1.4	1.4	1.4	1.4	1.4	1.4	1.4	1.4	1.4
pg_visibility	1.2	1.2	1.2	1.2	1.2	1.2	1.2	1.2	1.2
pgaudit	1.3	1.3	1.3	1.3	1.3.1	1.3.1	1.3.1	1.3.1	1.3.1
pgcrypto	1.3	1.3	1.3	1.3	1.3	1.3	1.3	1.3	1.3
pglogical	NA	NA	NA	2.2.2	2.2.2	2.2.2	2.2.2	2.2.2	2.2.2
pglogical_origin	NA	NA	NA	1.0.0	1.0.0	1.0.0	1.0.0	1.0.0	1.0.0
pgrouting	2.6.1	2.6.1	2.6.1	2.6.1	2.6.1	2.6.1	2.6.1	2.6.1	2.6.1
pgrowlocks	1.2	1.2	1.2	1.2	1.2	1.2	1.2	1.2	1.2
pgstattuple	1.5	1.5	1.5	1.5	1.5	1.5	1.5	1.5	1.5
pgTAP	1.0.0	1.0.0	1.1.0	1.1.0	1.1.0	1.1.0	1.1.0	1.1.0	1.1.0
plcoffee	2.3.8	2.3.8	2.3.8	2.3.14	2.3.14	2.3.14	2.3.14	2.3.14	2.3.14
plls	2.3.8	2.3.8	2.3.8	2.3.14	2.3.14	2.3.14	2.3.14	2.3.14	2.3.14
plperl	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
plpgsql	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
plprofiler	NA	4.1	4.1	4.1	4.1	4.1	4.1	4.1	4.1
pltcl	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
plv8	2.3.8	2.3.8	2.3.8	2.3.14	2.3.14	2.3.14	2.3.14	2.3.14	2.3.14
PostGIS	2.5.1	2.5.2	2.5.2	2.5.2	2.5.2	2.5.2	2.5.2	3.1	3.1
postgis_tiger_geocoder	2.5.1	2.5.2	2.5.2	2.5.2	2.5.2	2.5.2	2.5.2	3.1	3.1

Extension	11.4	11.6	11.7	11.8	11.9	11.11	11.12	11.13	11.14
postgis_topology	2.5.1	2.5.2	2.5.2	2.5.2	2.5.2	2.5.2	2.5.2	3.1	3.1
postgres_fdw	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
prefix	1.2.0	1.2.0	1.2.0	1.2.0	1.2.0	1.2.0	1.2.0	1.2.0	1.2.0
RDKit	NA	NA	NA	3.8	3.8	3.8	3.8	3.8	3.8
rds_activity_stream	1.3	1.3	1.3	1.3	1.3	1.3	1.3	1.3	1.3
sslinfo	1.2	1.2	1.2	1.2	1.2	1.2	1.2	1.2	1.2
tablefunc	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
test_parser	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
tsm_system_rows	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
tsm_system_time	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
unaccent	1.1	1.1	1.1	1.1	1.1	1.1	1.1	1.1	1.1
uuid-ossp	1.1	1.1	1.1	1.1	1.1	1.1	1.1	1.1	1.1
wal2json	2.3	2.3	2.3	2.3	2.3	2.3	2.3	2.3	2.3

Extensions supported for Aurora PostgreSQL 10.x

The following table shows PostgreSQL extension versions currently supported on Aurora PostgreSQL versions 10.x. "NA" indicates that the extension isn't available for that PostgreSQL version. For more information about PostgreSQL extensions, see [Packaging Related Objects into an Extension](#).

Note

- The `adminpack` extension is no longer supported because it accesses the file system.
- The `plperlu` extension is no longer supported because it is an untrusted language extension.
- The `pltclu` extension is no longer supported because it is an untrusted language extension.

Extension	10.4	10.5	10.6	10.7	10.11	10.12	10.13	10.14	10.16	10.17	10.18	10.19
address_standardizer	2.4.4	2.4.4	2.4.4	2.4.4	2.4.4	2.4.4	2.4.4	2.4.4	2.4.4	2.4.4	2.4.4	2.4.4
address_standardizer_d2t4_4us	2.4.4	2.4.4	2.4.4	2.4.4	2.4.4	2.4.4	2.4.4	2.4.4	2.4.4	2.4.4	2.4.4	2.4.4
adminpack	1.1	1.1	1.1	NA								
amcheck	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
apg_plan_mgmt (p. 1670)	1.0.1	1.0.1	1.0.1	1.0.1	2.0	2.0	2.0	2.0	2.0	2.0	2.0	2.0
aurora_stat_utils	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
aws_commons	NA	NA	1.0	1.0	1.1	1.1	1.1	1.1	1.1	1.1	1.1	1.1
aws_ml	NA	NA	NA	NA	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
aws_s3	NA	NA	1.0	1.0	1.1	1.1	1.1	1.1	1.1	1.1	1.1	1.1

Extension	10.4	10.5	10.6	10.7	10.11	10.12	10.13	10.14	10.16	10.17	10.18	10.19
bloom	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
btree_gin	1.2	1.2	1.2	1.2	1.2	1.2	1.2	1.2	1.2	1.2	1.2	1.2
btree_gist	1.5	1.5	1.5	1.5	1.5	1.5	1.5	1.5	1.5	1.5	1.5	1.5
chkpass	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
citext	1.4	1.4	1.4	1.4	1.4	1.4	1.4	1.4	1.4	1.4	1.4	1.4
cube	1.2	1.2	1.2	1.2	1.2	1.2	1.2	1.2	1.2	1.2	1.2	1.2
dblink	1.2	1.2	1.2	1.2	1.2	1.2	1.2	1.2	1.2	1.2	1.2	1.2
dict_int	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
dict_xsyn	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
earthdistance	1.1	1.1	1.1	1.1	1.1	1.1	1.1	1.1	1.1	1.1	1.1	1.1
fuzzystrmatch	1.1	1.1	1.1	1.1	1.1	1.1	1.1	1.1	1.1	1.1	1.1	1.1
hll	2.10	2.10	2.10	2.10	2.10	2.10	2.10	2.10	2.10	2.10	2.10	2.10
hstore	1.4	1.4	1.4	1.4	1.4	1.4	1.4	1.4	1.4	1.4	1.4	1.4
hstore_plperl	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
hstore_plperlu	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
intagg	1.1	1.1	1.1	1.1	1.1	1.1	1.1	1.1	1.1	1.1	1.1	1.1
intarray	1.2	1.2	1.2	1.2	1.2	1.2	1.2	1.2	1.2	1.2	1.2	1.2
ip4r	2.1	2.1	2.1	2.1	2.1	2.1	2.1	2.1	2.1	2.1	2.1	2.1
isn	1.1	1.1	1.1	1.1	1.1	1.1	1.1	1.1	1.1	1.1	1.1	1.1
log_fdw	1.1	1.1	1.1	1.1	1.1	1.1	1.1	1.1	1.1	1.1	1.1	1.1
ltree	1.1	1.1	1.1	1.1	1.1	1.1	1.1	1.1	1.1	1.1	1.1	1.1
orafce	3.6	3.6	3.6	3.6	3.6	3.8	3.8	3.8	3.8	3.16	3.16	3.16
pg_buffercache	1.3	1.3	1.3	1.3	1.3	1.3	1.3	1.3	1.3	1.3	1.3	1.3
pg_freespacemap	1.2	1.2	1.2	1.2	1.2	1.2	1.2	1.2	1.2	1.2	1.2	1.2
pg_hint_plan	1.3.0	1.3.1	1.3.1	1.3.1	1.3.3	1.3.3	1.3.5	1.3.5	1.3.5	1.3.5	1.3.5	1.3.6
pg_prewarm	1.1	1.1	1.1	1.1	1.1	1.1	1.1	1.1	1.1	1.1	1.1	1.1
pg_repack	1.4.3	1.4.3	1.4.3	1.4.3	1.4.3	1.4.3	1.4.3	1.4.3	1.4.3	1.4.3	1.4.3	1.4.3
pg_similarity	NA	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
pg_stat_statements	1.5	1.5	1.6	1.6	1.6	1.6	1.6	1.6	1.6	1.6	1.6	1.6
pg_trgm	1.3	1.3	1.3	1.3	1.3	1.3	1.3	1.3	1.3	1.3	1.3	1.3
pg_visibility	1.2	1.2	1.2	1.2	1.2	1.2	1.2	1.2	1.2	1.2	1.2	1.2

Extension	10.4	10.5	10.6	10.7	10.11	10.12	10.13	10.14	10.16	10.17	10.18	10.19
pgaudit	1.2	1.2	1.2	1.2	1.2	1.2	1.2.1	1.2.1	1.2.1	1.2.1	1.2.1	1.2.1
pgcrypto	1.3	1.3	1.3	1.3	1.3	1.3	1.3	1.3	1.3	1.3	1.3	1.3
pglogical	NA	NA	NA	NA	NA	NA	2.2.2	2.2.2	2.2.2	2.2.2	2.2.2	2.2.2
pglogical_origin	NA	NA	NA	NA	NA	NA	1.0.0	1.0.0	1.0.0	1.0.0	1.0.0	1.0.0
pgrouting	2.5.2	2.5.2	2.5.2	2.5.2	2.5.2	2.5.2	2.5.2	2.5.2	2.5.2	2.5.2	2.5.2	2.5.2
pgrowlocks	1.2	1.2	1.2	1.2	1.2	1.2	1.2	1.2	1.2	1.2	1.2	1.2
pgstattuple	1.5	1.5	1.5	1.5	1.5	1.5	1.5	1.5	1.5	1.5	1.5	1.5
plcoffee	2.1.2	2.1.2	2.1.2	2.1.2	2.1.2	2.1.2	2.3.14	2.3.14	2.3.14	2.3.14	2.3.14	2.3.14
plls	2.1.2	2.1.2	2.1.2	2.1.2	2.1.2	2.1.2	2.3.14	2.3.14	2.3.14	2.3.14	2.3.14	2.3.14
plperl	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
plperlu	1.0	1.0	1.0	NA	NA	NA	NA	NA	NA	NA	NA	NA
plpgsql	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
plprofiler	NA	NA	NA	NA	4.0	4.1	4.1	4.1	4.1	4.1	4.1	4.1
pltcl	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
pltclu	1.0	1.0	1.0	NA	NA	NA	NA	NA	NA	NA	NA	NA
plv8	2.1.2	2.1.2	2.1.2	2.1.2	2.1.2	2.1.2	2.3.14	2.3.14	2.3.14	2.3.14	2.3.14	2.3.14
PostGIS	2.4.4	2.4.4	2.4.4	2.4.4	2.4.4	2.4.4	2.4.4	2.4.4	2.4.4	2.4.7	3.1	3.1
postgis_tiger_geocoder	2.4.4	2.4.4	2.4.4	2.4.4	2.4.4	2.4.4	2.4.4	2.4.4	2.4.4	2.4.7	3.1	3.1
postgis_topology	2.4.4	2.4.4	2.4.4	2.4.4	2.4.4	2.4.4	2.4.4	2.4.4	2.4.4	2.4.7	3.1	3.1
postgres_fdw	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
prefix	1.2.0	1.2.0	1.2.0	1.2.0	1.2.0	1.2.0	1.2.0	1.2.0	1.2.0	1.2.0	1.2.0	1.2.0
RDKit	NA	NA	NA	NA	NA	NA	3.8	3.8	3.8	3.8	3.8	3.8
rds_activity_stream	NA	1.2	1.2	1.2	1.2	1.2	1.2	1.2	1.2	1.2	1.2	1.2
sslinfo	1.2	1.2	1.2	1.2	1.2	1.2	1.2	1.2	1.2	1.2	1.2	1.2
tablefunc	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
test_parser	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
tsearch2	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
tsm_system_rows	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
tsm_system_time	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
unaccent	1.1	1.1	1.1	1.1	1.1	1.1	1.1	1.1	1.1	1.1	1.1	1.1
uuid-ossp	1.1	1.1	1.1	1.1	1.1	1.1	1.1	1.1	1.1	1.1	1.1	1.1

Extension	10.4	10.5	10.6	10.7	10.11	10.12	10.13	10.14	10.16	10.17	10.18	10.19
wal2json	NA	NA	NA	2.1	2.1	2.3	2.3	2.3	2.3	2.3	2.3	2.3

Aurora PostgreSQL apg_plan_mgmt extension versions

Topics

- [Version 2.0 of the Aurora PostgreSQL apg_plan_mgmt extension \(p. 1679\)](#)
- [Version 1.0.1 of the Aurora PostgreSQL apg_plan_mgmt extension \(p. 1680\)](#)

Version 2.0 of the Aurora PostgreSQL apg_plan_mgmt extension

You use the `apg_plan_mgmt` extension with query plan management. For more about how to install, upgrade, and use the `apg_plan_mgmt` extension, see [Managing query execution plans for Aurora PostgreSQL \(p. 1460\)](#).

The `apg_plan_mgmt` extension changes for version 2.0 include the following:

New extension features

1. You can now manage all queries inside SQL functions, whether they have parameters or not.
2. You can now manage all queries inside PL/pgSQL functions, whether they have parameters or not.
3. You can now manage queries in generic plans, whether they have parameters or not. To learn more about generic plans versus custom plans, see the `PREPARE` statement in the [PostgreSQL documentation](#).
4. You can now use query plan management to enforce the use of specific types of aggregate methods in query plans.

Extension improvements

1. You can now save plans with a size up to 8KB times the setting of the `max_worker_processes` parameter. Previously the maximum plan size was 8KB.
2. Fixed bugs for unnamed prepared statements such as those from JDBC.
3. Previously, when you tried to do `CREATE EXTENSION apg_plan_mgmt` when it is not loaded in the `shared_preload_libraries`, the PostgreSQL backend connection was dropped. Now, an error message prints and the connection is not dropped.
4. The default value of the `cardinality_error` in the `apg_plan_mgmt.plans` table is `NULL`, but it can be set to `-1` during the `apg_plan_mgmt.evolve_plan_baselines` function. `NULL` is now used consistently.
5. Plans are now saved for queries that refer to temporary tables.
6. The default maximum number of plans is increased from 1000 to 10000.
7. The following pgss parameters are deprecated because the automatic plan capture mode should be used instead of those parameters.
 - `apg_plan_mgmt.pgss_min_calls`
 - `apg_plan_mgmt.pgss_min_mean_time_ms`
 - `apg_plan_mgmt.pgss_min_stddev_time_ms`
 - `apg_plan_mgmt.pgss_min_total_time_ms`

Version 1.0.1 of the Aurora PostgreSQL `apg_plan_mgmt` extension

The `apg_plan_mgmt` extension changes for version 1.0.1 include the following:

New extension features

1. The `validate_plans` function has a new action value called `update_plan_hash`. This action updates the `plan_hash` ID for plans that can't be reproduced exactly. The `update_plan_hash` value also allows you to fix a plan by rewriting the SQL. You can then register the good plan as an Approved plan for the original SQL. Following is an example of using the `update_plan_hash` action.

```
UPDATE apg_plan_mgmt.plans SET plan_hash = new_plan_hash, plan_outline
= good_plan_outline
WHERE sql_hash = bad_plan_sql_hash AND plan_hash = bad_plan_plan_hash;
SELECT apg_plan_mgmt.validate_plans(bad_plan_sql_hash, bad_plan_plan_hash,
'update_plan_hash');
SELECT apg_plan_mgmt.reload();
```

2. A new `get_explain_stmt` function is available that generates the text of an `EXPLAIN` statement for the specified SQL statement. It includes the parameters `sql_hash`, `plan_hash` and `explain_options`.

The parameter `explain_options` can be any comma-separated list of valid `EXPLAIN` options, as shown following.

```
analyze,verbose,buffers,hashes,format json
```

If the parameter `explain_options` is `NULL` or an empty string, the `get_explain_stmt` function generates a simple `EXPLAIN` statement.

To create an `EXPLAIN` script for your workload or a portion of it, use the `\a`, `\t`, and `\o` options to redirect the output to a file. For example, you can create an `EXPLAIN` script for the top-ranked (top-K) statements by using the PostgreSQL `pg_stat_statements` view sorted by `total_time` in `DESC` order.

3. The precise location of the Gather parallel query operator is determined by costing, and may change slightly over time. To prevent these differences from invalidating the entire plan, query plan management now computes the same `plan_hash` even if the Gather operators move to different places in the plan tree.
4. Support is added for nonparameterized statements inside pl/pgsql functions.
5. Overhead is reduced when the `apg_plan_mgmt` extension is installed on multiple databases in the same cluster while two or more databases are being accessed concurrently. Also, this release fixed a bug in this area that caused plans to not be stored in shared memory.

Extension improvements

1. Improvements to the `evolve_plan_baselines` function.
 - a. The `evolve_plan_baselines` function now computes a `cardinality_error` metric over all nodes in the plan. Using this metric, you can identify any plan where the cardinality estimation error is large, and the plan quality is more doubtful. Long-running statements with high `cardinality_error` values are high-priority candidates for query tuning.
 - b. Reports generated by `evolve_plan_baselines` now include `sql_hash`, `plan_hash`, and the `plan status`.
 - c. You can now allow `evolve_plan_baselines` to approve previously Rejected plans.

- d. The meaning of `speedup_factor` for `evolve_plan_baselines` is now always relative to the baseline plan. For example, a value of 1.1 now means 10 percent faster than the baseline plan. A value of 0.9 means 10 percent slower than the baseline plan. The comparison is made using running time alone instead of total time.
 - e. The `evolve_plan_baselines` function now warms the cache in a new way. It does this by running the baseline plan, then running the baseline plan one more time, and then running the candidate plan once. Previously, `evolve_plan_baselines` ran the candidate plan twice. This approach added significantly to running time, especially for slow candidate plans. However, running the candidate plan twice is more reliable when the candidate plan uses an index that isn't used in the baseline plan.
2. Query plan management no longer saves plans that refer to system tables or views, temporary tables, or the query plan management's own tables.
 3. Bug fixes include caching a plan immediately when saved and fixing a bug that caused the back end to terminate.

Upgrading the PostgreSQL DB engine for Aurora PostgreSQL

When Aurora PostgreSQL supports a new version of a database engine, you can upgrade your DB clusters to the new version. There are two kinds of upgrades for PostgreSQL DB clusters: major version upgrades and minor version upgrades.

Major version upgrades can contain database changes that are not backward-compatible with existing applications. As a result, you must manually perform major version upgrades of your DB instances. You can initiate a major version upgrade by modifying your DB cluster. However, before you perform a major version upgrade, we recommend that you follow the steps described in [How to perform a major version upgrade \(p. 1683\)](#).

In contrast, *minor version upgrades* include only changes that are backward-compatible with existing applications. You can initiate a minor version upgrade manually by modifying your DB cluster. Or you can enable the **Auto minor version upgrade** option when creating or modifying a DB cluster. Doing so means that your DB cluster is automatically upgraded after Aurora PostgreSQL tests and approves the new version. For more details, see [Automatic minor version upgrades for PostgreSQL \(p. 1688\)](#). For information about manually performing a minor version upgrade, see [Manually upgrading the Aurora PostgreSQL engine \(p. 1686\)](#).

Aurora DB clusters that are configured as logical replication publishers or subscribers can't undergo a major version upgrade. Before upgrading, you need to stop replication and drop any logical slots. For more information, see [Stopping logical replication \(p. 1437\)](#).

For how to determine valid upgrade targets, see [Determining which engine version to upgrade to \(p. 1682\)](#).

Topics

- [Overview of upgrading Aurora PostgreSQL \(p. 1682\)](#)
- [Determining which engine version to upgrade to \(p. 1682\)](#)
- [How to perform a major version upgrade \(p. 1683\)](#)
- [Manually upgrading the Aurora PostgreSQL engine \(p. 1686\)](#)
- [In-place major upgrades for global databases \(p. 1687\)](#)
- [Automatic minor version upgrades for PostgreSQL \(p. 1688\)](#)
- [Upgrading PostgreSQL extensions \(p. 1689\)](#)

Overview of upgrading Aurora PostgreSQL

Major version upgrades can contain database changes that are not backward-compatible with previous versions of the database. This functionality can cause your existing applications to stop working correctly. As a result, Amazon Aurora doesn't apply major version upgrades automatically. To perform a major version upgrade, you modify your DB cluster manually.

To safely upgrade your DB instances, Aurora PostgreSQL uses the `pg_upgrade` utility described in the [PostgreSQL documentation](#). After the writer upgrade completes, each reader instance experiences a brief outage while it's upgraded to the new major version automatically.

Aurora PostgreSQL takes a DB cluster snapshot before a major version upgrade begins. It doesn't take a DB cluster snapshot before a minor version upgrade.

If you want to return to a previous version after a major version upgrade is complete, you can restore the DB cluster from this snapshot. You can also restore the DB cluster to a specific point in time before either a major or minor version upgrade started. For more information, see [Restoring from a DB cluster snapshot \(p. 497\)](#) or [Restoring a DB cluster to a specified time \(p. 537\)](#).

During the major version upgrade process, a cloned volume is allocated. If the upgrade fails for some reason, such as due to a schema incompatibility, Aurora PostgreSQL uses this clone to roll back the upgrade. Note, when more than 15 clones of a source volume are allocated, subsequent clones become full copies and will take longer. This can cause the upgrade process to take longer as well. If Aurora PostgreSQL rolls back the upgrade, be aware of the following:

- You may see billing entries and metrics for both the original volume and the cloned volume allocated during the upgrade. Aurora PostgreSQL will clean up the extra volume after the cluster backup retention window is beyond the time of the upgrade.
- The next cross region snapshot copy from this cluster will be a full copy instead of an incremental copy.

Determining which engine version to upgrade to

To determine which major engine version that you can upgrade your database to, use the `describe-db-engine-versions` CLI command. If you can't do a major version upgrade. You first upgrade to a minor version that has a major version upgrade path.

For example, the following command displays the major engine versions available for upgrading a DB cluster currently running the Aurora PostgreSQL engine version 10.11.

Example

For Linux, macOS, or Unix:

```
aws rds describe-db-engine-versions \
--engine aurora-postgresql \
--engine-version 10.11 \
--query 'DBEngineVersions[?IsMajorVersionUpgrade == `true`].
{EngineVersion:EngineVersion}' \
--output text
```

For Windows:

```
aws rds describe-db-engine-versions ^
--engine aurora-postgresql ^
--engine-version 10.11 ^
```

```
--query "DBEngineVersions[].ValidUpgradeTarget[?IsMajorVersionUpgrade == `true`].  
{EngineVersion:EngineVersion}" ^  
--output text
```

How to perform a major version upgrade

Major version upgrades can contain database changes that are not backward-compatible with previous versions of the database. This functionality can cause your existing applications to stop working correctly. As a result, Amazon Aurora doesn't apply major version upgrades automatically. To perform a major version upgrade, you modify your DB cluster manually.

The following Aurora PostgreSQL major version upgrades are available for Graviton2-based instances.

Current source version	Major upgrade targets
9.6.9 and higher minor versions	10.11 or higher minor versions
10.7 and higher minor versions	11.7 or higher minor versions
11.7 and higher minor versions	12.4 or higher minor versions
12.4 and higher minor versions	13.3 or higher minor versions

The following Aurora PostgreSQL major version upgrades are available for Intel-based instances.

Current source version	Major upgrade targets
9.6.9 and higher minor versions	10.11 or higher minor versions
10.7 and higher minor versions	11.7 or higher minor versions
11.7 and higher minor versions	12.4 or higher minor versions
12.7 and higher minor versions	13.3 or higher minor versions

Before applying an upgrade to your production DB clusters, make sure that you thoroughly test any upgrade to verify that your applications work correctly.

We recommend the following process when upgrading an Aurora PostgreSQL DB cluster:

1. Have a version-compatible parameter group ready.

If you are using a custom DB instance or DB cluster parameter group, you have two options:

- Specify the default DB instance, DB cluster parameter group, or both for the new DB engine version.
- Create your own custom parameter group for the new DB engine version.

If you associate a new DB instance or DB cluster parameter group as a part of the upgrade request, make sure to reboot the database after the upgrade completes to apply the parameters. If a DB instance needs to be rebooted to apply the parameter group changes, the instance's parameter group status shows `pending-reboot`. You can view an instance's parameter group status in the console or by using a CLI command such as `describe-db-instances` or `describe-db-clusters`.

2. Check for unsupported usage:

- Commit or roll back all open prepared transactions before attempting an upgrade. You can use the following query to verify that there are no open prepared transactions on your instance.

```
SELECT count(*) FROM pg_catalog.pg_prepared_xacts;
```

- Remove all uses of the *reg** data types before attempting an upgrade. Except for `regtype` and `regclass`, you can't upgrade the *reg** data types. The `pg_upgrade` utility can't persist this data type, which is used by Amazon Aurora to do the upgrade. For more information about the `pg_upgrade` utility, see the [PostgreSQL documentation](#).

To verify that there are no uses of unsupported *reg** data types, use the following query for each database.

```
SELECT count(*) FROM pg_catalog.pg_class c, pg_catalog.pg_namespace n,
pg_catalog.pg_attribute a
WHERE c.oid = a.attrelid
    AND NOT a.attisdropped
    AND a.atttypid IN ('pg_catalog.regproc'::pg_catalog.regtype,
'pg_catalog.regprocedure'::pg_catalog.regtype,
'pg_catalog.regoper'::pg_catalog.regtype,
'pg_catalog.regoperator'::pg_catalog.regtype,
'pg_catalog.regconfig'::pg_catalog.regtype,
'pg_catalog.regdictionary'::pg_catalog.regtype)
    AND c.relnamespace = n.oid
    AND n.nspname NOT IN ('pg_catalog', 'information_schema');
```

3. Perform a backup.

The upgrade process creates a DB cluster snapshot of your DB cluster during upgrading. If you also want to do a manual backup before the upgrade process, see [Creating a DB cluster snapshot \(p. 495\)](#) for more information.

4. Upgrade certain extensions to the latest available version before performing the major version upgrade. The extensions to update include the following:

- `pgRouting`
- `postgis_raster`
- `postgis_tiger_geocoder`
- `postgis_topology`
- `address_standardizer`
- `address_standardizer_data_us`

Run the following command for each extension that you are using.

```
ALTER EXTENSION PostgreSQL-extension UPDATE TO 'new-version'
```

For more information, see [Upgrading PostgreSQL extensions \(p. 1689\)](#).

5. If you're upgrading to version 11.x, drop the extensions that it doesn't support before performing the major version upgrade. The extensions to drop include:

- `chkpass`
- `tsearch2`

6. Drop unknown data types, depending on your target version.

PostgreSQL version 10 doesn't support the `unknown` data type. If a version 9.6 database uses the `unknown` data type, an upgrade to version 10 shows an error message such as the following.

```
Database instance is in a state that cannot be upgraded: PreUpgrade checks failed:  
The instance could not be upgraded because the 'unknown' data type is used in user  
tables.
```

Please remove all usages of the 'unknown' data type and try again."

To find the unknown data type in your database so that you can remove such columns or change them to supported data types, use the following SQL code for each database.

```
SELECT n.nspname, c.relname, a.attname
  FROM pg_catalog.pg_class c,
       pg_catalog.pg_namespace n,
       pg_catalog.pg_attribute a
 WHERE c.oid = a.attrelid AND NOT a.attisdropped AND
       a.atttypid = 'pg_catalog.unknown)::pg_catalog.regtyp AND
       c.relkind IN ('r','m','c') AND
       c.relnamespace = n.oid AND
       n.nspname !~ '^pg_temp_' AND
       n.nspname !~ '^pg_toast_temp_' AND n.nspname NOT IN ('pg_catalog',
       'information_schema');
```

7. Perform a dry run upgrade.

We highly recommend testing a major version upgrade on a duplicate of your production database before trying the upgrade on your production database. To create a duplicate test instance, you can either restore your database from a recent snapshot or clone your database. For more information, see [Restoring from a snapshot \(p. 498\)](#) or [Cloning a volume for an Aurora DB cluster \(p. 402\)](#).

For more information, see [Manually upgrading the Aurora PostgreSQL engine \(p. 1686\)](#).

8. Upgrade your production instance.

When your dry-run major version upgrade is successful, you should be able to upgrade your production database with confidence. For more information, see [Manually upgrading the Aurora PostgreSQL engine \(p. 1686\)](#).

Note

During the upgrade process, you can't do a point-in-time restore of your cluster. Aurora PostgreSQL takes a DB cluster snapshot during the upgrade process if your backup retention period is greater than 0. You can perform a point-in-time restore to times before the upgrade began and after the automatic snapshot of your instance has completed.

For information about an upgrade in progress, you can use Amazon RDS to view two logs that the pg_upgrade utility produces. These are pg_upgrade_internal.log and pg_upgrade_server.log. Amazon Aurora appends a timestamp to the file name for these logs. You can view these logs as you can any other log. For more information, see [Monitoring Amazon Aurora log files \(p. 695\)](#).

9. Upgrade PostgreSQL extensions. The PostgreSQL upgrade process doesn't upgrade any PostgreSQL extensions. For more information, see [Upgrading PostgreSQL extensions \(p. 1689\)](#).

After you complete a major version upgrade, we recommend the following:

- Run the ANALYZE operation to refresh the pg_statistic table.
- If you upgraded to PostgreSQL version 10, run REINDEX on any hash indexes you have. Hash indexes were changed in version 10 and must be rebuilt. To locate invalid hash indexes, run the following SQL for each database that contains hash indexes.

```
SELECT idx.indrelid::regclass AS table_name,
       idx.indexrelid::regclass AS index_name
  FROM pg_catalog.pg_index idx
       JOIN pg_catalog.pg_class cls ON cls.oid = idx.indexrelid
       JOIN pg_catalog.pg_am am ON am.oid = cls.relam
```

```
WHERE am.amname = 'hash'  
AND NOT idx.indisvalid;
```

- Consider testing your application on the upgraded database with a similar workload to verify that everything works as expected. After the upgrade is verified, you can delete this test instance.

Manually upgrading the Aurora PostgreSQL engine

To perform an upgrade of an Aurora PostgreSQL DB cluster, use the following instructions for the AWS Management Console, the AWS CLI, or the RDS API.

Note

If you're performing a minor upgrade on an Aurora global database, upgrade all of the secondary clusters before you upgrade the primary cluster.

Console

To upgrade the engine version of a DB cluster by using the console

1. Sign in to the AWS Management Console and open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. In the navigation pane, choose **Databases**, and then choose the DB cluster that you want to upgrade.
3. Choose **Modify**. The **Modify DB cluster** page appears.
4. For **Engine version**, choose the new version.
5. Choose **Continue** and check the summary of modifications.
6. To apply the changes immediately, choose **Apply immediately**. Choosing this option can cause an outage in some cases. For more information, see [Modifying an Amazon Aurora DB cluster \(p. 372\)](#).
7. On the confirmation page, review your changes. If they are correct, choose **Modify Cluster** to save your changes.

Or choose **Back** to edit your changes or **Cancel** to cancel your changes.

AWS CLI

To upgrade the engine version of a DB cluster, use the CLI `modify-db-cluster` command. Specify the following parameters:

- `--db-cluster-identifier` – the name of the DB cluster.
- `--engine-version` – the version number of the database engine to upgrade to. For information about valid engine versions, use the AWS CLI `describe-db-engine-versions` command.
- `--allow-major-version-upgrade` – a required flag when the `--engine-version` parameter is a different major version than the DB cluster's current major version.
- `--no-apply-immediately` – apply changes during the next maintenance window. To apply changes immediately, use `--apply-immediately`.

Example

For Linux, macOS, or Unix:

```
aws rds modify-db-cluster \  
  --db-cluster-identifier mydbcluster \  
  --engine-version new_version \  
  --allow-major-version-upgrade \  
  --no-apply-immediately
```

For Windows:

```
aws rds modify-db-cluster ^
--db-cluster-identifier mydbcluster ^
--engine-version new_version ^
--allow-major-version-upgrade ^
--no-apply-immediately
```

RDS API

To upgrade the engine version of a DB cluster, use the [ModifyDBCluster](#) operation. Specify the following parameters:

- `DBClusterIdentifier` – the name of the DB cluster, for example *mydbcluster*.
- `EngineVersion` – the version number of the database engine to upgrade to. For information about valid engine versions, use the [DescribeDBEngineVersions](#) operation.
- `AllowMajorVersionUpgrade` – a required flag when the `EngineVersion` parameter is a different major version than the DB cluster's current major version.
- `ApplyImmediately` – whether to apply changes immediately or during the next maintenance window. To apply changes immediately, set the value to `true`. To apply changes during the next maintenance window, set the value to `false`.

In-place major upgrades for global databases

For an Aurora global database, you upgrade the global database cluster. Aurora automatically upgrades all of the clusters at the same time and makes sure that they all run the same engine version. This requirement is because any changes to system tables, data file formats, and so on, are automatically replicated to all the secondary clusters.

Follow the instructions in [How to perform a major version upgrade \(p. 1683\)](#). When you specify what to upgrade, make sure to choose the global database cluster instead of one of the clusters it contains.

If you use the AWS Management Console, choose the item with the role **Global database**.

DB identifier	Role	
global-cluster	Global database	Aur
cluster1	Primary cluster	Aur
dbinstance-1	Writer instance	Aur
cluster-2	Secondary cluster	Aur
dbinstance-2	Reader instance	Aur

If you use the AWS CLI or RDS API, start the upgrade process by calling the [modify-global-cluster](#) command or [ModifyGlobalCluster](#) operation instead of `modify-db-cluster` or `ModifyDBCluster`.

Note

You can't perform a major version upgrade of the Aurora DB engine if the recovery point objective (RPO) feature is turned on. Before you upgrade the DB engine, make sure that this

feature is turned off. For more information about the RPO feature, see [Managing RPOs for Aurora PostgreSQL-based global databases \(p. 272\)](#).

Automatic minor version upgrades for PostgreSQL

For each PostgreSQL major version, one minor version is designated by Amazon Aurora as the automatic upgrade version. After a minor version has been tested and approved by Amazon Aurora, the minor version upgrade occurs automatically during your maintenance window. Aurora doesn't automatically set newer released minor versions as the automatic upgrade version. Before Aurora designates a newer automatic upgrade version, several criteria are considered, such as the following:

- Known security issues
- Bugs in the PostgreSQL community version
- Overall fleet stability since the minor version was released

You can use the following AWS CLI command and script to determine the current automatic upgrade minor versions.

```
aws rds describe-db-engine-versions --engine aurora-postgresql | grep -A 1 AutoUpgrade|  
grep -A 2 true |grep PostgreSQL | sort --unique | sed -e 's/"Description": "//g'
```

If no results are returned, there is no automatic minor version upgrade available and scheduled.

A PostgreSQL DB instance is automatically upgraded during your maintenance window if the following criteria are met:

- The DB cluster has the **Auto minor version upgrade** option turned on.
- The DB cluster is running a minor DB engine version that is less than the current automatic upgrade minor version.

If any of the DB instances in a cluster don't have the auto minor version upgrade setting turned on, Aurora doesn't automatically upgrade any of the instances in that cluster. Make sure to keep that setting consistent for all the DB instances in the cluster.

Turning on automatic minor version upgrades

To turn on automatic minor version upgrades for an Aurora PostgreSQL DB cluster, use the following instructions for the AWS Management Console, the AWS CLI, or the RDS API.

Console

Follow the general procedure to modify the DB instances in your cluster, as described in [Modify a DB instance in a DB cluster \(p. 373\)](#). Repeat this procedure for each DB instance in your cluster.

To use the console to implement automatic minor version upgrades for your cluster

1. Sign in to the Amazon RDS console, choose **Databases**, and find the DB cluster where you want to turn automatic minor version upgrade on or off.
2. Choose each DB instance in the DB cluster that you want to modify. Apply the following change for each DB instance in sequence:
 - a. Choose **Modify**.
 - b. In the **Maintenance** section, select the **Enable auto minor version upgrade** box.
 - c. Choose **Continue** and check the summary of modifications.
 - d. (Optional) Choose **Apply immediately** to apply the changes immediately.

- e. On the confirmation page, choose **Modify DB instance**.

AWS CLI

To use the CLI to implement minor version upgrades, use the [modify-db-instance](#) command.

When you call the [modify-db-instance](#) AWS CLI command, specify the name of your DB instance for the `--db-instance-identifier` option and `true` for the `--auto-minor-version-upgrade` option. Optionally, specify the `--apply-immediately` option to immediately turn this setting on for your DB instance. Run a separate `modify-db-instance` command for each DB instance in the cluster.

You can use a CLI command such as the following to check the status of **Enable auto minor version upgrade** for all of the DB instances in your Aurora PostgreSQL clusters.

```
aws rds describe-db-instances \
--query '*[]'.
{DBClusterIdentifier:DBClusterIdentifier,DBInstanceIdentifier:DBInstanceIdentifier,AutoMinorVersionUpgrade}
```

That command produces output similar to the following.

```
[  
 {  
   "DBInstanceIdentifier": "db-t2-medium-instance",  
   "DBClusterIdentifier": "cluster-57-2020-06-03-6411",  
   "AutoMinorVersionUpgrade": true  
 },  
 {  
   "DBInstanceIdentifier": "db-t2-small-original-size",  
   "DBClusterIdentifier": "cluster-57-2020-06-03-6411",  
   "AutoMinorVersionUpgrade": false  
 },  
 {  
   "DBInstanceIdentifier": "instance-2020-05-01-2332",  
   "DBClusterIdentifier": "cluster-57-2020-05-01-4615",  
   "AutoMinorVersionUpgrade": true  
 },  
 ... output omitted ...
```

RDS API

To use the API to implement minor version upgrades, use the [ModifyDBInstance](#) operation.

Call the [ModifyDBInstance](#) API operation, and specify the name of your DB cluster for the `DBInstanceIdentifier` parameter and `true` for the `AutoMinorVersionUpgrade` parameter. Optionally, set the `ApplyImmediately` parameter to `true` to immediately turn this setting on for your DB instance. Call a separate `ModifyDBInstance` operation for each DB instance in the cluster.

Upgrading PostgreSQL extensions

A PostgreSQL engine upgrade doesn't automatically upgrade any PostgreSQL extensions. Installing PostgreSQL extensions requires `rds_superuser` privileges, and the permissions are typically delegated to only those users (roles) that use the extension. This means that upgrading all extensions in an Aurora PostgreSQL DB instance after a database engine upgrade might involve many different users (roles). Keep this in mind also if you want to automate the upgrade process by using scripts. For more information about PostgreSQL privileges and roles, see [Security with Amazon Aurora PostgreSQL \(p. 1276\)](#).

Note

If you are running the PostGIS extension in your Amazon RDS PostgreSQL DB instance, see [PostGIS_Extensions_Upgrade](#) in the PostGIS documentation to upgrade the extension.

To update an extension after an engine upgrade, use the `ALTER EXTENSION UPDATE` command.

```
ALTER EXTENSION extension_name UPDATE TO 'new_version';
```

To list your currently installed extensions, use the PostgreSQL `pg_extension` catalog in the following command.

```
SELECT * FROM pg_extension;
```

To view a list of the specific extension versions that are available for your installation, use the PostgreSQL `pg_available_extension_versions` view in the following command.

```
SELECT * FROM pg_available_extension_versions;
```

Aurora PostgreSQL long-term support (LTS) releases

Each new Aurora PostgreSQL version remains available for a certain amount of time for you to use when you create or upgrade a DB cluster. After this period, you must upgrade any clusters that use that version. You can manually upgrade your cluster before the support period ends, or Aurora can automatically upgrade it for you when its Aurora PostgreSQL version is no longer supported.

Aurora designates certain Aurora PostgreSQL versions as long-term support (LTS) releases. Database clusters that use LTS releases can stay on the same version longer and undergo fewer upgrade cycles than clusters that use non-LTS releases. LTS minor versions include only bug fixes (through patch versions); an LTS version doesn't include new features released after its introduction.

Once a year, DB clusters running on an LTS minor version are patched to the latest patch version of the LTS release. We do this patching to help ensure that you benefit from cumulative security and stability fixes. We might patch an LTS minor version more frequently if there are critical fixes, such as for security, that need to be applied.

Note

To remain on an LTS minor version for the duration of its lifecycle, make sure to turn off **Auto minor version upgrade** for your DB instances. To avoid automatically upgrading your DB cluster from the LTS minor version, set **Auto minor version upgrade** to **No** on all DB instances in your Aurora cluster.

We recommend that you upgrade to the latest release, instead of using the LTS release, for most of your Aurora PostgreSQL clusters. Doing so takes advantage of Aurora as a managed service and gives you access to the latest features and bug fixes. LTS releases are intended for clusters with the following characteristics:

- You can't afford downtime on your Aurora PostgreSQL application for upgrades outside of rare occurrences for critical patches.
- The testing cycle for the cluster and associated applications takes a long time for each update to the Aurora PostgreSQL database engine.
- The database version for your Aurora PostgreSQL cluster has all the DB engine features and bug fixes that your application needs.

The current LTS releases for Aurora PostgreSQL are as follows:

- PostgreSQL 12.9 (Aurora PostgreSQL version 12.9.1). It was released on February 25, 2022. For more information, see [Aurora PostgreSQL release 12.9 \(p. 1604\)](#).
- PostgreSQL 11.9 (Aurora PostgreSQL release 3.4. It was released on December 11, 2020. For more information about this version, see [PostgreSQL 11.9, Aurora PostgreSQL release 3.4 \(p. 1613\)](#)

For information about how to identify Aurora and database engine versions, see [Identifying versions of Amazon Aurora PostgreSQL \(p. 1598\)](#).

Best practices with Amazon Aurora

Following, you can find information on general best practices and options for using or migrating data to an Amazon Aurora DB cluster.

Some of the best practices for Amazon Aurora are specific to a particular database engine. For more information about Aurora best practices specific to a database engine, see the following.

Database engine	Best practices
Amazon Aurora MySQL	See Best practices with Amazon Aurora MySQL (p. 1033)
Amazon Aurora PostgreSQL	See Best practices with Amazon Aurora PostgreSQL (p. 1423)

Note

For common recommendations for Aurora, see [Viewing Amazon Aurora recommendations \(p. 558\)](#).

Topics

- [Basic operational guidelines for Amazon Aurora \(p. 1692\)](#)
- [DB instance RAM recommendations \(p. 1692\)](#)
- [Monitoring Amazon Aurora \(p. 1693\)](#)
- [Working with DB parameter groups and DB cluster parameter groups \(p. 1693\)](#)
- [Amazon Aurora best practices presentation video \(p. 1693\)](#)

Basic operational guidelines for Amazon Aurora

The following are basic operational guidelines that everyone should follow when working with Amazon Aurora. The Amazon RDS Service Level Agreement requires that you follow these guidelines:

- Monitor your memory, CPU, and storage usage. You can set up Amazon CloudWatch to notify you when usage patterns change or when you approach the capacity of your deployment. This way, you can maintain system performance and availability.
- If your client application is caching the Domain Name Service (DNS) data of your DB instances, set a time-to-live (TTL) value of less than 30 seconds. The underlying IP address of a DB instance can change after a failover. Thus, caching the DNS data for an extended time can lead to connection failures if your application tries to connect to an IP address that no longer is in service. Aurora DB clusters with multiple read replicas can experience connection failures also when connections use the reader endpoint and one of the read replica instances is in maintenance or is deleted.
- Test failover for your DB cluster to understand how long the process takes for your use case. Testing failover can help you ensure that the application that accesses your DB cluster can automatically connect to the new DB cluster after failover.

DB instance RAM recommendations

To optimize performance, allocate enough RAM so that your working set resides almost completely in memory. To determine whether your working set is almost all in memory, examine the following metrics in Amazon CloudWatch:

- **VolumeReadIOPS** – This metric measures the average number of read I/O operations from a cluster volume, reported at 5-minute intervals. The value of **VolumeReadIOPS** should be small and stable. In some cases, you might find your read I/O is spiking or is higher than usual. If so, investigate the DB instances in your DB cluster to see which DB instances are causing the increased I/O.

Tip

If your Aurora MySQL cluster uses parallel query, you might see an increase in **VolumeReadIOPS** values. Parallel queries don't use the buffer pool. Thus, although the queries are fast, this optimized processing can result in an increase in read operations and associated charges.

- **BufferCacheHitRatio** – This metric measures the percentage of requests that are served by the buffer cache of a DB instance in your DB cluster. This metric gives you an insight into the amount of data that is being served from memory. If the hit ratio is low, it's a good indication that your queries on this DB instance are going to disk more often than not. In this case, investigate your workload to see which queries are causing this behavior.

If, after investigating your workload, you find that you need more memory, consider scaling up the DB instance class to a class with more RAM. After doing so, you can investigate the metrics discussed preceding and continue to scale up as necessary. If your Aurora cluster is larger than 40 TB, don't use db.t2 or db.t3 instance classes. For more information about monitoring a DB cluster, see [Viewing metrics in the Amazon RDS console \(p. 563\)](#).

Monitoring Amazon Aurora

Amazon Aurora provides a variety of Amazon CloudWatch metrics that you can monitor to determine the health and performance of your Aurora DB cluster. You can use various tools, such as the AWS Management Console, AWS CLI, and CloudWatch API, to view Aurora metrics. For more information, see [Monitoring metrics in an Amazon Aurora cluster \(p. 541\)](#).

Working with DB parameter groups and DB cluster parameter groups

We recommend that you try out DB parameter group and DB cluster parameter group changes on a test DB cluster before applying parameter group changes to your production DB cluster. Improperly setting DB engine parameters can have unintended adverse effects, including degraded performance and system instability.

Always use caution when modifying DB engine parameters, and back up your DB cluster before modifying a DB parameter group. For information about backing up your DB cluster, see [Backing up and restoring an Amazon Aurora DB cluster \(p. 490\)](#).

Amazon Aurora best practices presentation video

The 2016 AWS Summit conference in Chicago included a presentation on best practices for creating and configuring an Amazon Aurora DB cluster to be more secure and highly available. For a video of the presentation, see [Amazon Aurora best practices](#) on the AWS YouTube channel.

Performing a proof of concept with Amazon Aurora

Following, you can find an explanation of how to set up and run a proof of concept for Aurora. A *proof of concept* is an investigation that you do to see if Aurora is a good fit with your application. The proof of concept can help you understand Aurora features in the context of your own database applications and how Aurora compares with your current database environment. It can also show what level of effort you need to move data, port SQL code, tune performance, and adapt your current management procedures.

In this topic, you can find an overview and a step-by-step outline of the high-level procedures and decisions involved in running a proof of concept, listed following. For detailed instructions, you can follow links to the full documentation for specific subjects.

Overview of an Aurora proof of concept

When you conduct a proof of concept for Amazon Aurora, you learn what it takes to port your existing data and SQL applications to Aurora. You exercise the important aspects of Aurora at scale, using a volume of data and activity that's representative of your production environment. The objective is to feel confident that the strengths of Aurora match up well with the challenges that cause you to outgrow your previous database infrastructure. At the end of a proof of concept, you have a solid plan to do larger-scale performance benchmarking and application testing. At this point, you understand the biggest work items on your way to a production deployment.

The following advice about best practices can help you avoid common mistakes that cause problems during benchmarking. However, this topic doesn't cover the step-by-step process of performing benchmarks and doing performance tuning. Those procedures vary depending on your workload and the Aurora features that you use. For detailed information, consult performance-related documentation such as [Managing performance and scaling for Aurora DB clusters \(p. 396\)](#), [Amazon Aurora MySQL performance enhancements \(p. 746\)](#), [Managing Amazon Aurora PostgreSQL \(p. 1360\)](#), and [Monitoring DB load with Performance Insights on Amazon Aurora \(p. 573\)](#).

The information in this topic applies mainly to applications where your organization writes the code and designs the schema and that support the MySQL and PostgreSQL open-source database engines. If you're testing a commercial application or code generated by an application framework, you might not have the flexibility to apply all of the guidelines. In such cases, check with your AWS representative to see if there are Aurora best practices or case studies for your type of application.

1. Identify your objectives

When you evaluate Aurora as part of a proof of concept, you choose what measurements to make and how to evaluate the success of the exercise.

You must ensure that all of the functionality of your application is compatible with Aurora. Because Aurora major versions are wire-compatible with corresponding major versions of MySQL and PostgreSQL, most applications developed for those engines are also compatible with Aurora. However, you must still validate compatibility on a per-application basis.

For example, some of the configuration choices that you make when you set up an Aurora cluster influence whether you can or should use particular database features. You might start with the most general-purpose kind of Aurora cluster, known as *provisioned*. You might then decide if a specialized configuration such as serverless or parallel query offers benefits for your workload.

Use the following questions to help identify and quantify your objectives:

- Does Aurora support all functional use cases of your workload?
- What dataset size or load level do you want? Can you scale to that level?
- What are your specific query throughput or latency requirements? Can you reach them?
- What is the minimum acceptable amount of planned or unplanned downtime for your workload? Can you achieve it?
- What are the necessary metrics for operational efficiency? Can you accurately monitor them?
- Does Aurora support your specific business goals, such as cost reduction, increase in deployment, or provisioning speed? Do you have a way to quantify these goals?
- Can you meet all security and compliance requirements for your workload?

Take some time to build knowledge about Aurora database engines and platform capabilities, and review the service documentation. Take note of all the features that can help you achieve your desired outcomes. One of these might be workload consolidation, described in the AWS Database Blog post [How to plan and optimize Amazon Aurora with MySQL compatibility for consolidated workloads](#). Another might be demand-based scaling, described in [Using Amazon Aurora Auto Scaling with Aurora replicas \(p. 427\)](#) in the *Amazon Aurora User Guide*. Others might be performance gains or simplified database operations.

2. Understand your workload characteristics

Evaluate Aurora in the context of your intended use case. Aurora is a good choice for online transaction processing (OLTP) workloads. You can also run reports on the cluster that holds the real-time OLTP data without provisioning a separate data warehouse cluster. You can recognize if your use case falls into these categories by checking for the following characteristics:

- High concurrency, with dozens, hundreds, or thousands of simultaneous clients.
- Large volume of low-latency queries (milliseconds to seconds).
- Short, real-time transactions.
- Highly selective query patterns, with index-based lookups.
- For HTAP, analytical queries that can take advantage of Aurora parallel query.

One of the key factors affecting your database choices is the velocity of the data. *High velocity* involves data being inserted and updated very frequently. Such a system might have thousands of connections and hundreds of thousands of simultaneous queries reading from and writing to a database. Queries in high-velocity systems usually affect a relatively small number of rows, and typically access multiple columns in the same row.

Aurora is designed to handle high-velocity data. Depending on the workload, an Aurora cluster with a single r4.16xlarge DB instance can process more than 600,000 SELECT statements per second. Again depending on workload, such a cluster can process 200,000 INSERT, UPDATE, and DELETE statements per second. Aurora is a row store database and is ideally suited for high-volume, high-throughput, and highly parallelized OLTP workloads.

Aurora can also run reporting queries on the same cluster that handles the OLTP workload. Aurora supports up to 15 [replicas \(p. 70\)](#), each of which is on average within 10–20 milliseconds of the primary

instance. Analysts can query OLTP data in real time without copying the data to a separate data warehouse cluster. With Aurora clusters using the parallel query feature, you can offload much of the processing, filtering, and aggregation work to the massively distributed Aurora storage subsystem.

Use this planning phase to familiarize yourself with the capabilities of Aurora, other AWS services, the AWS Management Console, and the AWS CLI. Also, check how these work with the other tooling that you plan to use in the proof of concept.

3. Practice with the AWS Management Console or AWS CLI

As a next step, practice with the AWS Management Console or the AWS CLI, to become familiar with these tools and with Aurora.

Practice with the AWS Management Console

The following initial activities with Aurora database clusters are mainly so you can familiarize yourself with the AWS Management Console environment and practice setting up and modifying Aurora clusters. If you use the MySQL-compatible and PostgreSQL-compatible database engines with Amazon RDS, you can build on that knowledge when you use Aurora.

By taking advantage of the Aurora shared storage model and features such as replication and snapshots, you can treat entire database clusters as another kind of object that you freely manipulate. You can set up, tear down, and change the capacity of Aurora clusters frequently during the proof of concept. You aren't locked into early choices about capacity, database settings, and physical data layout.

To get started, set up an empty Aurora cluster. Choose the **provisioned** capacity type and **regional** location for your initial experiments.

Connect to that cluster using a client program such as a SQL command-line application. Initially, you connect using the cluster endpoint. You connect to that endpoint to perform any write operations, such as data definition language (DDL) statements and extract, transform, load (ETL) processes. Later in the proof of concept, you connect query-intensive sessions using the reader endpoint, which distributes the query workload among multiple DB instances in the cluster.

Scale the cluster out by adding more Aurora Replicas. For those procedures, see [Replication with Amazon Aurora \(p. 70\)](#). Scale the DB instances up or down by changing the AWS instance class. Understand how Aurora simplifies these kinds of operations, so that if your initial estimates for system capacity are inaccurate, you can adjust later without starting over.

Create a snapshot and restore it to a different cluster.

Examine cluster metrics to see activity over time, and how the metrics apply to the DB instances in the cluster.

It's useful to become familiar with how to do these things through the AWS Management Console in the beginning. After you understand what you can do with Aurora, you can progress to automating those operations using the AWS CLI. In the following sections, you can find more details about the procedures and best practices for these activities during the proof-of-concept period.

Practice with the AWS CLI

We recommend automating deployment and management procedures, even in a proof-of-concept setting. To do so, become familiar with the AWS CLI if you're not already. If you use the MySQL-

compatible and PostgreSQL-compatible database engines with Amazon RDS, you can build on that knowledge when you use Aurora.

Aurora typically involves groups of DB instances arranged in clusters. Thus, many operations involve determining which DB instances are associated with a cluster and then performing administrative operations in a loop for all the instances.

For example, you might automate steps such as creating Aurora clusters, then scaling them up with larger instance classes or scaling them out with additional DB instances. Doing so helps you to repeat any stages in your proof of concept and explore what-if scenarios with different kinds or configurations of Aurora clusters.

Learn the capabilities and limitations of infrastructure deployment tools such as AWS CloudFormation. You might find activities that you do in a proof-of-concept context aren't suitable for production use. For example, the AWS CloudFormation behavior for modification is to create a new instance and delete the current one, including its data. For more details on this behavior, see [Update behaviors of stack resources](#) in the *AWS CloudFormation User Guide*.

4. Create your Aurora cluster

With Aurora, you can explore what-if scenarios by adding DB instances to the cluster and scaling up the DB instances to more powerful instance classes. You can also create clusters with different configuration settings to run the same workload side by side. With Aurora, you have a lot of flexibility to set up, tear down, and reconfigure DB clusters. Given this, it's helpful to practice these techniques in the early stages of the proof-of-concept process. For the general procedures to create Aurora clusters, see [Creating an Amazon Aurora DB cluster \(p. 125\)](#).

Where practical, start with a cluster using the following settings. Skip this step only if you have certain specific use cases in mind. For example, you might skip this step if your use case requires a specialized kind of Aurora cluster. Or you might skip it if you need a particular combination of database engine and version.

- Amazon Aurora.
- MySQL 5.7 compatibility. This combination of database engine and version has wide compatibility with other Aurora features and substantial customer usage for production applications.
- Turn off **Easy create**. For the proof of concept, we recommend that you be aware of all the settings you choose so that you can create identical or slightly different clusters later.
- Regional. The **Global** setting is for specific high availability scenarios. You can try it out later after your initial functional and performance experiments.
- One writer, multiple readers. This is the most widely used, general purpose kind of cluster. This setting persists for the life of the cluster. Thus, if you later do experiments with other kinds of clusters such as serverless or parallel query, you create other clusters and compare and contrast the results on each.
- Choose the **Dev/Test** template. This choice isn't significant for your proof-of-concept activities.
- For **DB instance class**, choose **Memory optimized classes** and one of the **xlarge** instance classes. You can adjust the instance class up or down later.
- Under **Multi-AZ Deployment**, choose **Create an Aurora Replica or Reader node in a different AZ**. Many of the most useful aspects of Aurora involve clusters of multiple DB instances. It makes sense to always start with at least two DB instances in any new cluster. Using a different Availability Zone for the second DB instance helps to test different high availability scenarios.
- When you pick names for the DB instances, use a generic naming convention. Don't refer to any cluster DB instance as the "master" or "writer," because different DB instances assume those roles as needed. We recommend using something like `clustername-az-serialnumber`, for example `myprodappdb-a-01`. These pieces uniquely identify the DB instance and its placement.

- Set the backup retention high for the Aurora cluster. With a long retention period, you can do point-in-time recovery (PITR) for a period up to 35 days. You can reset your database to a known state after running tests involving DDL and data manipulation language (DML) statements. You can also recover if you delete or change data by mistake.
- Turn on additional recovery, logging, and monitoring features at cluster creation. Turn on all the choices under **Backtrack**, **Performance Insights**, **Monitoring**, and **Log exports**. With these features enabled, you can test the suitability of features such as backtracking, Enhanced Monitoring, or Performance Insights for your workload. You can also easily investigate performance and perform troubleshooting during the proof of concept.

5. Set up your schema

On the Aurora cluster, set up databases, tables, indexes, foreign keys, and other schema objects for your application. If you're moving from another MySQL-compatible or PostgreSQL-compatible database system, expect this stage to be simple and straightforward. You use the same SQL syntax and command line or other client applications that you're familiar with for your database engine.

To issue SQL statements on your cluster, find its cluster endpoint and supply that value as the connection parameter to your client application. You can find the cluster endpoint on the [Connectivity](#) tab of the detail page of your cluster. The cluster endpoint is the one labeled **Writer**. The other endpoint, labeled **Reader**, represents a read-only connection that you can supply to end users who run reports or other read-only queries. For help with any issues around connecting to your cluster, see [Connecting to an Amazon Aurora DB cluster \(p. 281\)](#).

If you're porting your schema and data from a different database system, expect to make some schema changes at this point. These schema changes are to match the SQL syntax and capabilities available in Aurora. You might leave out certain columns, constraints, triggers, or other schema objects at this point. Doing so can be useful particularly if these objects require rework for Aurora compatibility and aren't significant for your objectives with the proof of concept.

If you're migrating from a database system with a different underlying engine than Aurora's, consider using the AWS Schema Conversion Tool (AWS SCT) to simplify the process. For details, see the [AWS Schema Conversion Tool User Guide](#). For general details about migration and porting activities, see the [Migrating Your Databases to Amazon Aurora](#) AWS whitepaper.

During this stage, you can evaluate whether there are inefficiencies in your schema setup, for example in your indexing strategy or other table structures such as partitioned tables. Such inefficiencies can be amplified when you deploy your application on a cluster with multiple DB instances and a heavy workload. Consider whether you can fine-tune such performance aspects now, or during later activities such as a full benchmark test.

6. Import your data

During the proof of concept, you bring across the data, or a representative sample, from your former database system. If practical, set up at least some data in each of your tables. Doing so helps to test compatibility of all data types and schema features. After you have exercised the basic Aurora features, scale up the amount of data. By the time you finish the proof of concept, you should test your ETL tools, queries, and overall workload with a dataset that's big enough to draw accurate conclusions.

You can use several techniques to import either physical or logical backup data to Aurora. For details, see [Migrating data to an Amazon Aurora MySQL DB cluster \(p. 781\)](#) or [Migrating data to Amazon Aurora with PostgreSQL compatibility \(p. 1283\)](#) depending on the database engine you're using in the proof of concept.

Experiment with the ETL tools and technologies that you're considering. See which one best meets your needs. Consider both throughput and flexibility. For example, some ETL tools perform a one-time transfer, and others involve ongoing replication from the old system to Aurora.

If you're migrating from a MySQL-compatible system to Aurora MySQL, you can use the native data transfer tools. The same applies if you're migrating from a PostgreSQL-compatible system to Aurora PostgreSQL. If you're migrating from a database system that uses a different underlying engine than Aurora does, you can experiment with the AWS Database Migration Service (AWS DMS). For details about AWS DMS, see the [AWS Database Migration Service User Guide](#).

For details about migration and porting activities, see the AWS whitepaper [Aurora migration handbook](#).

7. Port your SQL code

Trying out SQL and associated applications requires different levels of effort depending on different cases. In particular, the level of effort depends on whether you move from a MySQL-compatible or PostgreSQL-compatible system or another kind.

- If you're moving from RDS for MySQL or RDS for PostgreSQL, the SQL changes are small enough that you can try the original SQL code with Aurora and manually incorporate needed changes.
- Similarly, if you move from an on-premises database compatible with MySQL or PostgreSQL, you can try the original SQL code and manually incorporate changes.
- If you're coming from a different commercial database, the required SQL changes are more extensive. In this case, consider using the AWS SCT.

During this stage, you can evaluate whether there are inefficiencies in your schema setup, for example in your indexing strategy or other table structures such as partitioned tables. Consider whether you can fine-tune such performance aspects now, or during later activities such as a full benchmark test.

You can verify the database connection logic in your application. To take advantage of Aurora distributed processing, you might need to use separate connections for read and write operations, and use relatively short sessions for query operations. For information about connections, see [9. Connect to Aurora \(p. 1700\)](#).

Consider if you had to make compromises and tradeoffs to work around issues in your production database. Build time into the proof-of-concept schedule to make improvements to your schema design and queries. To judge if you can achieve easy wins in performance, operating cost, and scalability, try the original and modified applications side by side on different Aurora clusters.

For details about migration and porting activities, see the AWS whitepaper [Aurora migration handbook](#).

8. Specify configuration settings

You can also review your database configuration parameters as part of the Aurora proof-of-concept exercise. You might already have MySQL or PostgreSQL configuration settings tuned for performance and scalability in your current environment. The Aurora storage subsystem is adapted and tuned for a distributed cloud-based environment with a high-speed storage subsystem. As a result, many former database engine settings don't apply. We recommend conducting your initial experiments with the default Aurora configuration settings. Reapply settings from your current environment only if you encounter performance and scalability bottlenecks. If you're interested, you can look more deeply into this subject in [Introducing the Aurora storage engine](#) on the AWS Database Blog.

Aurora makes it easy to reuse the optimal configuration settings for a particular application or use case. Instead of editing a separate configuration file for each DB instance, you manage sets of parameters that

you assign to entire clusters or specific DB instances. For example, the time zone setting applies to all DB instances in the cluster, and you can adjust the page cache size setting for each DB instance.

You start with one of the default parameter sets, and apply changes to only the parameters that you need to fine-tune. For details about working with parameter groups, see [Amazon Aurora DB cluster and DB instance parameters \(p. 341\)](#). For the configuration settings that are or aren't applicable to Aurora clusters, see [Aurora MySQL configuration parameters \(p. 1042\)](#) or [Amazon Aurora PostgreSQL parameters \(p. 1547\)](#) depending on your database engine.

9. Connect to Aurora

As you find when doing your initial schema and data setup and running sample queries, you can connect to different endpoints in an Aurora cluster. The endpoint to use depends on whether the operation is a read such as `SELECT` statement, or a write such as a `CREATE` or `INSERT` statement. As you increase the workload on an Aurora cluster and experiment with Aurora features, it's important for your application to assign each operation to the appropriate endpoint.

By using the cluster endpoint for write operations, you always connect to a DB instance in the cluster that has read/write capability. By default, only one DB instance in an Aurora cluster has read/write capability. This DB instance is called the *primary instance*. If the original primary instance becomes unavailable, Aurora activates a failover mechanism and a different DB instance takes over as the primary.

Similarly, by directing `SELECT` statements to the reader endpoint, you spread the work of processing queries among the DB instances in the cluster. Each reader connection is assigned to a different DB instance using round-robin DNS resolution. Doing most of the query work on the read-only DB Aurora Replicas reduces the load on the primary instance, freeing it to handle DDL and DML statements.

Using these endpoints reduces the dependency on hard-coded hostnames, and helps your application to recover more quickly from DB instance failures.

Note

Aurora also has custom endpoints that you create. Those endpoints usually aren't needed during a proof of concept.

The Aurora Replicas are subject to a replica lag, even though that lag is usually 10 to 20 milliseconds. You can monitor the replication lag and decide whether it is within the range of your data consistency requirements. In some cases, your read queries might require strong read consistency (read-after-write consistency). In these cases, you can continue using the cluster endpoint for them and not the reader endpoint.

To take full advantage of Aurora capabilities for distributed parallel execution, you might need to change the connection logic. Your objective is to avoid sending all read requests to the primary instance. The read-only Aurora Replicas are standing by, with all the same data, ready to handle `SELECT` statements. Code your application logic to use the appropriate endpoint for each kind of operation. Follow these general guidelines:

- Avoid using a single hard-coded connection string for all database sessions.
- If practical, enclose write operations such as DDL and DML statements in functions in your client application code. That way, you can make different kinds of operations use specific connections.
- Make separate functions for query operations. Aurora assigns each new connection to the reader endpoint to a different Aurora Replica to balance the load for read-intensive applications.
- For operations involving sets of queries, close and reopen the connection to the reader endpoint when each set of related queries is finished. Use connection pooling if that feature is available in your software stack. Directing queries to different connections helps Aurora to distribute the read workload among the DB instances in the cluster.

For general information about connection management and endpoints for Aurora, see [Connecting to an Amazon Aurora DB cluster \(p. 281\)](#). For a deep dive on this subject, see [Aurora MySQL database administrator's handbook – Connection management](#).

10. Run your workload

After the schema, data, and configuration settings are in place, you can begin exercising the cluster by running your workload. Use a workload in the proof of concept that mirrors the main aspects of your production workload. We recommend always making decisions about performance using real-world tests and workloads rather than synthetic benchmarks such as sysbench or TPC-C. Wherever practical, gather measurements based on your own schema, query patterns, and usage volume.

As much as practical, replicate the actual conditions under which the application will run. For example, you typically run your application code on Amazon EC2 instances in the same AWS Region and the same virtual private cloud (VPC) as the Aurora cluster. If your production application runs on multiple EC2 instances spanning multiple Availability Zones, set up your proof-of-concept environment in the same way. For more information on AWS Regions, see [Regions and Availability Zones](#) in the *Amazon RDS User Guide*. To learn more about the Amazon VPC service, see [What is Amazon VPC?](#) in the *Amazon VPC User Guide*.

After you've verified that the basic features of your application work and you can access the data through Aurora, you can exercise aspects of the Aurora cluster. Some features you might want to try are concurrent connections with load balancing, concurrent transactions, and automatic replication.

By this point, the data transfer mechanisms should be familiar, and so you can run tests with a larger proportion of sample data.

This stage is when you can see the effects of changing configuration settings such as memory limits and connection limits. Revisit the procedures that you explored in [8. Specify configuration settings \(p. 1699\)](#).

You can also experiment with mechanisms such as creating and restoring snapshots. For example, you can create clusters with different AWS instance classes, numbers of AWS Replicas, and so on. Then on each cluster, you can restore the same snapshot containing your schema and all your data. For the details of that cycle, see [Creating a DB cluster snapshot \(p. 495\)](#) and [Restoring from a DB cluster snapshot \(p. 497\)](#).

11. Measure performance

Best practices in this area are designed to ensure that all the right tools and processes are set up to quickly isolate abnormal behaviors during workload operations. They're also set up to see that you can reliably identify any applicable causes.

You can always see the current state of your cluster, or examine trends over time, by examining the **Monitoring** tab. This tab is available from the console detail page for each Aurora cluster or DB instance. It displays metrics from the Amazon CloudWatch monitoring service in the form of charts. You can filter the metrics by name, by DB instance, and by time period.

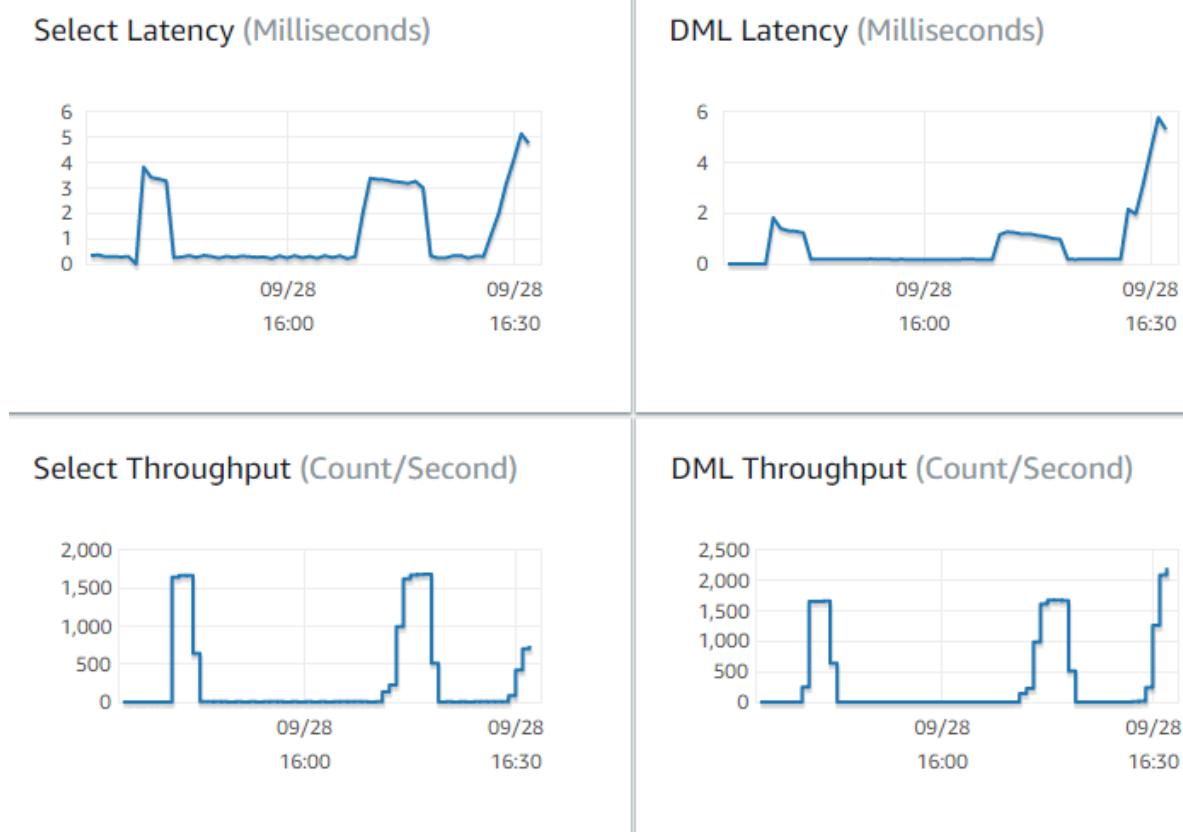
To have more choices on the **Monitoring** tab, enable Enhanced Monitoring and Performance Insights in the cluster settings. You can also enable those choices later if you didn't choose them when setting up the cluster.

To measure performance, you rely mostly on the charts showing activity for the whole Aurora cluster. You can verify whether the Aurora Replicas have similar load and response times. You can also see how the work is split up between the read/write primary instance and the read-only Aurora Replicas. If there is some imbalance between the DB instances or an issue affecting only one DB instance, you can examine the **Monitoring** tab for that specific instance.

After the environment and the actual workload are set up to emulate your production application, you can measure how well Aurora performs. The most important questions to answer are as follows:

- How many queries per second is Aurora processing? You can examine the **Throughput** metrics to see the figures for various kinds of operations.
- How long does it take, on average for Aurora to process a given query? You can examine the **Latency** metrics to see the figures for various kinds of operations.

To do so, look at the **Monitoring** tab for a given Aurora cluster in the [Amazon RDS console](#) as illustrated following.



If you can, establish baseline values for these metrics in your current environment. If that's not practical, construct a baseline on the Aurora cluster by executing a workload equivalent to your production application. For example, run your Aurora workload with a similar number of simultaneous users and queries. Then observe how the values change as you experiment with different instance classes, cluster size, configuration settings, and so on.

If the throughput numbers are lower than you expect, investigate further the factors affecting database performance for your workload. Similarly, if the latency numbers are higher than you expect, further investigate. To do so, monitor the secondary metrics for the DB server (CPU, memory, and so on). You can see whether the DB instances are close to their limits. You can also see how much extra capacity your DB instances have to handle more concurrent queries, queries against larger tables, and so on.

Tip

To detect metric values that fall outside the expected ranges, set up CloudWatch alarms.

When evaluating the ideal Aurora cluster size and capacity, you can find the configuration that achieves peak application performance without over-provisioning resources. One important factor is finding

the appropriate size for the DB instances in the Aurora cluster. Start by selecting an instance size that has similar CPU and memory capacity to your current production environment. Collect throughput and latency numbers for the workload at that instance size. Then, scale the instance up to the next larger size. See if the throughput and latency numbers improve. Also scale the instance size down, and see if the latency and throughput numbers remain the same. Your goal is to get the highest throughput, with the lowest latency, on the smallest instance possible.

Tip

Size your Aurora clusters and associated DB instances with enough existing capacity to handle sudden, unpredictable traffic spikes. For mission-critical databases, leave at least 20 percent spare CPU and memory capacity.

Run performance tests long enough to measure database performance in a warm, steady state. You might need to run the workload for many minutes or even a few hours before reaching this steady state. It's normal at the beginning of a run to have some variance. This variance happens because each Aurora Replica warms up its caches based on the `SELECT` queries that it handles.

Aurora performs best with transactional workloads involving multiple concurrent users and queries. To ensure that you're driving enough load for optimal performance, run benchmarks that use multithreading, or run multiple instances of the performance tests concurrently. Measure performance with hundreds or even thousands of concurrent client threads. Simulate the number of concurrent threads that you expect in your production environment. You might also perform additional stress tests with more threads to measure Aurora scalability.

12. Exercise Aurora high availability

Many of the main Aurora features involve high availability. These features include automatic replication, automatic failover, automatic backups with point-in-time restore, and ability to add DB instances to the cluster. The safety and reliability from features like these are important for mission-critical applications.

To evaluate these features requires a certain mindset. In earlier activities, such as performance measurement, you observe how the system performs when everything works correctly. Testing high availability requires you to think through worst-case behavior. You must consider various kinds of failures, even if such conditions are rare. You might intentionally introduce problems to make sure that the system recovers correctly and quickly.

Tip

For a proof of concept, set up all the DB instances in an Aurora cluster with the same AWS instance class. Doing so makes it possible to try out Aurora availability features without major changes to performance and scalability as you take DB instances offline to simulate failures.

We recommend using at least two instances in each Aurora cluster. The DB instances in an Aurora cluster can span up to three Availability Zones (AZs). Locate each of the first two or three DB instances in a different AZ. When you begin using larger clusters, spread your DB instances across all of the AZs in your AWS Region. Doing so increases fault tolerance capability. Even if a problem affects an entire AZ, Aurora can fail over to a DB instance in a different AZ. If you run a cluster with more than three instances, distribute the DB instances as evenly as you can over all three AZs.

Tip

The storage for an Aurora cluster is independent from the DB instances. The storage for each Aurora cluster always spans three AZs.

When you test high availability features, always use DB instances with identical capacity in your test cluster. Doing so avoids unpredictable changes in performance, latency, and so on whenever one DB instance takes over for another.

To learn how to simulate failure conditions to test high availability features, see [Testing Amazon Aurora using fault injection queries \(p. 829\)](#).

As part of your proof-of-concept exercise, one objective is to find the ideal number of DB instances and the optimal instance class for those DB instances. Doing so requires balancing the requirements of high availability and performance.

For Aurora, the more DB instances that you have in a cluster, the greater the benefits for high availability. Having more DB instances also improves scalability of read-intensive applications. Aurora can distribute multiple connections for `SELECT` queries among the read-only Aurora Replicas.

On the other hand, limiting the number of DB instances reduces the replication traffic from the primary node. The replication traffic consumes network bandwidth, which is another aspect of overall performance and scalability. Thus, for write-intensive OLTP applications, prefer to have a smaller number of large DB instances rather than many small DB instances.

In a typical Aurora cluster, one DB instance (the primary instance) handles all the DDL and DML statements. The other DB instances (the Aurora Replicas) handle only `SELECT` statements. Although the DB instances don't do exactly the same amount of work, we recommend using the same instance class for all the DB instances in the cluster. That way, if a failure happens and Aurora promotes one of the read-only DB instances to be the new primary instance, the primary instance has the same capacity as before.

If you need to use DB instances of different capacities in the same cluster, set up failover tiers for the DB instances. These tiers determine the order in which Aurora Replicas are promoted by the failover mechanism. Put DB instances that are a lot larger or smaller than the others into a lower failover tier. Doing so ensures that they are chosen last for promotion.

Exercise the data recovery features of Aurora, such as automatic point-in-time restore, manual snapshots and restore, and cluster backtracking. If appropriate, copy snapshots to other AWS Regions and restore into other AWS Regions to mimic DR scenarios.

Investigate your organization's requirements for restore time objective (RTO), restore point objective (RPO), and geographic redundancy. Most organizations group these items under the broad category of disaster recovery. Evaluate the Aurora high availability features described in this section in the context of your disaster recovery process to ensure that your RTO and RPO requirements are met.

13. What to do next

At the end of a successful proof-of-concept process, you confirm that Aurora is a suitable solution for you based on the anticipated workload. Throughout the preceding process, you've checked how Aurora works in a realistic operational environment and measured it against your success criteria.

After you get your database environment up and running with Aurora, you can move on to more detailed evaluation steps, leading to your final migration and production deployment. Depending on your situation, these other steps might or might not be included in the proof-of-concept process. For details about migration and porting activities, see the AWS whitepaper [Aurora migration handbook](#).

In another next step, consider the security configurations relevant for your workload and designed to meet your security requirements in a production environment. Plan what controls to put in place to protect access to the Aurora cluster master user credentials. Define the roles and responsibilities of database users to control access to data stored in the Aurora cluster. Take into account database access requirements for applications, scripts, and third-party tools or services. Explore AWS services and features such as AWS Secrets Manager and AWS Identity and Access Management (IAM) authentication.

At this point, you should understand the procedures and best practices for running benchmark tests with Aurora. You might find you need to do additional performance tuning. For details, see [Managing performance and scaling for Aurora DB clusters \(p. 396\)](#), [Amazon Aurora MySQL performance enhancements \(p. 746\)](#), [Managing Amazon Aurora PostgreSQL \(p. 1360\)](#), and [Monitoring DB load with Performance Insights on Amazon Aurora \(p. 573\)](#). If you do additional tuning, make sure that you're

familiar with the metrics that you gathered during the proof of concept. For a next step, you might create new clusters with different choices for configuration settings, database engine, and database version. Or you might create specialized kinds of Aurora clusters to match the needs of specific use cases.

For example, you can explore Aurora parallel query clusters for hybrid transaction/analytical processing (HTAP) applications. If wide geographic distribution is crucial for disaster recovery or to minimize latency, you can explore Aurora global databases. If your workload is intermittent or you're using Aurora in a development/test scenario, you can explore Aurora Serverless clusters.

Your production clusters might also need to handle high volumes of incoming connections. To learn those techniques, see the AWS whitepaper [Aurora MySQL database administrator's handbook – Connection management](#).

If, after the proof of concept, you decide that your use case is not suited for Aurora, consider these other AWS services:

- For purely analytic use cases, workloads benefit from a columnar storage format and other features more suitable to OLAP workloads. AWS services that address such use cases include the following:
 - [Amazon Redshift](#)
 - [Amazon EMR](#)
 - [Amazon Athena](#)
- Many workloads benefit from a combination of Aurora with one or more of these services. You can move data between these services by using these:
 - [AWS Glue](#)
 - [AWS DMS](#)
 - [Importing from Amazon S3](#), as described in the *Amazon Aurora User Guide*
 - [Exporting to Amazon S3](#), as described in the *Amazon Aurora User Guide*
 - Many other popular ETL tools

Security in Amazon Aurora

Cloud security at AWS is the highest priority. As an AWS customer, you benefit from a data center and network architecture that are built to meet the requirements of the most security-sensitive organizations.

Security is a shared responsibility between AWS and you. The [shared responsibility model](#) describes this as security of the cloud and security *in the cloud*:

- **Security of the cloud** – AWS is responsible for protecting the infrastructure that runs AWS services in the AWS Cloud. AWS also provides you with services that you can use securely. Third-party auditors regularly test and verify the effectiveness of our security as part of the [AWS compliance programs](#). To learn about the compliance programs that apply to Amazon Aurora (Aurora), see [AWS services in scope by compliance program](#).
- **Security in the cloud** – Your responsibility is determined by the AWS service that you use. You are also responsible for other factors including the sensitivity of your data, your organization's requirements, and applicable laws and regulations.

This documentation helps you understand how to apply the shared responsibility model when using Amazon Aurora. The following topics show you how to configure Amazon Aurora to meet your security and compliance objectives. You also learn how to use other AWS services that help you monitor and secure your Amazon Aurora resources.

You can manage access to your Amazon Aurora resources and your databases on a DB cluster. The method you use to manage access depends on what type of task the user needs to perform with Amazon Aurora:

- Run your DB cluster in a virtual private cloud (VPC) based on the Amazon VPC service for the greatest possible network access control. For more information about creating a DB cluster in a VPC, see [Amazon Virtual Private Cloud VPCs and Amazon Aurora \(p. 1787\)](#).
- Use AWS Identity and Access Management (IAM) policies to assign permissions that determine who is allowed to manage Amazon Aurora resources. For example, you can use IAM to determine who is allowed to create, describe, modify, and delete DB clusters, tag resources, or modify security groups.

For information on setting up an IAM user, see [Create an IAM user \(p. 84\)](#).

- Use security groups to control what IP addresses or Amazon EC2 instances can connect to your databases on a DB cluster. When you first create a DB cluster, its firewall prevents any database access except through rules specified by an associated security group.
- Use Secure Socket Layer (SSL) or Transport Layer Security (TLS) connections with DB clusters running the Aurora MySQL or Aurora PostgreSQL. For more information on using SSL/TLS with a DB cluster, see [Using SSL/TLS to encrypt a connection to a DB cluster \(p. 1714\)](#).
- Use Amazon Aurora encryption to secure your DB clusters and snapshots at rest. Amazon Aurora encryption uses the industry standard AES-256 encryption algorithm to encrypt your data on the server that hosts your DB cluster. For more information, see [Encrypting Amazon Aurora resources \(p. 1709\)](#).
- Use the security features of your DB engine to control who can log in to the databases on a DB cluster. These features work just as if the database was on your local network.

For information about security with Aurora MySQL, see [Security with Amazon Aurora MySQL \(p. 774\)](#).

For information about security with Aurora PostgreSQL, see [Security with Amazon Aurora PostgreSQL \(p. 1276\)](#).

Aurora is part of the managed database service Amazon Relational Database Service (Amazon RDS). Amazon RDS is a web service that makes it easier to set up, operate, and scale a relational database in the cloud. If you are not already familiar with Amazon RDS, see the [Amazon RDS user guide](#).

Aurora includes a high-performance storage subsystem. Its MySQL- and PostgreSQL-compatible database engines are customized to take advantage of that fast distributed storage. Aurora also automates and standardizes database clustering and replication, which are typically among the most challenging aspects of database configuration and administration.

For both Amazon RDS and Aurora, you can access the RDS API programmatically, and you can use the AWS CLI to access the RDS API interactively. Some RDS API operations and AWS CLI commands apply to both Amazon RDS and Aurora, while others apply to either Amazon RDS or Aurora. For information about RDS API operations, see [Amazon RDS API reference](#). For more information about the AWS CLI, see [AWS Command Line Interface reference for Amazon RDS](#).

Note

You only have to configure security for your use cases. You don't have to configure security access for processes that Amazon Aurora manages. These include creating backups, replicating data between a primary DB instance and a read replica, and other processes.

For more information on managing access to Amazon Aurora resources and your databases on a DB cluster, see the following topics.

Topics

- [Database authentication with Amazon Aurora \(p. 1707\)](#)
- [Data protection in Amazon RDS \(p. 1709\)](#)
- [Identity and access management in Amazon Aurora \(p. 1724\)](#)
- [Logging and monitoring in Amazon Aurora \(p. 1771\)](#)
- [Compliance validation for Amazon Aurora \(p. 1773\)](#)
- [Resilience in Amazon Aurora \(p. 1774\)](#)
- [Infrastructure security in Amazon Aurora \(p. 1776\)](#)
- [Amazon RDS API and interface VPC endpoints \(AWS PrivateLink\) \(p. 1777\)](#)
- [Security best practices for Amazon Aurora \(p. 1779\)](#)
- [Controlling access with security groups \(p. 1780\)](#)
- [Master user account privileges \(p. 1782\)](#)
- [Using service-linked roles for Amazon Aurora \(p. 1783\)](#)
- [Amazon Virtual Private Cloud VPCs and Amazon Aurora \(p. 1787\)](#)

Database authentication with Amazon Aurora

Amazon Aurora supports several ways to authenticate database users.

Password authentication is available by default for all DB clusters. For Aurora MySQL, you can also add IAM database authentication. For Aurora PostgreSQL, you can also add either or both IAM database authentication and Kerberos authentication for the same DB cluster.

Password, Kerberos, and IAM database authentication use different methods of authenticating to the database. Therefore, a specific user can log in to a database using only one authentication method.

For PostgreSQL, use only one of the following role settings for a user of a specific database:

- To use IAM database authentication, assign the `rds_iam` role to the user.
- To use Kerberos authentication, assign the `rds_ad` role to the user.
- To use password authentication, don't assign either the `rds_iam` or `rds_ad` roles to the user.

Don't assign both the `rds_iam` and `rds_ad` roles to a user of a PostgreSQL database either directly or indirectly by nested grant access. If the `rds_iam` role is added to the master user, IAM authentication takes precedence over password authentication so the master user has to log in as an IAM user.

Topics

- [Password authentication \(p. 1708\)](#)
- [IAM database authentication \(p. 1708\)](#)
- [Kerberos authentication \(p. 1708\)](#)

Password authentication

With *password authentication*, your DB instance performs all administration of user accounts. You create users with SQL statements such as `CREATE USER`, with the appropriate clause required by the DB engine for specifying passwords. For example, in MySQL the statement is `CREATE USER name IDENTIFIED BY password`, while in PostgreSQL, the statement is `CREATE USER name WITH PASSWORD password`.

With password authentication, your database controls and authenticates user accounts. If a DB engine has strong password management features, they can enhance security. Database authentication might be easier to administer using password authentication when you have small user communities. Because clear text passwords are generated in this case, integrating with AWS Secrets Manager can enhance security.

For information about using Secrets Manager with Amazon Aurora, see [Creating a basic secret](#) and [Rotating secrets for supported Amazon RDS databases](#) in the *AWS Secrets Manager User Guide*. For information about programmatically retrieving your secrets in your custom applications, see [Retrieving the secret value](#) in the *AWS Secrets Manager User Guide*.

IAM database authentication

You can authenticate to your DB cluster using AWS Identity and Access Management (IAM) database authentication. IAM database authentication works with Aurora MySQL and Aurora PostgreSQL. With this authentication method, you don't need to use a password when you connect to a DB cluster. Instead, you use an authentication token.

For more information about IAM database authentication, including information about availability for specific DB engines, see [IAM database authentication \(p. 1743\)](#).

Kerberos authentication

Amazon Aurora supports external authentication of database users using Kerberos and Microsoft Active Directory. Kerberos is a network authentication protocol that uses tickets and symmetric-key cryptography to eliminate the need to transmit passwords over the network. Kerberos has been built into Active Directory and is designed to authenticate users to network resources, such as databases.

Amazon Aurora support for Kerberos and Active Directory provides the benefits of single sign-on and centralized authentication of database users. You can keep your user credentials in Active Directory. Active Directory provides a centralized place for storing and managing credentials for multiple DB instances.

You can enable your database users to authenticate against DB instances in two ways. They can use credentials stored either in AWS Directory Service for Microsoft Active Directory or in your on-premises Active Directory.

Currently, Aurora supports Kerberos authentication for Aurora PostgreSQL DB clusters. With Kerberos authentication, Aurora PostgreSQL DB clusters support one- and two-way forest trust relationships. For more information, see [Using Kerberos authentication with Aurora PostgreSQL \(p. 1534\)](#).

Data protection in Amazon RDS

The AWS [shared responsibility model](#) applies to data protection in Amazon Relational Database Service. As described in this model, AWS is responsible for protecting the global infrastructure that runs all of the AWS Cloud. You are responsible for maintaining control over your content that is hosted on this infrastructure. This content includes the security configuration and management tasks for the AWS services that you use. For more information about data privacy, see the [Data Privacy FAQ](#). For information about data protection in Europe, see the [AWS Shared Responsibility Model and GDPR](#) blog post on the [AWS Security Blog](#).

For data protection purposes, we recommend that you protect AWS account credentials and set up individual user accounts with AWS Identity and Access Management (IAM). That way each user is given only the permissions necessary to fulfill their job duties. We also recommend that you secure your data in the following ways:

- Use multi-factor authentication (MFA) with each account.
- Use SSL/TLS to communicate with AWS resources. We recommend TLS 1.2 or later.
- Set up API and user activity logging with AWS CloudTrail.
- Use AWS encryption solutions, along with all default security controls within AWS services.
- Use advanced managed security services such as Amazon Macie, which assists in discovering and securing personal data that is stored in Amazon S3.
- If you require FIPS 140-2 validated cryptographic modules when accessing AWS through a command line interface or an API, use a FIPS endpoint. For more information about the available FIPS endpoints, see [Federal Information Processing Standard \(FIPS\) 140-2](#).

We strongly recommend that you never put confidential or sensitive information, such as your customers' email addresses, into tags or free-form fields such as a **Name** field. This includes when you work with Amazon RDS or other AWS services using the console, API, AWS CLI, or AWS SDKs. Any data that you enter into tags or free-form fields used for names may be used for billing or diagnostic logs. If you provide a URL to an external server, we strongly recommend that you do not include credentials information in the URL to validate your request to that server.

Topics

- [Protecting data using encryption \(p. 1709\)](#)
- [Internetwork traffic privacy \(p. 1723\)](#)

Protecting data using encryption

You can enable encryption for database resources. You can also encrypt connections to DB clusters.

Topics

- [Encrypting Amazon Aurora resources \(p. 1709\)](#)
- [AWS KMS key management \(p. 1713\)](#)
- [Using SSL/TLS to encrypt a connection to a DB cluster \(p. 1714\)](#)
- [Rotating your SSL/TLS certificate \(p. 1715\)](#)

Encrypting Amazon Aurora resources

Amazon Aurora can encrypt your Amazon Aurora DB clusters. Data that is encrypted at rest includes the underlying storage for DB clusters, its automated backups, read replicas, and snapshots.

Amazon Aurora encrypted DB clusters use the industry standard AES-256 encryption algorithm to encrypt your data on the server that hosts your Amazon Aurora DB clusters. After your data is encrypted, Amazon Aurora handles authentication of access and decryption of your data transparently with a minimal impact on performance. You don't need to modify your database client applications to use encryption.

Note

For encrypted and unencrypted DB clusters, data that is in transit between the source and the read replicas is encrypted, even when replicating across AWS Regions.

Topics

- [Overview of encrypting Amazon Aurora resources \(p. 1710\)](#)
- [Enabling encryption for an Amazon Aurora DB cluster \(p. 1710\)](#)
- [Determining whether encryption is turned on for a DB cluster \(p. 1711\)](#)
- [Availability of Amazon Aurora encryption \(p. 1712\)](#)
- [Limitations of Amazon Aurora encrypted DB clusters \(p. 1712\)](#)

Overview of encrypting Amazon Aurora resources

Amazon Aurora encrypted DB clusters provide an additional layer of data protection by securing your data from unauthorized access to the underlying storage. You can use Amazon Aurora encryption to increase data protection of your applications deployed in the cloud, and to fulfill compliance requirements for encryption at rest.

For an Amazon Aurora encrypted DB cluster, all DB instances, logs, backups, and snapshots are encrypted. You can also encrypt a read replica of an Amazon Aurora encrypted cluster. Amazon Aurora uses an AWS KMS key to encrypt these resources. For more information about KMS keys, see [AWS KMS keys](#) in the *AWS Key Management Service Developer Guide*. Each DB instance in the DB cluster is encrypted using the same KMS key as the DB cluster. If you copy an encrypted snapshot, you can use a different KMS key to encrypt the target snapshot than the one that was used to encrypt the source snapshot.

You can use an AWS managed key, or you can create customer managed keys. To manage the customer managed keys used for encrypting and decrypting your Amazon Aurora resources, you use the [AWS Key Management Service \(AWS KMS\)](#). AWS KMS combines secure, highly available hardware and software to provide a key management system scaled for the cloud. Using AWS KMS, you can create customer managed keys and define the policies that control how these customer managed keys can be used. AWS KMS supports CloudTrail, so you can audit KMS key usage to verify that customer managed keys are being used appropriately. You can use your customer managed keys with Amazon Aurora and supported AWS services such as Amazon S3, Amazon EBS, and Amazon Redshift. For a list of services that are integrated with AWS KMS, see [AWS Service Integration](#).

Enabling encryption for an Amazon Aurora DB cluster

To enable encryption for a new DB cluster, choose **Enable encryption** on the console. For information on creating a DB cluster, see [Creating an Amazon Aurora DB cluster \(p. 125\)](#).

If you use the `create-db-cluster` AWS CLI command to create an encrypted DB cluster, set the `--storage-encrypted` parameter. If you use the `CreateDBCluster` API operation, set the `StorageEncrypted` parameter to true.

When you create an encrypted DB cluster, you can choose a customer managed key or the AWS managed key for Amazon Aurora to encrypt your DB cluster. If you don't specify the key identifier for a customer managed key, Amazon Aurora uses the AWS managed key for your new DB cluster. Amazon Aurora creates an AWS managed key for Amazon Aurora for your AWS account. Your AWS account has a different AWS managed key for Amazon Aurora for each AWS Region.

Once you have created an encrypted DB cluster, you can't change the KMS key used by that DB cluster. Therefore, be sure to determine your KMS key requirements before you create your encrypted DB cluster.

If you use the AWS CLI `create-db-cluster` command to create an encrypted DB cluster with a customer managed key, set the `--kms-key-id` parameter to any key identifier for the KMS key. If you use the Amazon RDS API `CreateDBInstance` operation, set the `KmsKeyId` parameter to any key identifier for the KMS key. To use a customer managed key in a different AWS account, specify the key ARN or alias ARN.

Important

Amazon Aurora can lose access to the KMS key for a DB cluster. For example, RDS loses access when the KMS key is disabled, or when RDS access to a KMS key is revoked. In these cases, the encrypted DB cluster goes into `inaccessible-encryption-credentials-recoverable` state. The DB cluster remains in this state for seven days. When you start the DB cluster during that time, it checks if the KMS key is active and recovers the KMS key if it is.

If the KMS key isn't recovered, then the encrypted DB cluster goes into the terminal `inaccessible-encryption-credentials` state. In this case, you can only restore the DB cluster from a backup. We strongly recommend that you always enable backups for encrypted DB instances to guard against the loss of encrypted data in your databases.

Determining whether encryption is turned on for a DB cluster

You can use the AWS Management Console, AWS CLI, or RDS API to determine whether encryption at rest is turned on for a DB cluster.

Console

To determine whether encryption at rest is turned on for a DB cluster

1. Sign in to the AWS Management Console and open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. In the navigation pane, choose **Databases**.
3. Choose the name of the DB cluster that you want to check to view its details.
4. Choose the **Configuration** tab and check the **Encryption** value.

It shows either **Enabled** or **Not enabled**.

The screenshot shows the AWS RDS console interface. At the top, the URL is `RDS > Databases > aurora-cl-mysql`. Below the title, there's a 'Modify' button and an 'Actions' dropdown. A 'Related' section lists other database instances: 'aurora-cl-mysql' (Regional cluster, Aurora MySQL, us-east-1, 2 instances, Enabled), 'dbinstance4' (Writer instance, Aurora MySQL, us-east-1a, db.t3.medium, Enabled), and 'dbinstance1' (Reader instance, Aurora MySQL, us-east-1b, db.t3.medium, Enabled). The main area shows the DB cluster details under the 'Configuration' tab. The 'Encryption' section is highlighted with a red box and contains the text 'Encryption' and 'Enabled'.

DB identifier	Role	Engine	Region & AZ	Size	Status
aurora-cl-mysql	Regional cluster	Aurora MySQL	us-east-1	2 instances	Enabled
dbinstance4	Writer instance	Aurora MySQL	us-east-1a	db.t3.medium	Enabled
dbinstance1	Reader instance	Aurora MySQL	us-east-1b	db.t3.medium	Enabled

Configuration	Capacity type	Availability	Encryption
DB cluster role Regional cluster	Provisioned: single-master DB cluster ID aurora-cl-mysql	IAM DB authentication Enabled	Encryption Enabled

AWS CLI

To determine whether encryption at rest is turned on for a DB cluster by using the AWS CLI, call the [describe-db-clusters](#) command with the following option:

- `--db-cluster-identifier` – The name of the DB cluster.

The following example uses a query to return either `TRUE` or `FALSE` regarding encryption at rest for the `mydb` DB cluster.

Example

```
aws rds describe-db-clusters --db-cluster-identifier mydb --query "*[].[StorageEncrypted:StorageEncrypted}" --output text
```

RDS API

To determine whether encryption at rest is turned on for a DB cluster by using the Amazon RDS API, call the [DescribeDBClusters](#) operation with the following parameter:

- `DBClusterIdentifier` – The name of the DB cluster.

Availability of Amazon Aurora encryption

Amazon Aurora encryption is currently available for all database engines and storage types.

Note

Amazon Aurora encryption is not available for the db.t2.micro DB instance class.

Limitations of Amazon Aurora encrypted DB clusters

The following limitations exist for Amazon Aurora encrypted DB clusters:

- You can't disable encryption on an encrypted DB cluster.
- You can't create an encrypted snapshot of an unencrypted DB cluster.
- A snapshot of an encrypted DB cluster must be encrypted using the same KMS key as the DB cluster.
- You can't convert an unencrypted DB cluster to an encrypted one. However, you can restore an unencrypted snapshot to an encrypted Aurora DB cluster. To do this, specify a KMS key when you restore from the unencrypted snapshot.
- You can't create an encrypted Aurora Replica from an unencrypted Aurora DB cluster. You can't create an unencrypted Aurora Replica from an encrypted Aurora DB cluster.
- To copy an encrypted snapshot from one AWS Region to another, you must specify the KMS key in the destination AWS Region. This is because KMS keys are specific to the AWS Region that they are created in.

The source snapshot remains encrypted throughout the copy process. Amazon Aurora uses envelope encryption to protect data during the copy process. For more information about envelope encryption, see [Envelope encryption](#) in the *AWS Key Management Service Developer Guide*.

- You can't unencrypt an encrypted DB cluster. However, you can export data from an encrypted DB cluster and import the data into an unencrypted DB cluster.

AWS KMS key management

Amazon Aurora automatically integrates with AWS Key Management Service (AWS KMS) for key management. Amazon Aurora uses envelope encryption. For more information about envelope encryption, see [Envelope encryption](#) in the *AWS Key Management Service Developer Guide*.

An AWS KMS key is a logical representation of a key. The KMS key includes metadata, such as the key ID, creation date, description, and key state. The KMS key also contains the key material used to encrypt and decrypt data. For more information about KMS keys, see [AWS KMS keys](#) in the *AWS Key Management Service Developer Guide*.

You can manage KMS keys used for Amazon Aurora encrypted DB clusters using the [AWS Key Management Service \(AWS KMS\)](#) in the [AWS KMS console](#), the AWS CLI, or the AWS KMS API. If you want full control over a KMS key, then you must create a customer managed key. For more information about customer managed keys, see [Customer managed keys](#) in the *AWS Key Management Service Developer Guide*.

AWS managed keys are KMS keys in your account that are created, managed, and used on your behalf by an AWS service that is integrated with AWS KMS. You can't delete, edit, or rotate AWS managed keys. For more information about AWS managed keys, see [AWS managed keys](#) in the *AWS Key Management Service Developer Guide*.

You can't share a snapshot that has been encrypted using the AWS managed key of the AWS account that shared the snapshot.

You can view audit logs of every action taken with an AWS managed or customer managed key by using [AWS CloudTrail](#).

Important

If you disable or revoke permissions to a KMS key used by an RDS database, RDS puts your database into a terminal state when access to the KMS key is required. This change could be immediate, or deferred, depending on the use case that required access to the KMS key. In this state, the DB cluster is no longer available, and the current state of the database can't be recovered. To restore the DB cluster, you must re-enable access to the KMS key for RDS, and then restore the DB cluster from the latest available backup.

Authorizing use of a customer managed key

When Aurora uses a customer managed key in cryptographic operations, it acts on behalf of the user who is creating or changing the Aurora resource.

To use the customer managed key for an Aurora resource on your behalf, a user must have permissions to call the following operations on the customer managed key:

- `kms:GenerateDataKey`
- `kms:Decrypt`

You can specify these required permissions in a key policy, or in an IAM policy if the key policy allows it.

You can make the IAM policy stricter in various ways. For example, to allow the customer managed key to be used only for requests that originate in Aurora, you can use the [kms:ViaService condition key](#) with the `rds.<region>.amazonaws.com` value.

You can also use the keys or values in the [encryption context](#) as a condition for using the customer managed key for cryptographic operations.

For more information, see [Allowing users in other accounts to use a KMS key](#) in the *AWS Key Management Service Developer Guide*.

Using SSL/TLS to encrypt a connection to a DB cluster

You can use Secure Socket Layer (SSL) or Transport Layer Security (TLS) from your application to encrypt a connection to a DB cluster running Aurora MySQL or Aurora PostgreSQL.

SSL/TLS connections provide one layer of security by encrypting data that moves between your client and a DB cluster. Using a server certificate provides an extra layer of security by validating that the connection is being made to an Amazon Aurora DB cluster. It does so by checking the server certificate that is automatically installed on all DB clusters that you provision.

Each DB engine has its own process for implementing SSL/TLS. To learn how to implement SSL/TLS for your DB cluster, use the link following that corresponds to your DB engine:

- [Security with Amazon Aurora MySQL \(p. 774\)](#)
- [Security with Amazon Aurora PostgreSQL \(p. 1276\)](#)

Note

All certificates are only available for download using SSL/TLS connections.

To get a certificate bundle that contains both the intermediate and root certificates for all AWS Regions, download from <https://truststore.pki.rds.amazonaws.com/global/global-bundle.pem>.

If your application is on Microsoft Windows and requires a PKCS7 file, you can download the PKCS7 certificate bundle. This bundle contains both the intermediate and root certificates at <https://truststore.pki.rds.amazonaws.com/global/global-bundle.p7b>.

Note

Amazon RDS Proxy and Aurora Serverless use certificates from the AWS Certificate Manager (ACM). If you are using RDS Proxy, you don't need to download Amazon RDS certificates or update applications that use RDS Proxy connections. For more information about using TLS/SSL with RDS Proxy, see [Using TLS/SSL with RDS Proxy \(p. 292\)](#).

If you are Aurora Serverless, downloading Amazon RDS certificates isn't required. For more information about using TLS/SSL with Aurora Serverless, see [Using TLS/SSL with Aurora Serverless v1 \(p. 150\)](#).

Certificate bundles for AWS Regions

To get a certificate bundle that contains both the intermediate and root certificates for an AWS Region, download from the link for the AWS Region in the following table.

AWS Region	Certificate bundle (PEM)	Certificate bundle (PKCS7)
US East (N. Virginia)	us-east-1-bundle.pem	us-east-1-bundle.p7b
US East (Ohio)	us-east-2-bundle.pem	us-east-2-bundle.p7b
US West (N. California)	us-west-1-bundle.pem	us-west-1-bundle.p7b
US West (Oregon)	us-west-2-bundle.pem	us-west-2-bundle.p7b
Africa (Cape Town)	af-south-1-bundle.pem	af-south-1-bundle.p7b
Asia Pacific (Hong Kong)	ap-east-1-bundle.pem	ap-east-1-bundle.p7b
Asia Pacific (Jakarta)	ap-southeast-3-bundle.pem	ap-southeast-3-bundle.p7b
Asia Pacific (Mumbai)	ap-south-1-bundle.pem	ap-south-1-bundle.p7b

AWS Region	Certificate bundle (PEM)	Certificate bundle (PKCS7)
Asia Pacific (Osaka)	ap-northeast-3-bundle.pem	ap-northeast-3-bundle.p7b
Asia Pacific (Tokyo)	ap-northeast-1-bundle.pem	ap-northeast-1-bundle.p7b
Asia Pacific (Seoul)	ap-northeast-2-bundle.pem	ap-northeast-2-bundle.p7b
Asia Pacific (Singapore)	ap-southeast-1-bundle.pem	ap-southeast-1-bundle.p7b
Asia Pacific (Sydney)	ap-southeast-2-bundle.pem	ap-southeast-2-bundle.p7b
Canada (Central)	ca-central-1-bundle.pem	ca-central-1-bundle.p7b
Europe (Frankfurt)	eu-central-1-bundle.pem	eu-central-1-bundle.p7b
Europe (Ireland)	eu-west-1-bundle.pem	eu-west-1-bundle.p7b
Europe (London)	eu-west-2-bundle.pem	eu-west-2-bundle.p7b
Europe (Milan)	eu-south-1-bundle.pem	eu-south-1-bundle.p7b
Europe (Paris)	eu-west-3-bundle.pem	eu-west-3-bundle.p7b
Europe (Stockholm)	eu-north-1-bundle.pem	eu-north-1-bundle.p7b
Middle East (Bahrain)	me-south-1-bundle.pem	me-south-1-bundle.p7b
South America (São Paulo)	sa-east-1-bundle.pem	sa-east-1-bundle.p7b

AWS GovCloud (US) certificates

To get a certificate bundle that contains both the intermediate and root certificates for the AWS GovCloud (US) Regions, download from <https://truststore.pki.us-gov-west-1.rds.amazonaws.com/global/global-bundle.pem>.

If your application is on Microsoft Windows and requires a PKCS7 file, you can download the PKCS7 certificate bundle. This bundle contains both the intermediate and root certificates at <https://truststore.pki.us-gov-west-1.rds.amazonaws.com/global/global-bundle.p7b>.

To get a certificate bundle that contains both the intermediate and root certificates for an AWS GovCloud (US) Region, download from the link for the AWS GovCloud (US) Region in the following table.

AWS GovCloud (US) Region	Certificate bundle (PEM)	Certificate bundle (PKCS7)
AWS GovCloud (US-East)	us-gov-east-1-bundle.pem	us-gov-east-1-bundle.p7b
AWS GovCloud (US-West)	us-gov-west-1-bundle.pem	us-gov-west-1-bundle.p7b

Rotating your SSL/TLS certificate

As of March 5, 2020, Amazon RDS CA-2015 certificates have expired. If you use or plan to use Secure Sockets Layer (SSL) or Transport Layer Security (TLS) with certificate verification to connect to your RDS DB instances, you require Amazon RDS CA-2019 certificates, which are enabled by default for new DB instances. If you currently do not use SSL/TLS with certificate verification, you might still have expired CA-2015 certificates and must update them to CA-2019 certificates if you plan to use SSL/TLS with certificate verification to connect to your RDS databases.

Follow these instructions to complete your updates. Before you update your DB instances to use the new CA certificate, make sure that you update your clients or applications connecting to your RDS databases.

Amazon RDS provides new CA certificates as an AWS security best practice. For information about the new certificates and the supported AWS Regions, see [Using SSL/TLS to encrypt a connection to a DB cluster \(p. 1714\)](#).

Note

Amazon RDS Proxy and Aurora Serverless use certificates from the AWS Certificate Manager (ACM). If you are using RDS Proxy, when you rotate your SSL/TLS certificate, you don't need to update applications that use RDS Proxy connections. For more information about using TLS/SSL with RDS Proxy, see [Using TLS/SSL with RDS Proxy \(p. 292\)](#).

If you are Aurora Serverless, rotating your SSL/TLS certificate isn't required. For more information about using TLS/SSL with Aurora Serverless, see [Using TLS/SSL with Aurora Serverless v1 \(p. 150\)](#).

Note

If you are using a Go version 1.15 application with a DB instance that was created or updated to the `rds-ca-2019` certificate prior to July 28, 2020, you must update the certificate again. Run the `modify-db-instance` command shown in the AWS CLI section using `rds-ca-2019` as the CA certificate identifier. In this case, it isn't possible to update the certificate using the AWS Management Console. If you created your DB instance or updated its certificate after July 28, 2020, no action is required. For more information, see [Go GitHub issue #39568](#).

Topics

- [Updating your CA certificate by modifying your DB instance \(p. 1716\)](#)
- [Updating your CA certificate by applying DB instance maintenance \(p. 1719\)](#)
- [Sample script for importing certificates into your trust store \(p. 1722\)](#)

Updating your CA certificate by modifying your DB instance

Complete the following steps to update your CA certificate.

To update your CA certificate by modifying your DB instance

1. Download the new SSL/TLS certificate as described in [Using SSL/TLS to encrypt a connection to a DB cluster \(p. 1714\)](#).
2. Update your applications to use the new SSL/TLS certificate.

The methods for updating applications for new SSL/TLS certificates depend on your specific applications. Work with your application developers to update the SSL/TLS certificates for your applications.

For information about checking for SSL/TLS connections and updating applications for each DB engine, see the following topics:

- [Updating applications to connect to Aurora MySQL DB clusters using new SSL/TLS certificates \(p. 778\)](#)
- [Updating applications to connect to Aurora PostgreSQL DB clusters using new SSL/TLS certificates \(p. 1280\)](#)

For a sample script that updates a trust store for a Linux operating system, see [Sample script for importing certificates into your trust store \(p. 1722\)](#).

Note

The certificate bundle contains certificates for both the old and new CA, so you can upgrade your application safely and maintain connectivity during the transition period. If you

are using the AWS Database Migration Service to migrate a database to a DB cluster, we recommend using the certificate bundle to ensure connectivity during the migration.

3. Modify the DB instance to change the CA from **rds-ca-2015** to **rds-ca-2019**.

Important

By default, this operation restarts your DB instance. If you don't want to restart your DB instance during this operation, you can use the `modify-db-instance` CLI command and specify the `--no-certificate-rotation-restart` option.

This option will not rotate the certificate until the next time the database restarts, either for planned or unplanned maintenance. This option is only recommended if you don't use SSL/TLS.

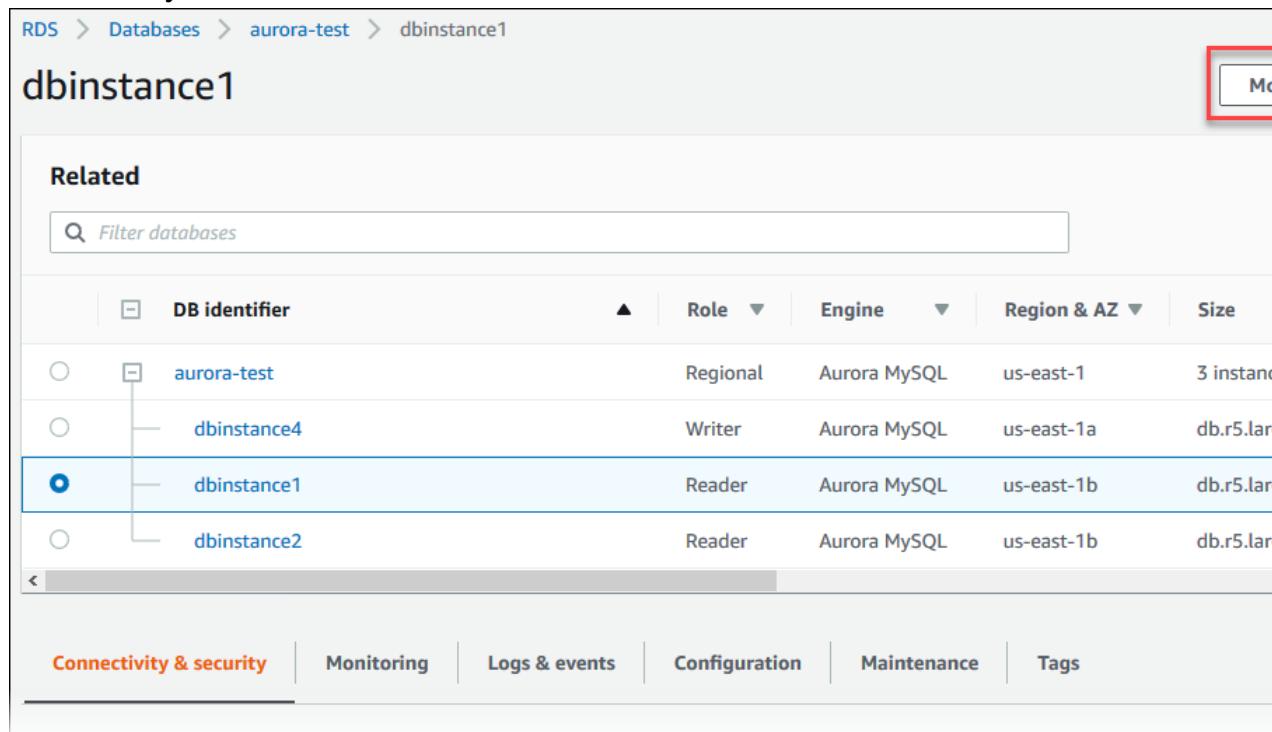
If you are experiencing connectivity issues after certificate expiry, use the `apply immediately` option by specifying **Apply immediately** in the console or by specifying the `--apply-immediately` option using the AWS CLI. By default, this operation is scheduled to run during your next maintenance window.

You can use the AWS Management Console or the AWS CLI to change the CA certificate from **rds-ca-2015** to **rds-ca-2019** for a DB instance.

Console

To change the CA from rds-ca-2015 to rds-ca-2019 for a DB instance

1. Sign in to the AWS Management Console and open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. In the navigation pane, choose **Databases**, and then choose the DB instance that you want to modify.
3. Choose **Modify**.



The **Modify DB Instance** page appears.

4. In the **Connectivity** section, choose **rds-ca-2019**.

5. Choose **Continue** and check the summary of modifications.

6. To apply the changes immediately, choose **Apply immediately**.

Important
Choosing this option restarts your database immediately.

7. On the confirmation page, review your changes. If they are correct, choose **Modify DB Instance** to save your changes.

Important
When you schedule this operation, make sure that you have updated your client-side trust store beforehand.

Or choose **Back** to edit your changes or **Cancel** to cancel your changes.

AWS CLI

To use the AWS CLI to change the CA from **rds-ca-2015** to **rds-ca-2019** for a DB instance, call the [modify-db-instance](#) command. Specify the DB instance identifier and the `--ca-certificate-identifier` option.

Important

When you schedule this operation, make sure that you have updated your client-side trust store beforehand.

Example

The following code modifies `mydbinstance` by setting the CA certificate to `rds-ca-2019`. The changes are applied during the next maintenance window by using `--no-apply-immediately`. Use `--apply-immediately` to apply the changes immediately.

Important

By default, this operation reboots your DB instance. If you don't want to reboot your DB instance during this operation, you can use the `modify-db-instance` CLI command and specify the `--no-certificate-rotation-restart` option.

This option will not rotate the certificate until the next time the database restarts, either for planned or unplanned maintenance. This option is only recommended if you do not use SSL/TLS.

Use `--apply-immediately` to apply the update immediately. By default, this operation is scheduled to run during your next maintenance window.

For Linux, macOS, or Unix:

```
aws rds modify-db-instance \
--db-instance-identifier mydbinstance \
--ca-certificate-identifier rds-ca-2019 \
--no-apply-immediately
```

For Windows:

```
aws rds modify-db-instance ^
--db-instance-identifier mydbinstance ^
--ca-certificate-identifier rds-ca-2019 ^
--no-apply-immediately
```

Updating your CA certificate by applying DB instance maintenance

Complete the following steps to update your CA certificate by applying DB instance maintenance.

To update your CA certificate by applying DB instance maintenance

1. Download the new SSL/TLS certificate as described in [Using SSL/TLS to encrypt a connection to a DB cluster \(p. 1714\)](#).
2. Update your database applications to use the new SSL/TLS certificate.

The methods for updating applications for new SSL/TLS certificates depend on your specific applications. Work with your application developers to update the SSL/TLS certificates for your applications.

For information about checking for SSL/TLS connections and updating applications for each DB engine, see the following topics:

- [Updating applications to connect to Aurora MySQL DB clusters using new SSL/TLS certificates \(p. 778\)](#)
- [Updating applications to connect to Aurora PostgreSQL DB clusters using new SSL/TLS certificates \(p. 1280\)](#)

For a sample script that updates a trust store for a Linux operating system, see [Sample script for importing certificates into your trust store \(p. 1722\)](#).

Note

The certificate bundle contains certificates for both the old and new CA, so you can upgrade your application safely and maintain connectivity during the transition period.

3. Apply DB instance maintenance to change the CA from **rds-ca-2015** to **rds-ca-2019**.

Important

You can choose to apply the change immediately. By default, this operation is scheduled to run during your next maintenance window.

You can use the AWS Management Console to apply DB instance maintenance to change the CA certificate from **rds-ca-2015** to **rds-ca-2019** for multiple DB instances.

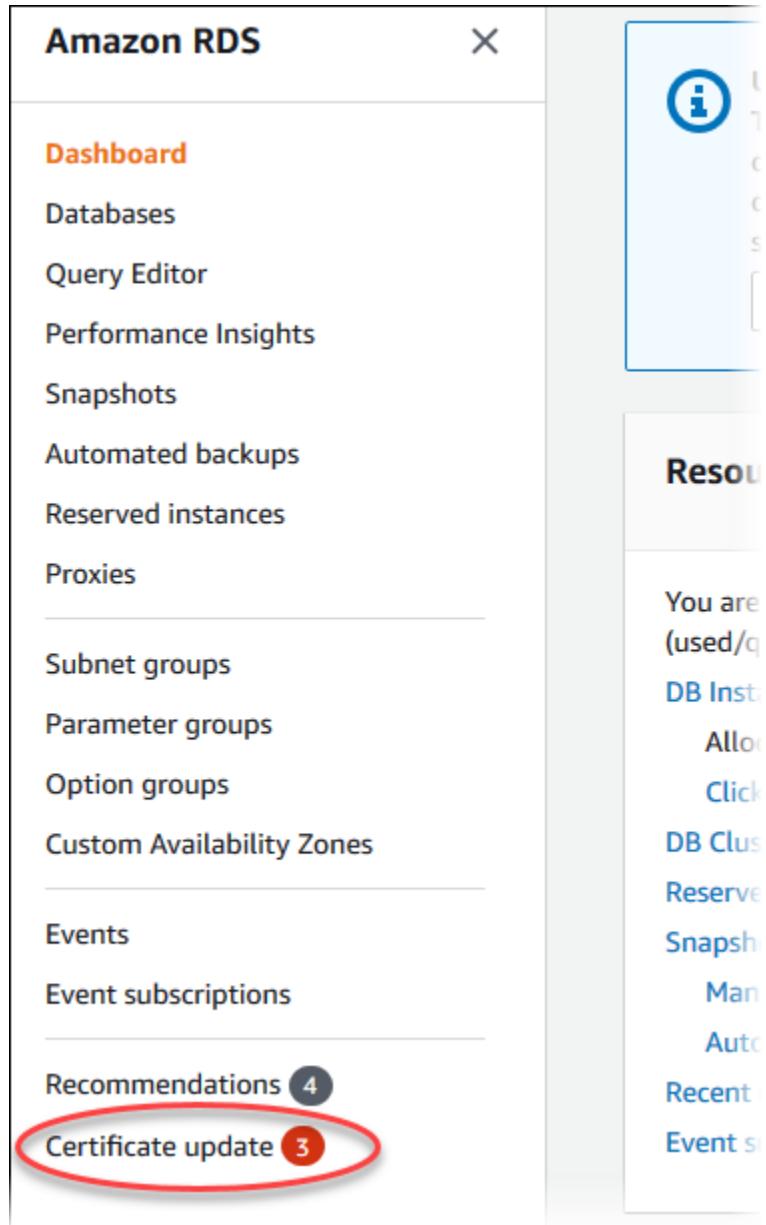
Updating your CA certificate by applying maintenance to multiple DB instances

Use the AWS Management Console to change the CA certificate for multiple DB instances.

To change the CA from rds-ca-2015 to rds-ca-2019 for multiple DB instances

1. Sign in to the AWS Management Console and open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. In the navigation pane, choose **Databases**.

In the navigation pane, there is a **Certificate update** option that shows the total number of affected DB instances.



Choose **Certificate update** in the navigation pane.

The **Update your Amazon RDS SSL/TLS certificates** page appears.

DB identifier	DB cluster identifier	Status	Apply date
mydbinstancecf	-	Requires Update	-
mydbinstancecf2	-	Requires Update	-
oracledb	-	Requires Update	-

Note

This page only shows the DB instances for the current AWS Region. If you have DB instances in more than one AWS Region, check this page in each AWS Region to see all DB instances with old SSL/TLS certificates.

3. Choose the DB instance you want to update.

You can schedule the certificate rotation for your next maintenance window by choosing **Update at the next maintenance window**. Apply the rotation immediately by choosing **Update now**.

Important

When your CA certificate is rotated, the operation restarts your DB instance.

If you experience connectivity issues after certificate expiry, use the **Update now** option.

4. If you choose **Update at the next maintenance window** or **Update now**, you are prompted to confirm the CA certificate rotation.

Important

Before scheduling the CA certificate rotation on your database, update any client applications that use SSL/TLS and the server certificate to connect. These updates are specific to your DB engine. To determine whether your applications use SSL/TLS and the server certificate to connect, see [Step 2: Update Your Database Applications to Use the New SSL/TLS Certificate \(p. 1719\)](#). After you have updated these client applications, you can confirm the CA certificate rotation.

Confirm rotation of CA certificate?

Before scheduling the CA certificate rotation, update client applications that connect to your database to use the new CA certificate. Not doing this will cause an interruption of connectivity between your applications and your database. [Get new CA certificates](#)

I understand that not doing so will break SSL/TLS connectivity to my database.

Cancel

Confirm

To continue, choose the check box, and then choose **Confirm**.

5. Repeat steps 3 and 4 for each DB instance that you want to update.

Sample script for importing certificates into your trust store

The following are sample shell scripts that import the certificate bundle into a trust store.

Each sample shell script uses keytool, which is part of the Java Development Kit (JDK). For information about installing the JDK, see [JDK Installation Guide](#).

Topics

- [Sample script for importing certificates on Linux \(p. 1722\)](#)
- [Sample script for importing certificates on macOS \(p. 1722\)](#)

Sample script for importing certificates on Linux

The following is a sample shell script that imports the certificate bundle into a trust store on a Linux operating system.

```
mydir=tmp/certs
if [ ! -e "${mydir}" ]
then
mkdir -p "${mydir}"
fi

truststore=${mydir}/rds-truststore.jks
storepassword=changeit

curl -sS "https://s3.amazonaws.com/rds-downloads/rds-combined-ca-bundle.pem" > ${mydir}/
rds-combined-ca-bundle.pem
awk 'split_after == 1 {n++;split_after=0} /-----END CERTIFICATE-----/ {split_after=1}{print
> "rds-ca-" n ".pem"}' < ${mydir}/rds-combined-ca-bundle.pem

for CERT in rds-ca-*; do
alias=$(openssl x509 -noout -text -in $CERT | perl -ne 'next unless /Subject:/; s/.*(CN=|CN = )//; print')
echo "Importing $alias"
keytool -import -file ${CERT} -alias "${alias}" -storepass ${storepassword} -keystore
${truststore} -noprompt
rm $CERT
done

rm ${mydir}/rds-combined-ca-bundle.pem

echo "Trust store content is: "

keytool -list -v -keystore "$truststore" -storepass ${storepassword} | grep Alias | cut -d
" " -f3- | while read alias
do
expiry=`keytool -list -v -keystore "$truststore" -storepass ${storepassword} -alias
"${alias}" | grep Valid | perl -ne 'if(/until: (.*)\n/) { print "$1\n"; }'`^
echo " Certificate ${alias} expires in '$expiry'"^
done
```

Sample script for importing certificates on macOS

The following is a sample shell script that imports the certificate bundle into a trust store on macOS.

```
mydir=tmp/certs
if [ ! -e "${mydir}" ]
then
```

```
mkdir -p "${mydir}"
fi

truststore=${mydir}/rds-truststore.jks
storepassword=changeit

curl -sS "https://s3.amazonaws.com/rds-downloads/rds-combined-ca-bundle.pem" > ${mydir}/
rds-combined-ca-bundle.pem
split -p "-----BEGIN CERTIFICATE-----" ${mydir}/rds-combined-ca-bundle.pem rds-ca-
for CERT in rds-ca-*; do
    alias=$(openssl x509 -noout -text -in $CERT | perl -ne 'next unless /Subject:/; s/.*(CN=|CN = )//; print')
    echo "Importing $alias"
    keytool -import -file ${CERT} -alias "${alias}" -storepass ${storepassword} -keystore
${truststore} -noprompt
    rm $CERT
done

rm ${mydir}/rds-combined-ca-bundle.pem

echo "Trust store content is: "

keytool -list -v -keystore "$truststore" -storepass ${storepassword} | grep Alias | cut -d
" " -f3- | while read alias
do
    expiry=`keytool -list -v -keystore "$truststore" -storepass ${storepassword} -alias
"${alias}" | grep Valid | perl -ne 'if(/until: (.*)\n/) { print "$1\n"; }'`^
    echo " Certificate ${alias} expires in '$expiry'"
done
```

Internetwork traffic privacy

Connections are protected both between Amazon Aurora and on-premises applications and between Amazon Aurora and other AWS resources within the same AWS Region.

Traffic between service and on-premises clients and applications

You have two connectivity options between your private network and AWS:

- An AWS Site-to-Site VPN connection. For more information, see [What is AWS Site-to-Site VPN?](#)
- An AWS Direct Connect connection. For more information, see [What is AWS Direct Connect?](#)

You get access to Amazon Aurora through the network by using AWS-published API operations. Clients must support Transport Layer Security (TLS) 1.0. We recommend TLS 1.2. Clients must also support cipher suites with Perfect Forward Secrecy (PFS), such as Ephemeral Diffie-Hellman (DHE) or Elliptic Curve Diffie-Hellman Ephemeral (ECDHE). Most modern systems such as Java 7 and later support these modes. Additionally, you must sign requests using an access key identifier and a secret access key that are associated with an IAM principal. Or you can use the [AWS security token service \(STS\)](#) to generate temporary security credentials to sign requests.

Identity and access management in Amazon Aurora

AWS Identity and Access Management (IAM) is an AWS service that helps an administrator securely control access to AWS resources. IAM administrators control who can be *authenticated* (signed in) and *authorized* (have permissions) to use Aurora resources. IAM is an AWS service that you can use with no additional charge.

Topics

- [Audience \(p. 1724\)](#)
- [Authenticating with identities \(p. 1724\)](#)
- [Managing access using policies \(p. 1726\)](#)
- [How Amazon Aurora works with IAM \(p. 1727\)](#)
- [Amazon Aurora identity-based policy examples \(p. 1730\)](#)
- [Preventing cross-service confused deputy problems \(p. 1741\)](#)
- [IAM database authentication \(p. 1743\)](#)
- [Troubleshooting Amazon Aurora identity and access \(p. 1769\)](#)

Audience

How you use AWS Identity and Access Management (IAM) differs, depending on the work you do in Aurora.

Service user – If you use the Aurora service to do your job, then your administrator provides you with the credentials and permissions that you need. As you use more Aurora features to do your work, you might need additional permissions. Understanding how access is managed can help you request the right permissions from your administrator. If you cannot access a feature in Aurora, see [Troubleshooting Amazon Aurora identity and access \(p. 1769\)](#).

Service administrator – If you're in charge of Aurora resources at your company, you probably have full access to Aurora. It's your job to determine which Aurora features and resources your employees should access. You must then submit requests to your IAM administrator to change the permissions of your service users. Review the information on this page to understand the basic concepts of IAM. To learn more about how your company can use IAM with Aurora, see [How Amazon Aurora works with IAM \(p. 1727\)](#).

IAM administrator – If you're an IAM administrator, you might want to learn details about how you can write policies to manage access to Aurora. To view example Aurora identity-based policies that you can use in IAM, see [Amazon Aurora identity-based policy examples \(p. 1730\)](#).

Authenticating with identities

Authentication is how you sign in to AWS using your identity credentials. For more information about signing in using the AWS Management Console, see [The IAM console and sign-in page](#) in the *IAM User Guide*.

You must be *authenticated* (signed in to AWS) as the AWS account root user, an IAM user, or by assuming an IAM role. You can also use your company's single sign-on authentication, or even sign in using Google or Facebook. In these cases, your administrator previously set up identity federation using IAM roles. When you access AWS using credentials from another company, you are assuming a role indirectly.

To sign in directly to the [AWS Management Console](#), use your password with your root user email or your IAM user name. You can access AWS programmatically using your root user or IAM user access keys. AWS provides SDK and command line tools to cryptographically sign your request using your credentials. If you don't use AWS tools, you must sign the request yourself. Do this using *Signature Version 4*, a protocol for authenticating inbound API requests. For more information about authenticating requests, see [Signature Version 4 signing process](#) in the *AWS General Reference*.

Regardless of the authentication method that you use, you might also be required to provide additional security information. For example, AWS recommends that you use multi-factor authentication (MFA) to increase the security of your account. To learn more, see [Using multi-factor authentication \(MFA\) in AWS](#) in the *IAM User Guide*.

AWS account root user

When you first create an AWS account, you begin with a single sign-in identity that has complete access to all AWS services and resources in the account. This identity is called the AWS account *root user* and is accessed by signing in with the email address and password that you used to create the account. We strongly recommend that you do not use the root user for your everyday tasks, even the administrative ones. Instead, adhere to the [best practice of using the root user only to create your first IAM user](#). Then securely lock away the root user credentials and use them to perform only a few account and service management tasks.

IAM users and groups

An *IAM user* is an identity within your AWS account that has specific permissions for a single person or application. An IAM user can have long-term credentials such as a user name and password or a set of access keys. To learn how to generate access keys, see [Managing access keys for IAM users](#) in the *IAM User Guide*. When you generate access keys for an IAM user, make sure you view and securely save the key pair. You cannot recover the secret access key in the future. Instead, you must generate a new access key pair.

An *IAM group* is an identity that specifies a collection of IAM users. You can't sign in as a group. You can use groups to specify permissions for multiple users at a time. Groups make permissions easier to manage for large sets of users. For example, you could have a group named *IAMAdmins* and give that group permissions to administer IAM resources.

Users are different from roles. A user is uniquely associated with one person or application, but a role is intended to be assumable by anyone who needs it. Users have permanent long-term credentials, but roles provide temporary credentials. To learn more, see [When to create an IAM user \(instead of a role\)](#) in the *IAM User Guide*.

You can authenticate to your DB cluster using IAM database authentication.

IAM database authentication works with Aurora. For more information about authenticating to your DB cluster using IAM, see [IAM database authentication \(p. 1743\)](#).

IAM roles

An *IAM role* is an identity within your AWS account that has specific permissions. It is similar to an IAM user, but is not associated with a specific person. You can temporarily assume an IAM role in the AWS Management Console by [switching roles](#). You can assume a role by calling an AWS CLI or AWS API operation or by using a custom URL. For more information about methods for using roles, see [Using IAM roles](#) in the *IAM User Guide*.

IAM roles with temporary credentials are useful in the following situations:

- **Temporary IAM user permissions** – An IAM user can assume an IAM role to temporarily take on different permissions for a specific task.

- **Federated user access** – Instead of creating an IAM user, you can use existing identities from AWS Directory Service, your enterprise user directory, or a web identity provider. These are known as *federated users*. AWS assigns a role to a federated user when access is requested through an [identity provider](#). For more information about federated users, see [Federated users and roles](#) in the *IAM User Guide*.
- **Cross-account access** – You can use an IAM role to allow someone (a trusted principal) in a different account to access resources in your account. Roles are the primary way to grant cross-account access. However, with some AWS services, you can attach a policy directly to a resource (instead of using a role as a proxy). To learn the difference between roles and resource-based policies for cross-account access, see [How IAM roles differ from resource-based policies](#) in the *IAM User Guide*.
- **AWS service access** – A service role is an [IAM role](#) that a service assumes to perform actions on your behalf. An IAM administrator can create, modify, and delete a service role from within IAM. For more information, see [Creating a role to delegate permissions to an AWS service](#) in the *IAM User Guide*.
- **Applications running on Amazon EC2** – You can use an IAM role to manage temporary credentials for applications that are running on an EC2 instance and making AWS CLI or AWS API requests. This is preferable to storing access keys within the EC2 instance. To assign an AWS role to an EC2 instance and make it available to all of its applications, you create an instance profile that is attached to the instance. An instance profile contains the role and enables programs that are running on the EC2 instance to get temporary credentials. For more information, see [Using an IAM role to grant permissions to applications running on Amazon EC2 instances](#) in the *IAM User Guide*.

To learn whether to use IAM roles, see [When to create an IAM role \(instead of a user\)](#) in the *IAM User Guide*.

Managing access using policies

You control access in AWS by creating policies and attaching them to IAM identities or AWS resources. A policy is an object in AWS that, when associated with an identity or resource, defines their permissions. AWS evaluates these policies when an entity (root user, IAM user, or IAM role) makes a request. Permissions in the policies determine whether the request is allowed or denied. Most policies are stored in AWS as JSON documents. For more information about the structure and contents of JSON policy documents, see [Overview of JSON policies](#) in the *IAM User Guide*.

An IAM administrator can use policies to specify who has access to AWS resources, and what actions they can perform on those resources. Every IAM entity (user or role) starts with no permissions. In other words, by default, users can do nothing, not even change their own password. To give a user permission to do something, an administrator must attach a permissions policy to a user. Or the administrator can add the user to a group that has the intended permissions. When an administrator gives permissions to a group, all users in that group are granted those permissions.

IAM policies define permissions for an action regardless of the method that you use to perform the operation. For example, suppose that you have a policy that allows the `iam:GetRole` action. A user with that policy can get role information from the AWS Management Console, the AWS CLI, or the AWS API.

Identity-based policies

Identity-based policies are JSON permissions policy documents that you can attach to an identity, such as an IAM user, role, or group. These policies control what actions that identity can perform, on which resources, and under what conditions. To learn how to create an identity-based policy, see [Creating IAM policies](#) in the *IAM User Guide*.

Identity-based policies can be further categorized as *inline policies* or *managed policies*. Inline policies are embedded directly into a single user, group, or role. Managed policies are standalone policies that you can attach to multiple users, groups, and roles in your AWS account. Managed policies include AWS managed policies and customer managed policies. To learn how to choose between a managed policy or an inline policy, see [Choosing between managed policies and inline policies](#) in the *IAM User Guide*.

The following AWS managed policies, which you can attach to users in your account, are specific to Amazon Aurora:

- **AmazonRDSReadOnlyAccess** – Grants read-only access to all Amazon Aurora resources for the AWS account specified.
- **AmazonRDSFullAccess** – Grants full access to all Amazon Aurora resources for the AWS account specified.

Other policy types

AWS supports additional, less-common policy types. These policy types can set the maximum permissions granted to you by the more common policy types.

- **Permissions boundaries** – A permissions boundary is an advanced feature in which you set the maximum permissions that an identity-based policy can grant to an IAM entity (IAM user or role). You can set a permissions boundary for an entity. The resulting permissions are the intersection of entity's identity-based policies and its permissions boundaries. Resource-based policies that specify the user or role in the `Principal` field are not limited by the permissions boundary. An explicit deny in any of these policies overrides the allow. For more information about permissions boundaries, see [Permissions boundaries for IAM entities](#) in the *IAM User Guide*.
- **Service control policies (SCPs)** – SCPs are JSON policies that specify the maximum permissions for an organization or organizational unit (OU) in AWS Organizations. AWS Organizations is a service for grouping and centrally managing multiple AWS accounts that your business owns. If you enable all features in an organization, then you can apply service control policies (SCPs) to any or all of your accounts. The SCP limits permissions for entities in member accounts, including each AWS account root user. For more information about Organizations and SCPs, see [How SCPs work](#) in the *AWS Organizations User Guide*.
- **Session policies** – Session policies are advanced policies that you pass as a parameter when you programmatically create a temporary session for a role or federated user. The resulting session's permissions are the intersection of the user or role's identity-based policies and the session policies. Permissions can also come from a resource-based policy. An explicit deny in any of these policies overrides the allow. For more information, see [Session policies](#) in the *IAM User Guide*.

Multiple policy types

When multiple types of policies apply to a request, the resulting permissions are more complicated to understand. To learn how AWS determines whether to allow a request when multiple policy types are involved, see [Policy evaluation logic](#) in the *IAM User Guide*.

For more information about identity and access management for Aurora, continue to the following pages:

- [How Amazon Aurora works with IAM \(p. 1727\)](#)
- [Troubleshooting Amazon Aurora identity and access \(p. 1769\)](#)

How Amazon Aurora works with IAM

Before you use IAM to manage access to Aurora, you should understand what IAM features are available to use with Aurora. To get a high-level view of how Aurora and other AWS services work with IAM, see [AWS services that work with IAM](#) in the *IAM User Guide*.

Topics

- [Aurora identity-based policies \(p. 1728\)](#)

- [Aurora resource-based policies \(p. 1729\)](#)
- [Authorization based on Aurora tags \(p. 1730\)](#)
- [Aurora IAM roles \(p. 1730\)](#)

Aurora identity-based policies

With IAM identity-based policies, you can specify allowed or denied actions and resources as well as the conditions under which actions are allowed or denied. Aurora supports specific actions, resources, and condition keys. To learn about all of the elements that you use in a JSON policy, see [IAM JSON policy elements reference](#) in the *IAM User Guide*.

Actions

The `Action` element of an IAM identity-based policy describes the specific action or actions that will be allowed or denied by the policy. Policy actions usually have the same name as the associated AWS API operation. The action is used in a policy to grant permissions to perform the associated operation.

Policy actions in Aurora use the following prefix before the action: `rds:`. For example, to grant someone permission to describe DB instances with the Amazon RDS `DescribeDBInstances` API operation, you include the `rds:DescribeDBInstances` action in their policy. Policy statements must include either an `Action` or `NotAction` element. Aurora defines its own set of actions that describe tasks that you can perform with this service.

To specify multiple actions in a single statement, separate them with commas as follows:

```
"Action": [  
    "rds:action1",  
    "rds:action2"]
```

You can specify multiple actions using wildcards (*). For example, to specify all actions that begin with the word `Describe`, include the following action:

```
"Action": "rds:Describe*"
```

To see a list of Aurora actions, see [Actions Defined by Amazon RDS](#) in the *Service Authorization Reference*.

Resources

The `Resource` element specifies the object or objects to which the action applies. Statements must include either a `Resource` or a `NotResource` element. You specify a resource using an ARN or using the wildcard (*) to indicate that the statement applies to all resources.

The DB instance resource has the following ARN:

```
arn:${Partition}:rds:${Region}:${Account}:{ ResourceType }/${Resource}
```

For more information about the format of ARNs, see [Amazon Resource Names \(ARNs\) and AWS service namespaces](#).

For example, to specify the `dbtest` DB instance in your statement, use the following ARN:

```
"Resource": "arn:aws:rds:us-west-2:123456789012:db:dbtest"
```

To specify all DB instances that belong to a specific account, use the wildcard (*):

```
"Resource": "arn:aws:rds:us-east-1:123456789012:db:/*"
```

Some RDS API operations, such as those for creating resources, cannot be performed on a specific resource. In those cases, you must use the wildcard (*).

```
"Resource": "*"
```

Many Amazon RDS API operations involve multiple resources. For example, `CreateDBInstance` creates a DB instance. You can specify that an IAM user must use a specific security group and parameter group when creating a DB instance. To specify multiple resources in a single statement, separate the ARNs with commas.

```
"Resource": [  
    "resource1",  
    "resource2"]
```

To see a list of Aurora resource types and their ARNs, see [Resources Defined by Amazon RDS](#) in the *Service Authorization Reference*. To learn with which actions you can specify the ARN of each resource, see [Actions Defined by Amazon RDS](#).

Condition keys

The `Condition` element (or `Condition block`) lets you specify conditions in which a statement is in effect. The `Condition` element is optional. You can build conditional expressions that use [condition operators](#), such as equals or less than, to match the condition in the policy with values in the request.

If you specify multiple `Condition` elements in a statement, or multiple keys in a single `Condition` element, AWS evaluates them using a logical AND operation. If you specify multiple values for a single condition key, AWS evaluates the condition using a logical OR operation. All of the conditions must be met before the statement's permissions are granted.

You can also use placeholder variables when you specify conditions. For example, you can grant an IAM user permission to access a resource only if it is tagged with their IAM user name. For more information, see [IAM policy elements: Variables and tags](#) in the *IAM User Guide*.

Aurora defines its own set of condition keys and also supports using some global condition keys. To see all AWS global condition keys, see [AWS global condition context keys](#) in the *IAM User Guide*.

All RDS API operations support the `aws:RequestedRegion` condition key.

To see a list of Aurora condition keys, see [Condition Keys for Amazon RDS](#) in the *Service Authorization Reference*. To learn with which actions and resources you can use a condition key, see [Actions Defined by Amazon RDS](#).

Examples

To view examples of Aurora identity-based policies, see [Amazon Aurora identity-based policy examples \(p. 1730\)](#).

Aurora resource-based policies

Aurora does not support resource-based policies.

Authorization based on Aurora tags

You can attach tags to Aurora resources or pass tags in a request to Aurora. To control access based on tags, you provide tag information in the [condition element](#) of a policy using the `aws:ResourceTag/key-name`, `aws:RequestTag/key-name`, or `aws:TagKeys` condition keys. For more information, see [Controlling access to AWS resources using tags](#) in the *IAM User Guide*. For more information about tagging Aurora resources, see [Specifying conditions: Using custom tags \(p. 1737\)](#).

To view an example identity-based policy for limiting access to a resource based on the tags on that resource, see [Grant permission for actions on a resource with a specific tag with two different values \(p. 1734\)](#).

Aurora IAM roles

An [IAM role](#) is an entity within your AWS account that has specific permissions.

Using temporary credentials with Aurora

You can use temporary credentials to sign in with federation, assume an IAM role, or to assume a cross-account role. You obtain temporary security credentials by calling AWS STS API operations such as [AssumeRole](#) or [GetFederationToken](#).

Aurora supports using temporary credentials.

Service-linked roles

[Service-linked roles](#) allow AWS services to access resources in other services to complete an action on your behalf. Service-linked roles appear in the Roles list in the IAM Management Console and are owned by the service. An IAM administrator can view but not edit the permissions for service-linked roles.

Aurora supports service-linked roles. For details about creating or managing Aurora service-linked roles, see [Using service-linked roles for Amazon Aurora \(p. 1783\)](#).

Service roles

This feature allows a service to assume a [service role](#) on your behalf. This role allows the service to access resources in other services to complete an action on your behalf. Service roles appear in the Roles list in the IAM Management Console and are owned by your account. This means that an IAM administrator can change the permissions for this role. However, doing so might break the functionality of the service.

Aurora supports service roles.

Amazon Aurora identity-based policy examples

By default, IAM users and roles don't have permission to create or modify Aurora resources. They also can't perform tasks using the AWS Management Console, AWS CLI, or AWS API. An IAM administrator must create IAM policies that grant users and roles permission to perform specific API operations on the specified resources they need. The administrator must then attach those policies to the IAM users or groups that require those permissions.

To learn how to create an IAM identity-based policy using these example JSON policy documents, see [Creating policies on the JSON tab](#) in the *IAM User Guide*.

Topics

- [Policy best practices \(p. 1731\)](#)

- [Using the Aurora console \(p. 1731\)](#)
- [Allow users to view their own permissions \(p. 1732\)](#)
- [Allow a user to create DB instances in an AWS account \(p. 1732\)](#)
- [Permissions required to use the console \(p. 1733\)](#)
- [Allow a user to perform any describe action on any RDS resource \(p. 1734\)](#)
- [Allow a user to create a DB instance that uses the specified DB parameter group and subnet group \(p. 1734\)](#)
- [Grant permission for actions on a resource with a specific tag with two different values \(p. 1734\)](#)
- [Prevent a user from deleting a DB instance \(p. 1735\)](#)
- [Deny all access to a resource \(p. 1735\)](#)
- [Example policies: Using condition keys \(p. 1735\)](#)
- [Specifying conditions: Using custom tags \(p. 1737\)](#)

Policy best practices

Identity-based policies are very powerful. They determine whether someone can create, access, or delete Aurora resources in your account. These actions can incur costs for your AWS account. When you create or edit identity-based policies, follow these guidelines and recommendations:

- **Get Started Using AWS Managed Policies** – To start using Aurora quickly, use AWS managed policies to give your employees the permissions they need. These policies are already available in your account and are maintained and updated by AWS. For more information, see [Get started using permissions with AWS managed policies](#) in the *IAM User Guide*.
- **Grant Least Privilege** – When you create custom policies, grant only the permissions required to perform a task. Start with a minimum set of permissions and grant additional permissions as necessary. Doing so is more secure than starting with permissions that are too lenient and then trying to tighten them later. For more information, see [Grant least privilege](#) in the *IAM User Guide*.
- **Enable MFA for Sensitive Operations** – For extra security, require IAM users to use multi-factor authentication (MFA) to access sensitive resources or API operations. For more information, see [Using multi-factor authentication \(MFA\) in AWS](#) in the *IAM User Guide*.
- **Use Policy Conditions for Extra Security** – To the extent that it's practical, define the conditions under which your identity-based policies allow access to a resource. For example, you can write conditions to specify a range of allowable IP addresses that a request must come from. You can also write conditions to allow requests only within a specified date or time range, or to require the use of SSL or MFA. For more information, see [IAM JSON policy elements: Condition](#) in the *IAM User Guide*.

Using the Aurora console

To access the Amazon Aurora console, you must have a minimum set of permissions. These permissions must enable you to list and view details about the Aurora resources in your AWS account. You can create an identity-based policy that is more restrictive than the minimum required permissions. However, if you do, the console doesn't function as intended for entities (IAM users or roles) with that policy.

To ensure that those entities can still use the Aurora console, also attach the following AWS managed policy to the entities. For more information, see [Adding permissions to a user](#) in the *IAM User Guide*.

AmazonRDSReadOnlyAccess

You don't need to allow minimum console permissions for users that are making calls only to the AWS CLI or the AWS API. Instead, allow access to only the actions that match the API operation that you're trying to perform.

Allow users to view their own permissions

This example shows how you might create a policy that allows IAM users to view the inline and managed policies that are attached to their user identity. This policy includes permissions to complete this action on the console or programmatically using the AWS CLI or AWS API.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ViewOwnUserInfo",
            "Effect": "Allow",
            "Action": [
                "iam:GetUserPolicy",
                "iam>ListGroupsForUser",
                "iam>ListAttachedUserPolicies",
                "iam>ListUserPolicies",
                "iam GetUser"
            ],
            "Resource": [
                "arn:aws:iam::*:user/${aws:username}"
            ]
        },
        {
            "Sid": "NavigateInConsole",
            "Effect": "Allow",
            "Action": [
                "iam:GetGroupPolicy",
                "iam:GetPolicyVersion",
                "iam GetPolicy",
                "iam>ListAttachedGroupPolicies",
                "iam>ListGroupPolicies",
                "iam>ListPolicyVersions",
                "iam>ListPolicies",
                "iam>ListUsers"
            ],
            "Resource": "*"
        }
    ]
}
```

Allow a user to create DB instances in an AWS account

The following is an example policy that allows the user with the ID 123456789012 to create DB instances for your AWS account. The policy requires that the name of the new DB instance begin with test. The new DB instance must also use the MySQL database engine and the db.t2.micro DB instance class. In addition, the new DB instance must use an option group and a DB parameter group that starts with default, and it must use the default subnet group.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowCreateDBInstanceOnly",
            "Effect": "Allow",
            "Action": [
                "rds>CreateDBInstance"
            ],
            "Resource": [
                "arn:aws:rds::123456789012:db:test*",
                "arn:aws:rds::123456789012:og:default*",
                "arn:aws:rds::123456789012:sg:default"
            ]
        }
    ]
}
```

```
        "arn:aws:rds:*:123456789012:pg:default*",
        "arn:aws:rds:*:123456789012:subgrp:default"
    ],
    "Condition": {
        "StringEquals": {
            "rds:DatabaseEngine": "mysql",
            "rds:DatabaseClass": "db.t2.micro"
        }
    }
}
]
```

The policy includes a single statement that specifies the following permissions for the IAM user:

- The policy allows the IAM user to create a DB instance using the [CreateDBInstance](#) API operation (this also applies to the [create-db-instance](#) AWS CLI command and the AWS Management Console).
- The `Resource` element specifies that the user can perform actions on or with resources. You specify resources using an Amazon Resources Name (ARN). This ARN includes the name of the service that the resource belongs to (`rds`), the AWS Region (* indicates any region in this example), the user account number (123456789012 is the user ID in this example), and the type of resource. For more information about creating ARNs, see [Working with Amazon Resource Names \(ARNs\) in Amazon RDS \(p. 482\)](#).

The `Resource` element in the example specifies the following policy constraints on resources for the user:

- The DB instance identifier for the new DB instance must begin with `test` (for example, `testCustomerData1`, `test-region2-data`).
- The option group for the new DB instance must begin with `default`.
- The DB parameter group for the new DB instance must begin with `default`.
- The subnet group for the new DB instance must be the `default` subnet group.
- The `Condition` element specifies that the DB engine must be MySQL and the DB instance class must be db.t2.micro. The `Condition` element specifies the conditions when a policy should take effect. You can add additional permissions or restrictions by using the `Condition` element. For more information about specifying conditions, see [Condition keys \(p. 1729\)](#). This example specifies the `rds:DatabaseEngine` and `rds:DatabaseClass` conditions. For information about the valid condition values for `rds:DatabaseEngine`, see the list under the `Engine` parameter in [CreateDBInstance](#). For information about the valid condition values for `rds:DatabaseClass`, see [Supported DB engines for DB instance classes \(p. 54\)](#).

The policy doesn't specify the `Principal` element because in an identity-based policy you don't specify the principal who gets the permission. When you attach policy to a user, the user is the implicit principal. When you attach a permission policy to an IAM role, the principal identified in the role's trust policy gets the permissions.

To see a list of Aurora actions, see [Actions Defined by Amazon RDS](#) in the *Service Authorization Reference*.

Permissions required to use the console

For a user to work with the console, that user must have a minimum set of permissions. These permissions allow the user to describe the Amazon Aurora resources for their AWS account and to provide other related information, including Amazon EC2 security and network information.

If you create an IAM policy that is more restrictive than the minimum required permissions, the console doesn't function as intended for users with that IAM policy. To ensure that those users can still use the console, also attach the `AmazonRDSReadOnlyAccess` managed policy to the user, as described in [Managing access using policies \(p. 1726\)](#).

You don't need to allow minimum console permissions for users that are making calls only to the AWS CLI or the Amazon RDS API.

The following policy grants full access to all Amazon Aurora resources for the root AWS account:

```
AmazonRDSFullAccess
```

Allow a user to perform any describe action on any RDS resource

The following permissions policy grants permissions to a user to run all of the actions that begin with `Describe`. These actions show information about an RDS resource, such as a DB instance. The wildcard character (*) in the `Resource` element indicates that the actions are allowed for all Amazon Aurora resources owned by the account.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowRDSDescribe",
            "Effect": "Allow",
            "Action": "rds:Describe*",
            "Resource": "*"
        }
    ]
}
```

Allow a user to create a DB instance that uses the specified DB parameter group and subnet group

The following permissions policy grants permissions to allow a user to only create a DB instance that must use the `mydbpg` DB parameter group and the `mydbsubnetgroup` DB subnet group.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor0",
            "Effect": "Allow",
            "Action": "rds>CreateDBInstance",
            "Resource": [
                "arn:aws:rds:::pg:mydbpg",
                "arn:aws:rds:::subgrp:mydbsubnetgroup"
            ]
        }
    ]
}
```

Grant permission for actions on a resource with a specific tag with two different values

You can use conditions in your identity-based policy to control access to Aurora resources based on tags. The following policy allows permission to perform the `ModifyDBInstance` and `CreateDBSnapshot` APIs on DB instances with either the `stage` tag set to `development` or `test`.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowDevTestCreate",
            "Effect": "Allow",
            "Action": [
                "rds:ModifyDBInstance",
                "rds>CreateDBSnapshot"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "rds:db-tag/stage": [
                        "development",
                        "test"
                    ]
                }
            }
        }
    ]
}
```

Prevent a user from deleting a DB instance

The following permissions policy grants permissions to prevent a user from deleting a specific DB instance. For example, you might want to deny the ability to delete your production DB instances to any user that is not an administrator.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "DenyDelete1",
            "Effect": "Deny",
            "Action": "rds>DeleteDBInstance",
            "Resource": "arn:aws:rds:us-west-2:123456789012:db:my-mysql-instance"
        }
    ]
}
```

Deny all access to a resource

You can explicitly deny access to a resource. Deny policies take precedence over allow policies. The following policy explicitly denies a user the ability to manage a resource:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Deny",
            "Action": "rds:*",
            "Resource": "arn:aws:rds:us-east-1:123456789012:db:mydb"
        }
    ]
}
```

Example policies: Using condition keys

Following are examples of how you can use condition keys in Amazon Aurora IAM permissions policies.

Example 1: Grant permission to create a DB instance that uses a specific DB engine and isn't MultiAZ

The following policy uses an RDS condition key and allows a user to create only DB instances that use the MySQL database engine and don't use MultiAZ. The Condition element indicates the requirement that the database engine is MySQL.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowMySQLCreate",
            "Effect": "Allow",
            "Action": "rds>CreateDBInstance",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "rds:DatabaseEngine": "mysql"
                },
                "Bool": {
                    "rds:MultiAz": false
                }
            }
        }
    ]
}
```

Example 2: Explicitly deny permission to create DB instances for certain DB instance classes and create DB instances that use Provisioned IOPS

The following policy explicitly denies permission to create DB instances that use the DB instance classes `r3.8xlarge` and `m4.10xlarge`, which are the largest and most expensive DB instance classes. This policy also prevents users from creating DB instances that use Provisioned IOPS, which incurs an additional cost.

Explicitly denying permission supersedes any other permissions granted. This ensures that identities do not accidentally get permission that you never want to grant.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "DenyLargeCreate",
            "Effect": "Deny",
            "Action": "rds>CreateDBInstance",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "rds:DatabaseClass": [
                        "db.r3.8xlarge",
                        "db.m4.10xlarge"
                    ]
                }
            }
        },
        {
            "Sid": "DenyPIOPSCreate",
            "Effect": "Deny",
            "Action": "rds>CreateDBInstance",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "rds:ProvisionedIOPS": "true"
                }
            }
        }
    ]
}
```

```

        "Condition": {
            "NumericNotEquals": {
                "rds:Piops": "0"
            }
        }
    ]
}

```

Example 3: Limit the set of tag keys and values that can be used to tag a resource

The following policy uses an RDS condition key and allows the addition of a tag with the key `stage` to be added to a resource with the values `test`, `qa`, and `production`.

```

{
    {
        "Version" : "2012-10-17",
        "Statement" : [
            {
                "Effect" : "Allow",
                "Action" : [ "rds:AddTagsToResource", "rds:RemoveTagsFromResource" ],
                "Resource" : "*",
                "Condition" : { "streq" : { "rds:req-tag/stage" : [ "test", "qa", "production" ] } }
            }
        ]
    }
}

```

Specifying conditions: Using custom tags

Amazon Aurora supports specifying conditions in an IAM policy using custom tags.

For example, suppose that you add a tag named `environment` to your DB instances with values such as `beta`, `staging`, `production`, and so on. If you do, you can create a policy that restricts certain users to DB instances based on the `environment` tag value.

Note

Custom tag identifiers are case-sensitive.

The following table lists the RDS tag identifiers that you can use in a `Condition` element.

RDS tag identifier	Applies to
<code>db-tag</code>	DB instances, including read replicas
<code>snapshot-tag</code>	DB snapshots
<code>ri-tag</code>	Reserved DB instances
<code>secgrp-tag</code>	DB security groups
<code>og-tag</code>	DB option groups
<code>pg-tag</code>	DB parameter groups
<code>subgrp-tag</code>	DB subnet groups

RDS tag identifier	Applies to
es-tag	Event subscriptions
cluster-tag	DB clusters
cluster-pg-tag	DB cluster parameter groups
cluster-snapshot-tag	DB cluster snapshots

The syntax for a custom tag condition is as follows:

```
"Condition": {"StringEquals": {"rds:rds-tag-identifier/tag-name": ["value"]}}
```

For example, the following Condition element applies to DB instances with a tag named environment and a tag value of production.

```
"Condition": {"StringEquals": {"rds:db-tag/environment": ["production"]}}
```

For information about creating tags, see [Tagging Amazon RDS resources \(p. 474\)](#).

Important

If you manage access to your RDS resources using tagging, we recommend that you secure access to the tags for your RDS resources. You can manage access to tags by creating policies for the AddTagsToResource and RemoveTagsFromResource actions. For example, the following policy denies users the ability to add or remove tags for all resources. You can then create policies to allow specific users to add or remove tags.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "DenyTagUpdates",
            "Effect": "Deny",
            "Action": [
                "rds:AddTagsToResource",
                "rds:RemoveTagsFromResource"
            ],
            "Resource": "*"
        }
    ]
}
```

To see a list of Aurora actions, see [Actions Defined by Amazon RDS](#) in the *Service Authorization Reference*.

Example policies: Using custom tags

Following are examples of how you can use custom tags in Amazon Aurora IAM permissions policies. For more information about adding tags to an Amazon Aurora resource, see [Working with Amazon Resource Names \(ARNs\) in Amazon RDS \(p. 482\)](#).

Note

All examples use the us-west-2 region and contain fictitious account IDs.

Example 1: Grant permission for actions on a resource with a specific tag with two different values

The following policy allows permission to perform the `ModifyDBInstance` and `CreateDBSnapshot` APIs on DB instances with either the stage tag set to development or test.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowDevTestCreate",
            "Effect": "Allow",
            "Action": [
                "rds:ModifyDBInstance",
                "rds>CreateDBSnapshot"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "rds:db-tag/stage": [
                        "development",
                        "test"
                    ]
                }
            }
        }
    ]
}
```

Example 2: Explicitly deny permission to create a DB instance that uses specified DB parameter groups

The following policy explicitly denies permission to create a DB instance that uses DB parameter groups with specific tag values. You might apply this policy if you require that a specific customer-created DB parameter group always be used when creating DB instances. Policies that use Deny are most often used to restrict access that was granted by a broader policy.

Explicitly denying permission supersedes any other permissions granted. This ensures that identities to not accidentally get permission that you never want to grant.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "DenyProductionCreate",
            "Effect": "Deny",
            "Action": "rds:CreateDBInstance",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "rds:pg-tag/usage": "prod"
                }
            }
        }
    ]
}
```

Example 3: Grant permission for actions on a DB instance with an instance name that is prefixed with a user name

The following policy allows permission to call any API (except to `AddTagsToResource` or `RemoveTagsFromResource`) on a DB instance that has a DB instance name that is prefixed with the user's name and that has a tag called `stage` equal to `devo` or that has no tag called `stage`.

The Resource line in the policy identifies a resource by its Amazon Resource Name (ARN). For more information about using ARNs with Amazon Aurora resources, see [Working with Amazon Resource Names \(ARNs\) in Amazon RDS \(p. 482\)](#).

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "AllowFullDevAccessNoTags",  
            "Effect": "Allow",  
            "NotAction": [  
                "rds:AddTagsToResource",  
                "rds:RemoveTagsFromResource"  
            ],  
            "Resource": "arn:aws:rds:*:123456789012:db:${aws:username}*",  
            "Condition": {  
                "StringEqualsIfExists": {  
                    "rds:db-tag/stage": "devo"  
                }  
            }  
        }  
    ]  
}
```

Preventing cross-service confused deputy problems

The *confused deputy problem* is a security issue where an entity that doesn't have permission to perform an action can coerce a more-privileged entity to perform the action. In AWS, cross-service impersonation can result in the confused deputy problem.

Cross-service impersonation can occur when one service (the *calling service*) calls another service (the *called service*). The calling service can be manipulated to use its permissions to act on another customer's resources in a way that it shouldn't have permission to access. To prevent this, AWS provides tools that can help you protect your data for all services with service principals that have been given access to resources in your account. For more information, see [The confused deputy problem](#) in the *IAM User Guide*.

To limit the permissions that Amazon RDS gives another service for a specific resource, we recommend using the `aws:SourceArn` and `aws:SourceAccount` global condition context keys in resource policies.

In some cases, the `aws:SourceArn` value doesn't contain the account ID, for example when you use the Amazon Resource Name (ARN) for an Amazon S3 bucket. In these cases, make sure to use both global condition context keys to limit permissions. In some cases, you use both global condition context keys and the `aws:SourceArn` value contains the account ID. In these cases, make sure that the `aws:SourceAccount` value and the account in the `aws:SourceArn` use the same account ID when they're used in the same policy statement. If you want only one resource to be associated with the cross-service access, use `aws:SourceArn`. If you want to allow any resource in the specified AWS account to be associated with the cross-service use, use `aws:SourceAccount`.

Make sure that the value of `aws:SourceArn` is an ARN for an Amazon RDS resource type. For more information, see [Working with Amazon Resource Names \(ARNs\) in Amazon RDS \(p. 482\)](#).

The most effective way to protect against the confused deputy problem is to use the `aws:SourceArn` global condition context key with the full ARN of the resource. In some cases, you might not know the full ARN of the resource or you might be specifying multiple resources. In these cases, use the `aws:SourceArn` global context condition key with wildcards (*) for the unknown portions of the ARN. An example is `arn:aws:rds:*:123456789012:*`.

The following example shows how you can use the `aws:SourceArn` and `aws:SourceAccount` global condition context keys in Amazon RDS to prevent the confused deputy problem.

```
{  
    "Version": "2012-10-17",  
    "Statement": {  
        "Sid": "ConfusedDeputyPreventionExamplePolicy",  
        "Effect": "Allow",  
        "Principal": {  
            "Service": "rds.amazonaws.com"  
        },  
        "Action": "sts:AssumeRole",  
        "Condition": {  
            "ArnLike": {  
                "aws:SourceArn": "arn:aws:rds:us-east-1:123456789012:db/mydbinstance"  
            },  
            "StringEquals": {  
                "aws:SourceAccount": "123456789012"  
            }  
        }  
    }  
}
```

For more examples of policies that use the `aws:SourceArn` and `aws:SourceAccount` global condition context keys, see the following sections:

- [Setting up access to an Amazon S3 bucket \(p. 1439\)](#) (PostgreSQL import)
- [Setting up access to an Amazon S3 bucket \(p. 1452\)](#) (PostgreSQL export)

IAM database authentication

You can authenticate to your DB cluster using AWS Identity and Access Management (IAM) database authentication. IAM database authentication works with MariaDB, Aurora MySQL and Aurora PostgreSQL. With this authentication method, you don't need to use a password when you connect to a DB cluster. Instead, you use an authentication token.

An *authentication token* is a unique string of characters that Amazon Aurora generates on request. Authentication tokens are generated using AWS Signature Version 4. Each token has a lifetime of 15 minutes. You don't need to store user credentials in the database, because authentication is managed externally using IAM. You can also still use standard database authentication. The token is only used for authentication and doesn't affect the session after it is established.

IAM database authentication provides the following benefits:

- Network traffic to and from the database is encrypted using Secure Socket Layer (SSL) or Transport Layer Security (TLS). For more information about using SSL/TLS with Amazon Aurora, see [Using SSL/TLS to encrypt a connection to a DB cluster \(p. 1714\)](#).
- You can use IAM to centrally manage access to your database resources, instead of managing access individually on each DB cluster.
- For applications running on Amazon EC2, you can use profile credentials specific to your EC2 instance to access your database instead of a password, for greater security.

Topics

- [Availability for IAM database authentication \(p. 1743\)](#)
- [Limitations for IAM database authentication \(p. 1744\)](#)
- [MariaDB and Aurora MySQL recommendations for IAM database authentication \(p. 1744\)](#)
- [Enabling and disabling IAM database authentication \(p. 1744\)](#)
- [Creating and using an IAM policy for IAM database access \(p. 1746\)](#)
- [Creating a database account using IAM authentication \(p. 1749\)](#)
- [Connecting to your DB cluster using IAM authentication \(p. 1750\)](#)

Availability for IAM database authentication

IAM database authentication is available for the following database engines:

- **Aurora MySQL**
 - Aurora MySQL version 3, all minor versions
 - Aurora MySQL version 2, all minor versions
 - Aurora MySQL version 1.10 and higher 1.1 minor versions
- **Aurora PostgreSQL**
 - All Aurora PostgreSQL 13 versions
 - All Aurora PostgreSQL 12 versions
 - Aurora PostgreSQL 11.6 and higher 11 versions
 - Aurora PostgreSQL 10.11 and higher 10 versions
 - Aurora PostgreSQL 9.6.16 and higher 9.6 versions

For more information, see [Amazon Aurora PostgreSQL releases and engine versions \(p. 1599\)](#).

For Aurora MySQL, all supported DB instance classes support IAM database authentication, except for db.t2.small and db.t3.small. For information about the supported DB instance classes, see [Supported DB engines for DB instance classes \(p. 54\)](#).

Limitations for IAM database authentication

When using IAM database authentication, the following limitations apply:

- The maximum number of connections per second for your DB cluster might be limited depending on its DB instance class and your workload.
- Currently, IAM database authentication doesn't support all global condition context keys.

For more information about global condition context keys, see [AWS global condition context keys](#) in the *IAM User Guide*.

- Currently, IAM database authentication isn't supported for CNAMEs.
- For PostgreSQL, if the IAM role (`rds_iam`) is added to the master user, IAM authentication takes precedence over Password authentication so the master user has to log in as an IAM user.

MariaDB and Aurora MySQL recommendations for IAM database authentication

We recommend the following when using the MariaDB or Aurora MySQL DB engine:

- Use IAM database authentication as a mechanism for temporary, personal access to databases.
- Use IAM database authentication only for workloads that can be easily retried.
- Use IAM database authentication when your application requires fewer than 200 new IAM database authentication connections per second.

The database engines that work with Amazon Aurora don't impose any limits on authentication attempts per second. However, when you use IAM database authentication, your application must generate an authentication token. Your application then uses that token to connect to the DB cluster. If you exceed the limit of maximum new connections per second, then the extra overhead of IAM database authentication can cause connection throttling. The extra overhead can cause even existing connections to drop. For information about the maximum total connections for Aurora MySQL, see [Maximum connections to an Aurora MySQL DB instance \(p. 813\)](#).

Note

These recommendations don't apply to Aurora PostgreSQL DB clusters.

Enabling and disabling IAM database authentication

By default, IAM database authentication is disabled on DB clusters. You can enable or disable IAM database authentication using the AWS Management Console, AWS CLI, or the API.

You can enable IAM database authentication when you perform one of the following actions:

- To create a new DB cluster with IAM database authentication enabled, see [Creating an Amazon Aurora DB cluster \(p. 125\)](#).
- To modify a DB cluster to enable IAM database authentication, see [Modifying an Amazon Aurora DB cluster \(p. 372\)](#).
- To restore a DB cluster from a snapshot with IAM database authentication enabled, see [Restoring from a DB cluster snapshot \(p. 497\)](#).
- To restore a DB cluster to a point in time with IAM database authentication enabled, see [Restoring a DB cluster to a specified time \(p. 537\)](#).

Console

Each creation or modification workflow has a **Database authentication** section, where you can enable or disable IAM database authentication. In that section, choose **Password and IAM database authentication** to enable IAM database authentication.

To enable or disable IAM database authentication for an existing DB cluster

1. Open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. In the navigation pane, choose **Databases**.
3. Choose the DB cluster that you want to modify.

Note

You can only enable IAM authentication if all DB instances in the DB cluster are compatible with IAM. Check the compatibility requirements in [Availability for IAM database authentication \(p. 1743\)](#).

4. Choose **Modify**.
5. In the **Database authentication** section, choose **Password and IAM database authentication** to enable IAM database authentication.
6. Choose **Continue**.
7. To apply the changes immediately, choose **Immediately** in the **Scheduling of modifications** section.
8. Choose **Modify cluster**.

AWS CLI

To create a new DB cluster with IAM authentication by using the AWS CLI, use the `create-db-cluster` command. Specify the `--enable-iam-database-authentication` option.

To update an existing DB cluster to have or not have IAM authentication, use the AWS CLI command `modify-db-cluster`. Specify either the `--enable-iam-database-authentication` or `--no-enable-iam-database-authentication` option, as appropriate.

Note

You can only enable IAM authentication if all DB instances in the DB cluster are compatible with IAM. Check the compatibility requirements in [Availability for IAM database authentication \(p. 1743\)](#).

By default, Aurora performs the modification during the next maintenance window. If you want to override this and enable IAM DB authentication as soon as possible, use the `--apply-immediately` parameter.

If you are restoring a DB cluster, use one of the following AWS CLI commands:

- `restore-db-cluster-to-point-in-time`
- `restore-db-cluster-from-db-snapshot`

The IAM database authentication setting defaults to that of the source snapshot. To change this setting, set the `--enable-iam-database-authentication` or `--no-enable-iam-database-authentication` option, as appropriate.

RDS API

To create a new DB instance with IAM authentication by using the API, use the API operation `CreateDBCluster`. Set the `EnableIAMDatabaseAuthentication` parameter to `true`.

To update an existing DB cluster to have IAM authentication, use the API operation [ModifyDBCluster](#). Set the `EnableIAMDatabaseAuthentication` parameter to `true` to enable IAM authentication, or `false` to disable it.

Note

You can only enable IAM authentication if all DB instances in the DB cluster are compatible with IAM. Check the compatibility requirements in [Availability for IAM database authentication \(p. 1743\)](#).

If you are restoring a DB cluster, use one of the following API operations:

- [RestoreDBClusterFromSnapshot](#)
- [RestoreDBClusterToPointInTime](#)

The IAM database authentication setting defaults to that of the source snapshot. To change this setting, set the `EnableIAMDatabaseAuthentication` parameter to `true` to enable IAM authentication, or `false` to disable it.

Creating and using an IAM policy for IAM database access

To allow an IAM user or role to connect to your DB cluster, you must create an IAM policy. After that, you attach the policy to an IAM user or role.

Note

To learn more about IAM policies, see [Identity and access management in Amazon Aurora \(p. 1724\)](#).

The following example policy allows an IAM user to connect to a DB cluster using IAM database authentication.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": [  
                "rds-db:connect"  
            ],  
            "Resource": [  
                "arn:aws:rds-db:us-east-2:1234567890:dbuser:cluster-ABCDEFGHIJKLM01234/db_user"  
            ]  
        }  
    ]  
}
```

Important

An IAM administrator user can access DB clusters without explicit permissions in an IAM policy. The example in [Create an IAM user \(p. 84\)](#) creates an IAM administrator user. If you want to restrict administrator access to DB clusters, you can create an IAM role with the appropriate, lesser privileged permissions and assign it to the administrator.

Note

Don't confuse the `rds-db:` prefix with other RDS API operation prefixes that begin with `rds:`. You use the `rds-db:` prefix and the `rds-db:connect` action only for IAM database authentication. They aren't valid in any other context.

Currently, the IAM console displays an error for policies with the `rds-db:connect` action. You can ignore this error.

The example policy includes a single statement with the following elements:

- **Effect** – Specify Allow to grant access to the DB cluster. If you don't explicitly allow access, then access is denied by default.
- **Action** – Specify `rds-db:connect` to allow connections to the DB cluster.
- **Resource** – Specify an Amazon Resource Name (ARN) that describes one database account in one DB cluster. The ARN format is as follows.

```
arn:aws:rds-db:region:account-id:dbuser:DbClusterResourceId/db-user-name
```

In this format, replace the following:

- *region* is the AWS Region for the DB cluster. In the example policy, the AWS Region is `us-east-2`.
- *account-id* is the AWS account number for the DB cluster. In the example policy, the account number is `1234567890`.
- *DbClusterResourceId* is the identifier for the DB cluster. This identifier is unique to an AWS Region and never changes. In the example policy, the identifier is `cluster-ABCDEFGHIJKL01234`.

To find a DB cluster resource ID in the AWS Management Console for Amazon Aurora, choose the DB cluster to see its details. Then choose the **Configuration** tab. The **Resource ID** is shown in the **Configuration** section.

Alternatively, you can use the AWS CLI command to list the identifiers and resource IDs for all of your DB cluster in the current AWS Region, as shown following.

```
aws rds describe-db-clusters --query "DBClusters[*].  
[DBClusterIdentifier,DbClusterResourceId]"
```

Note

If you are connecting to a database through RDS Proxy, specify the proxy resource ID, such as `prx-ABCDEFGHIJKL01234`. For information about using IAM database authentication with RDS Proxy, see [Connecting to a proxy using IAM authentication \(p. 304\)](#).

- *db-user-name* is the name of the database account to associate with IAM authentication. In the example policy, the database account is `db_user`.

You can construct other ARNs to support various access patterns. The following policy allows access to two different database accounts in a DB cluster.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": [  
                "rds-db:connect"  
            ],  
            "Resource": [  
                "arn:aws:rds-db:us-east-2:123456789012:dbuser:cluster-ABCDEFGHIJKL01234/  
jane_doe",  
                "arn:aws:rds-db:us-east-2:123456789012:dbuser:cluster-ABCDEFGHIJKL01234/  
mary_roe"  
            ]  
        }  
    ]  
}
```

```
    ]  
}
```

The following policy uses the "*" character to match all DB clusters and database accounts for a particular AWS account and AWS Region.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": [  
                "rds-db:connect"  
            ],  
            "Resource": [  
                "arn:aws:rds-db:us-east-2:1234567890:dbuser:/*"  
            ]  
        }  
    ]  
}
```

The following policy matches all of the DB clusters for a particular AWS account and AWS Region. However, the policy only grants access to DB clusters that have a `jane_doe` database account.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": [  
                "rds-db:connect"  
            ],  
            "Resource": [  
                "arn:aws:rds-db:us-east-2:123456789012:dbuser:*/jane_doe"  
            ]  
        }  
    ]  
}
```

The IAM user or role has access to only those databases that the database user does. For example, suppose that your DB cluster has a database named `dev`, and another database named `test`. If the database user `jane_doe` has access only to `dev`, any IAM users or roles that access that DB cluster with the `jane_doe` user also have access only to `dev`. This access restriction is also true for other database objects, such as tables, views, and so on.

An IAM administrator must create IAM policies that grant users and roles permission to perform specific API operations on the specified resources they need. The administrator must then attach those policies to the IAM users or groups that require those permissions. For examples of policies, see [Amazon Aurora identity-based policy examples \(p. 1730\)](#).

Attaching an IAM policy to an IAM user or role

After you create an IAM policy to allow database authentication, you need to attach the policy to an IAM user or role. For a tutorial on this topic, see [Create and attach your first customer managed policy](#) in the *IAM User Guide*.

As you work through the tutorial, you can use one of the policy examples shown in this section as a starting point and tailor it to your needs. At the end of the tutorial, you have an IAM user with an attached policy that can make use of the `rds-db:connect` action.

Note

You can map multiple IAM users or roles to the same database user account. For example, suppose that your IAM policy specified the following resource ARN.

```
arn:aws:rds-db:us-east-2:123456789012:dbuser:cluster-12ABC34DEFG5HIJ6KLMNOP78QR/jane_doe
```

If you attach the policy to IAM users *Jane*, *Bob*, and *Diego*, then each of those users can connect to the specified DB cluster using the `jane_doe` database account.

Creating a database account using IAM authentication

With IAM database authentication, you don't need to assign database passwords to the user accounts you create. If you remove an IAM user that is mapped to a database account, you should also remove the database account with the `DROP USER` statement.

Note

The user name used for IAM authentication must match the case of the user name in the database.

Topics

- [Using IAM authentication with MariaDB or MySQL \(p. 1749\)](#)
- [Using IAM authentication with PostgreSQL \(p. 1750\)](#)

Using IAM authentication with MariaDB or MySQL

With MariaDB or MySQL, authentication is handled by `AWSAuthenticationPlugin`—an AWS-provided plugin that works seamlessly with IAM to authenticate your IAM users. Connect to the DB cluster and issue the `CREATE USER` statement, as shown in the following example.

```
CREATE USER jane_doe IDENTIFIED WITH AWSAuthenticationPlugin AS 'RDS';
```

The `IDENTIFIED WITH` clause allows MariaDB or MySQL to use the `AWSAuthenticationPlugin` to authenticate the database account (`jane_doe`). The `AS 'RDS'` clause refers to the authentication method. Make sure the specified database user name is the same as a resource in the IAM policy for IAM database access. For more information, see [Creating and using an IAM policy for IAM database access \(p. 1746\)](#).

Note

If you see the following message, it means that the AWS-provided plugin is not available for the current DB cluster.

`ERROR 1524 (HY000): Plugin 'AWSAuthenticationPlugin' is not loaded`
To troubleshoot this error, verify that you are using a supported configuration and that you have enabled IAM database authentication on your DB cluster. For more information, see [Availability for IAM database authentication \(p. 1743\)](#) and [Enabling and disabling IAM database authentication \(p. 1744\)](#).

After you create an account using `AWSAuthenticationPlugin`, you manage it in the same way as other database accounts. For example, you can modify account privileges with `GRANT` and `REVOKE` statements, or modify various account attributes with the `ALTER USER` statement.

Using IAM authentication with PostgreSQL

To use IAM authentication with PostgreSQL, connect to the DB cluster, create database users, and then grant them the `rds_iam` role as shown in the following example.

```
CREATE USER db_userx;
GRANT rds_iam TO db_userx;
```

Make sure the specified database user name is the same as a resource in the IAM policy for IAM database access. For more information, see [Creating and using an IAM policy for IAM database access \(p. 1746\)](#).

Note that a PostgreSQL database user can use either IAM or Kerberos authentication but not both, so this user can't also have the `rds_ad` role. This also applies to nested memberships. For more information, see [Step 7: Create Kerberos authentication PostgreSQL logins \(p. 1544\)](#).

Connecting to your DB cluster using IAM authentication

With IAM database authentication, you use an authentication token when you connect to your DB cluster. An *authentication token* is a string of characters that you use instead of a password. After you generate an authentication token, it's valid for 15 minutes before it expires. If you try to connect using an expired token, the connection request is denied.

Every authentication token must be accompanied by a valid signature, using AWS signature version 4. (For more information, see [Signature Version 4 signing process in the AWS General Reference](#).) The AWS CLI and an AWS SDK, such as the AWS SDK for Java or AWS SDK for Python (Boto3), can automatically sign each token you create.

You can use an authentication token when you connect to Amazon Aurora from another AWS service, such as AWS Lambda. By using a token, you can avoid placing a password in your code. Alternatively, you can use an AWS SDK to programmatically create and programmatically sign an authentication token.

After you have a signed IAM authentication token, you can connect to an Aurora DB cluster. Following, you can find out how to do this using either a command line tool or an AWS SDK, such as the AWS SDK for Java or AWS SDK for Python (Boto3).

For more information, see the following blog posts:

- [Use IAM authentication to connect with SQL Workbench/J to Aurora MySQL or Amazon RDS for MySQL](#)
- [Using IAM authentication to connect with pgAdmin Amazon Aurora PostgreSQL or Amazon RDS for PostgreSQL](#)

The following are prerequisites for connecting to your DB cluster using IAM authentication:

- [Enabling and disabling IAM database authentication \(p. 1744\)](#)
- [Creating and using an IAM policy for IAM database access \(p. 1746\)](#)
- [Creating a database account using IAM authentication \(p. 1749\)](#)

Topics

- [Connecting to your DB cluster using IAM authentication from the command line: AWS CLI and mysql client \(p. 1751\)](#)
- [Connecting to your DB cluster using IAM authentication from the command line: AWS CLI and psql client \(p. 1752\)](#)
- [Connecting to your DB cluster using IAM authentication and the AWS SDK for .NET \(p. 1754\)](#)
- [Connecting to your DB cluster using IAM authentication and the AWS SDK for Go \(p. 1756\)](#)

- [Connecting to your DB cluster using IAM authentication and the AWS SDK for Java \(p. 1760\)](#)
- [Connecting to your DB cluster using IAM authentication and the AWS SDK for Python \(Boto3\) \(p. 1767\)](#)

Connecting to your DB cluster using IAM authentication from the command line: AWS CLI and mysql client

You can connect from the command line to an Aurora DB cluster with the AWS CLI and `mysql` command line tool as described following.

The following are prerequisites for connecting to your DB cluster using IAM authentication:

- [Enabling and disabling IAM database authentication \(p. 1744\)](#)
- [Creating and using an IAM policy for IAM database access \(p. 1746\)](#)
- [Creating a database account using IAM authentication \(p. 1749\)](#)

Note

For information about connecting to your database using SQL Workbench/J with IAM authentication, see the blog post [Use IAM authentication to connect with SQL Workbench/J to Aurora MySQL or Amazon RDS for MySQL](#).

Topics

- [Generating an IAM authentication token \(p. 1751\)](#)
- [Connecting to a DB cluster \(p. 1751\)](#)

Generating an IAM authentication token

The following example shows how to get a signed authentication token using the AWS CLI.

```
aws rds generate-db-auth-token \
--hostname rdsmysql.123456789012.us-west-2.rds.amazonaws.com \
--port 3306 \
--region us-west-2 \
--username jane_doe
```

In the example, the parameters are as follows:

- `--hostname` – The host name of the DB cluster that you want to access
- `--port` – The port number used for connecting to your DB cluster
- `--region` – The AWS Region where the DB cluster is running
- `--username` – The database account that you want to access

The first several characters of the token look like the following.

```
rdsmysql.123456789012.us-west-2.rds.amazonaws.com:3306/?Action=connect&DBUser=jane_doe&X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Expires=900...
```

Connecting to a DB cluster

The general format for connecting is shown following.

```
mysql --host=hostName --port=portNumber --ssl-ca=full_path_to_ssl_certificate --enable-cleartext-plugin --user=userName --password=authToken
```

The parameters are as follows:

- `--host` – The host name of the DB cluster that you want to access
- `--port` – The port number used for connecting to your DB cluster
- `--ssl-ca` – The full path to the SSL certificate file that contains the public key

For more information, see [Using SSL/TLS with Aurora MySQL DB clusters \(p. 775\)](#).

To download an SSL certificate, see [Using SSL/TLS to encrypt a connection to a DB cluster \(p. 1714\)](#).

- `--enable-cleartext-plugin` – A value that specifies that `AWSAuthenticationPlugin` must be used for this connection

If you are using a MariaDB client, the `--enable-cleartext-plugin` option isn't required.

- `--user` – The database account that you want to access
- `--password` – A signed IAM authentication token

The authentication token consists of several hundred characters. It can be unwieldy on the command line. One way to work around this is to save the token to an environment variable, and then use that variable when you connect. The following example shows one way to perform this workaround. In the example, `/sample_dir/` is the full path to the SSL certificate file that contains the public key.

```
RDSHOST="mysqlcluster.cluster-123456789012.us-east-1.rds.amazonaws.com"  
TOKEN="$(aws rds generate-db-auth-token --hostname $RDSHOST --port 3306 --region us-west-2  
--username jane_doe )"  
  
mysql --host=$RDSHOST --port=3306 --ssl-ca=/sample_dir/global-bundle.pem --enable-  
cleartext-plugin --user=jane_doe --password=$TOKEN
```

When you connect using `AWSAuthenticationPlugin`, the connection is secured using SSL. To verify this, type the following at the `mysql>` command prompt.

```
show status like 'Ssl%';
```

The following lines in the output show more details.

```
+-----+-----+  
| Variable_name | Value  
+-----+-----+  
| ... | ...  
| Ssl_cipher | AES256-SHA  
| ... | ...  
| Ssl_version | TLSv1.1  
| ... | ...  
+-----+
```

Connecting to your DB cluster using IAM authentication from the command line: AWS CLI and psql client

You can connect from the command line to an Aurora PostgreSQL DB cluster with the AWS CLI and `psql` command line tool as described following.

The following are prerequisites for connecting to your DB cluster using IAM authentication:

- [Enabling and disabling IAM database authentication \(p. 1744\)](#)
- [Creating and using an IAM policy for IAM database access \(p. 1746\)](#)
- [Creating a database account using IAM authentication \(p. 1749\)](#)

Note

For information about connecting to your database using pgAdmin with IAM authentication, see the blog post [Using IAM authentication to connect with pgAdmin Amazon Aurora PostgreSQL or Amazon RDS for PostgreSQL](#).

Topics

- [Generating an IAM authentication token \(p. 1753\)](#)
- [Connecting to an Aurora PostgreSQL cluster \(p. 1753\)](#)

Generating an IAM authentication token

The authentication token consists of several hundred characters so it can be unwieldy on the command line. One way to work around this is to save the token to an environment variable, and then use that variable when you connect. The following example shows how to use the AWS CLI to get a signed authentication token using the `generate-db-auth-token` command, and store it in a `PGPASSWORD` environment variable.

```
export RDSHOST="mypostgres-cluster.cluster-123456789012.us-west-2.rds.amazonaws.com"
export PGPASSWORD=$(aws rds generate-db-auth-token --hostname $RDSHOST --port 5432 --
region us-west-2 --username jane_doe )"
```

In the example, the parameters to the `generate-db-auth-token` command are as follows:

- `--hostname` – The host name of the DB cluster (cluster endpoint) that you want to access
- `--port` – The port number used for connecting to your DB cluster
- `--region` – The AWS Region where the DB cluster is running
- `--username` – The database account that you want to access

The first several characters of the generated token look like the following.

```
mypostgres-cluster.cluster-123456789012.us-west-2.rds.amazonaws.com:5432/?Action=connect&DBUser=jane_doe&X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Expires=900...
```

Connecting to an Aurora PostgreSQL cluster

The general format for using `psql` to connect is shown following.

```
psql "host=hostName port=portNumber sslmode=verify-full
sslrootcert=full_path_to_ssl_certificate dbname=DBName user=userName password=authToken"
```

The parameters are as follows:

- `host` – The host name of the DB cluster (cluster endpoint) that you want to access
- `port` – The port number used for connecting to your DB cluster
- `sslmode` – The SSL mode to use

When you use `sslmode=verify-full`, the SSL connection verifies the DB cluster endpoint against the endpoint in the SSL certificate.

- `sslrootcert` – The full path to the SSL certificate file that contains the public key

For more information, see [Securing Aurora PostgreSQL data with SSL/TLS \(p. 1277\)](#).

To download an SSL certificate, see [Using SSL/TLS to encrypt a connection to a DB cluster \(p. 1714\)](#).

- `dbname` – The database that you want to access
- `user` – The database account that you want to access
- `password` – A signed IAM authentication token

The following example shows using `psql` to connect. In the example, `psql` uses the environment variable `RDSHOST` for the host and the environment variable `PGPASSWORD` for the generated token. Also, `/sample_dir/` is the full path to the SSL certificate file that contains the public key.

```
export RDSHOST="mypostgres-cluster.cluster-123456789012.us-west-2.rds.amazonaws.com"
export PGPASSWORD=$(aws rds generate-db-auth-token --hostname $RDSHOST --port 5432 --
region us-west-2 --username jane_doe )"

psql "host=$RDSHOST port=5432 sslmode=verify-full sslrootcert=/sample_dir/global-bundle.pem
dbname=DBName user=jane_doe password=$PGPASSWORD"
```

Connecting to your DB cluster using IAM authentication and the AWS SDK for .NET

You can connect to an Aurora MySQL or Aurora PostgreSQL DB cluster with the AWS SDK for .NET as described following.

The following are prerequisites for connecting to your DB cluster using IAM authentication:

- [Enabling and disabling IAM database authentication \(p. 1744\)](#)
- [Creating and using an IAM policy for IAM database access \(p. 1746\)](#)
- [Creating a database account using IAM authentication \(p. 1749\)](#)

The following code examples show how to generate an authentication token, and then use it to connect to a DB cluster.

To run this code example, you need the [AWS SDK for .NET](#), found on the AWS site. The `AWSSDK.Core` and the `AWSSDK.RDS` packages are required. To connect to a DB instance, use the .NET database connector for the DB engine, such as `MySQLConnector` for MariaDB or MySQL, or `Npgsql` for PostgreSQL.

This code connects to an Aurora MySQL DB cluster.

Modify the values of the following variables as needed:

- `server` – The endpoint of the DB cluster that you want to access
- `user` – The database account that you want to access
- `password` – The password for the specified user
- `database` – The database that you want to access
- `port` – The port number used for connecting to your DB cluster
- `SslMode` – The SSL mode to use

When you use `SslMode=Required`, the SSL connection verifies the DB cluster endpoint against the endpoint in the SSL certificate.

- **SslCa** – The full path to the SSL certificate for Amazon Aurora

To download a certificate, see [Using SSL/TLS to encrypt a connection to a DB cluster \(p. 1714\)](#).

```
using System;
using System.Data;
using MySql.Data;
using MySql.Data.MySqlClient;
using Amazon;

namespace ubuntu
{
    class Program
    {
        static void Main(string[] args)
        {
            var pwd =
Amazon.RDS.Util.RDSS AuthTokenGenerator.GenerateAuthToken(RegionEndpoint.USEast1,
"mysqlcluster.cluster-123456789012.us-east-1.rds.amazonaws.com", 3306, "jane_doe");
            // for debug only Console.WriteLine("{0}\n", pwd); //this verifies the token is
generated

            MySqlConnection conn = new
MySqlConnection($"server=mysqlcluster.cluster-123456789012.us-
east-1.rds.amazonaws.com;user=jane_doe;database=mydB;port=3306;password=password;SslMode=Required;SslCa
conn.Open();

            // Define a query
            MySqlCommand sampleCommand = new MySqlCommand("SHOW DATABASES;", conn);

            // Execute a query
            MySqlDataReader mysqlDataRdr = sampleCommand.ExecuteReader();

            // Read all rows and output the first column in each row
            while (mysqlDataRdr.Read())
                Console.WriteLine(mysqlDataRdr[0]);

            mysqlDataRdr.Close();
            // Close connection
            conn.Close();
        }
    }
}
```

This code connects to an Aurora PostgreSQL DB cluster.

Modify the values of the following variables as needed:

- **Server** – The endpoint of the DB cluster that you want to access
- **User ID** – The database account that you want to access
- **Password** – The password for the specified user
- **Database** – The database that you want to access
- **Port** – The port number used for connecting to your DB cluster
- **SSL Mode** – The SSL mode to use

When you use `SSL Mode=Required`, the SSL connection verifies the DB cluster endpoint against the endpoint in the SSL certificate.

- **SSL Certificate** – The full path to the SSL certificate for Amazon Aurora

To download a certificate, see [Using SSL/TLS to encrypt a connection to a DB cluster \(p. 1714\)](#).

```
using System;
using Npgsql;
using Amazon.RDS.Util;

namespace ConsoleApp1
{
    class Program
    {
        static void Main(string[] args)
        {
            var pwd =
RDSAuthTokenGenerator.GenerateAuthToken("postgresmycluster.cluster-123456789012.us-
east-1.rds.amazonaws.com", 5432, "jane_doe");
// for debug only Console.WriteLine("{0}\n", pwd); //this verifies the token is generated

            NpgsqlConnection conn = new
NpgsqlConnection($"Server=postgresmycluster.cluster-123456789012.us-
east-1.rds.amazonaws.com;User Id=jane_doe;Password=password;Database=mydb;SSL
Mode=Require;SSL Certificate=full_path_to_ssl_certificate");
            conn.Open();

            // Define a query
            NpgsqlCommand cmd = new NpgsqlCommand("select count(*) FROM pg_user",
conn);

            // Execute a query
            NpgsqlDataReader dr = cmd.ExecuteReader();

            // Read all rows and output the first column in each row
            while (dr.Read())
                Console.WriteLine("{0}\n", dr[0]);

            // Close connection
            conn.Close();
        }
    }
}
```

Connecting to your DB cluster using IAM authentication and the AWS SDK for Go

You can connect to an Aurora MySQL or Aurora PostgreSQL DB cluster with the AWS SDK for Go as described following.

The following are prerequisites for connecting to your DB cluster using IAM authentication:

- [Enabling and disabling IAM database authentication \(p. 1744\)](#)
- [Creating and using an IAM policy for IAM database access \(p. 1746\)](#)
- [Creating a database account using IAM authentication \(p. 1749\)](#)

To run these code examples, you need the [AWS SDK for Go](#), found on the AWS site.

Modify the values of the following variables as needed:

- `dbName` – The database that you want to access
- `dbUser` – The database account that you want to access
- `dbHost` – The endpoint of the DB cluster that you want to access
- `dbPort` – The port number used for connecting to your DB cluster
- `region` – The AWS Region where the DB cluster is running

In addition, make sure the imported libraries in the sample code exist on your system.

Important

The examples in this section use the following code to provide credentials that access a database from a local environment:

```
creds := credentials.NewEnvCredentials()
```

If you are accessing a database from an AWS service, such as Amazon EC2 or Amazon ECS, you can replace the code with the following code:

```
sess := session.Must(session.NewSession())
```

```
creds := sess.Config.Credentials
```

If you make this change, make sure you add the following import:

```
"github.com/aws/aws-sdk-go/aws/session"
```

Topics

- [Connecting using IAM authentication and the AWS SDK for Go V2 \(p. 1757\)](#)
- [Connecting using IAM authentication and the AWS SDK for Go V1. \(p. 1758\)](#)

Connecting using IAM authentication and the AWS SDK for Go V2

You can connect to a DB cluster using IAM authentication and the AWS SDK for Go V2.

The following code examples show how to generate an authentication token, and then use it to connect to a DB cluster.

This code connects to an Aurora MySQL DB cluster.

```
package main

import (
    "context"
    "database/sql"
    "fmt"

    "github.com/aws/aws-sdk-go-v2/config"
    "github.com/aws/aws-sdk-go-v2/feature/rds/auth"
    _ "github.com/lib/pq"
)

func main() {

    var dbName string = "DatabaseName"
    var dbUser string = "DatabaseUser"
    var dbHost string = "mysqlcluster.cluster-123456789012.us-east-1.rds.amazonaws.com"
    var dbPort int = 3306
    var dbEndpoint string = fmt.Sprintf("%s:%d", dbHost, dbPort)
    var region string = "us-east-1"

    cfg, err := config.LoadDefaultConfig(context.TODO())
    if err != nil {
        panic("configuration error: " + err.Error())
    }

    authenticationToken, err := auth.BuildAuthToken(
        context.TODO(), dbEndpoint, region, dbUser, cfg.Credentials)
    if err != nil {
        panic("failed to create authentication token: " + err.Error())
    }

    dsn := fmt.Sprintf("%s:%s@tcp(%s)/%s?tls=true&allowCleartextPasswords=true",
        dbUser, authenticationToken, dbEndpoint, dbName,
    )
}
```

```

db, err := sql.Open("mysql", dsn)
if err != nil {
    panic(err)
}

err = db.Ping()
if err != nil {
    panic(err)
}
}

```

This code connects to an Aurora PostgreSQL DB cluster.

```

package main

import (
    "context"
    "database/sql"
    "fmt"

    "github.com/aws/aws-sdk-go-v2/config"
    "github.com/aws/aws-sdk-go-v2/feature/rds/auth"
    _ "github.com/lib/pq"
)

func main() {

    var dbName string = "DatabaseName"
    var dbUser string = "DatabaseUser"
    var dbHost string = "postgresmycluster.cluster-123456789012.us-
east-1.rds.amazonaws.com"
    var dbPort int = 5432
    var dbEndpoint string = fmt.Sprintf("%s:%d", dbHost, dbPort)
    var region string = "us-east-1"

    cfg, err := config.LoadDefaultConfig(context.TODO())
    if err != nil {
        panic("configuration error: " + err.Error())
    }

    authenticationToken, err := auth.BuildAuthToken(
        context.TODO(), dbEndpoint, region, dbUser, cfg.Credentials)
    if err != nil {
        panic("failed to create authentication token: " + err.Error())
    }

    dsn := fmt.Sprintf("host=%s port=%d user=%s password=%s dbname=%s",
        dbHost, dbPort, dbUser, authenticationToken, dbName,
    )

    db, err := sql.Open("postgres", dsn)
    if err != nil {
        panic(err)
    }

    err = db.Ping()
    if err != nil {
        panic(err)
    }
}

```

Connecting using IAM authentication and the AWS SDK for Go V1.

You can connect to a DB cluster using IAM authentication and the AWS SDK for Go V1

The following code examples show how to generate an authentication token, and then use it to connect to a DB cluster.

This code connects to an Aurora MySQL DB cluster.

```
package main

import (
    "database/sql"
    "fmt"
    "log"

    "github.com/aws/aws-sdk-go/aws/credentials"
    "github.com/aws/aws-sdk-go/service/rds/rdsutils"
    _ "github.com/go-sql-driver/mysql"
)

func main() {
    dbName := "app"
    dbUser := "jane_doe"
    dbHost := "mysqlcluster.cluster-123456789012.us-east-1.rds.amazonaws.com"
    dbPort := 3306
    dbEndpoint := fmt.Sprintf("%s:%d", dbHost, dbPort)
    region := "us-east-1"

    creds := credentials.NewEnvCredentials()
    authToken, err := rdsutils.BuildAuthToken(dbEndpoint, region, dbUser, creds)
    if err != nil {
        panic(err)
    }

    dsn := fmt.Sprintf("%s:%s@tcp(%s)/%s?tls=true&allowCleartextPasswords=true",
        dbUser, authToken, dbEndpoint, dbName,
    )

    db, err := sql.Open("mysql", dsn)
    if err != nil {
        panic(err)
    }

    err = db.Ping()
    if err != nil {
        panic(err)
    }
}
```

This code connects to an Aurora PostgreSQL DB cluster.

```
package main

import (
    "database/sql"
    "fmt"

    "github.com/aws/aws-sdk-go/aws/credentials"
    "github.com/aws/aws-sdk-go/service/rds/rdsutils"
    _ "github.com/lib/pq"
)

func main() {
    dbName := "app"
    dbUser := "jane_doe"
    dbHost := "postgresmycluster.cluster-123456789012.us-east-1.rds.amazonaws.com"
    dbPort := 5432
```

```
dbEndpoint := fmt.Sprintf("%s:%d", dbHost, dbPort)
region := "us-east-1"

creds := credentials.NewEnvCredentials()
authToken, err := rdsutils.BuildAuthToken(dbEndpoint, region, dbUser, creds)
if err != nil {
    panic(err)
}

dsn := fmt.Sprintf("host=%s port=%d user=%s password=%s dbname=%s",
    dbHost, dbPort, dbUser, authToken, dbName,
)

db, err := sql.Open("postgres", dsn)
if err != nil {
    panic(err)
}

err = db.Ping()
if err != nil {
    panic(err)
}
}
```

Connecting to your DB cluster using IAM authentication and the AWS SDK for Java

You can connect to an Aurora MySQL or Aurora PostgreSQL DB cluster with the AWS SDK for Java as described following.

The following are prerequisites for connecting to your DB cluster using IAM authentication:

- [Enabling and disabling IAM database authentication \(p. 1744\)](#)
- [Creating and using an IAM policy for IAM database access \(p. 1746\)](#)
- [Creating a database account using IAM authentication \(p. 1749\)](#)
- [Set up the AWS SDK for Java](#)

Topics

- [Generating an IAM authentication token \(p. 1760\)](#)
- [Manually constructing an IAM authentication token \(p. 1761\)](#)
- [Connecting to a DB cluster \(p. 1764\)](#)

Generating an IAM authentication token

If you are writing programs using the AWS SDK for Java, you can get a signed authentication token using the `RdsIamAuthTokenGenerator` class. Using this class requires that you provide AWS credentials. To do this, you create an instance of the `DefaultAWSCredentialsProviderChain` class. `DefaultAWSCredentialsProviderChain` uses the first AWS access key and secret key that it finds in the [default credential provider chain](#). For more information about AWS access keys, see [Managing access keys for IAM users](#).

After you create an instance of `RdsIamAuthTokenGenerator`, you can call the `getAuthToken` method to obtain a signed token. Provide the AWS Region, host name, port number, and user name. The following code example illustrates how to do this.

```
package com.amazonaws.codesamples;
```

```

import com.amazonaws.auth.DefaultAWSCredentialsProviderChain;
import com.amazonaws.services.rds.auth.GetIamAuthTokenRequest;
import com.amazonaws.services.rds.auth.RdsIamAuthTokenGenerator;

public class GenerateRDSAuthToken {

    public static void main(String[] args) {

        String region = "us-west-2";
        String hostname = "rdsmysql.123456789012.us-west-2.rds.amazonaws.com";
        String port = "3306";
        String username = "jane_doe";

        System.out.println(generateAuthToken(region, hostname, port, username));
    }

    static String generateAuthToken(String region, String hostName, String port, String
username) {

        RdsIamAuthTokenGenerator generator = RdsIamAuthTokenGenerator.builder()
            .credentials(new DefaultAWSCredentialsProviderChain())
            .region(region)
            .build();

        String authToken = generator.getAuthToken(
            GetIamAuthTokenRequest.builder()
                .hostname(hostName)
                .port(Integer.parseInt(port))
                .userName(username)
                .build());

        return authToken;
    }
}

```

Manually constructing an IAM authentication token

In Java, the easiest way to generate an authentication token is to use `RdsIamAuthTokenGenerator`. This class creates an authentication token for you, and then signs it using AWS signature version 4. For more information, see [Signature version 4 signing process](#) in the *AWS General Reference*.

However, you can also construct and sign an authentication token manually, as shown in the following code example.

```

package com.amazonaws.codesamples;

import com.amazonaws.SdkClientException;
import com.amazonaws.auth.DefaultAWSCredentialsProviderChain;
import com.amazonaws.auth.SigningAlgorithm;
import com.amazonaws.util.BinaryUtils;
import org.apache.commons.lang3.StringUtils;

import javax.crypto.Mac;
import javax.crypto.spec.SecretKeySpec;
import java.nio.charset.Charset;
import java.security.MessageDigest;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.SortedMap;
import java.util.TreeMap;

import static com.amazonaws.auth.internal.SignerConstants.AWS4_TERMINATOR;

```

```

import static com.amazonaws.util.StringUtils.UTF8;

public class CreateRDSAuthTokenManually {
    public static String httpMethod = "GET";
    public static String action = "connect";
    public static String canonicalURIParameter = "/";
    public static SortedMap<String, String> canonicalQueryParameters = new TreeMap();
    public static String payload = StringUtils.EMPTY;
    public static String signedHeader = "host";
    public static String algorithm = "AWS4-HMAC-SHA256";
    public static String serviceName = "rds-db";
    public static String requestWithoutSignature;

    public static void main(String[] args) throws Exception {

        String region = "us-west-2";
        String instanceName = "rdsmysql.123456789012.us-west-2.rds.amazonaws.com";
        String port = "3306";
        String username = "jane_doe";

        Date now = new Date();
        String date = new SimpleDateFormat("yyyyMMdd").format(now);
        String dateTimeStamp = new SimpleDateFormat("yyyyMMdd'T'HHmmss'Z'").format(now);
        DefaultAWSCredentialsProviderChain creds = new
DefaultAWSCredentialsProviderChain();
        String awsAccessKey = creds.getCredentials().getAWSAccessKeyId();
        String awsSecretKey = creds.getCredentials().getAWSSecretKey();
        String expiryMinutes = "900";

        System.out.println("Step 1: Create a canonical request:");
        String canonicalString = createCanonicalString(username, awsAccessKey, date,
dateTimeStamp, region, expiryMinutes, instanceName, port);
        System.out.println(canonicalString);
        System.out.println();

        System.out.println("Step 2: Create a string to sign:");
        String stringToSign = createStringToSign(dateTimeStamp, canonicalString,
awsAccessKey, date, region);
        System.out.println(stringToSign);
        System.out.println();

        System.out.println("Step 3: Calculate the signature:");
        String signature = BinaryUtils.toHex(calculateSignature(stringToSign,
new SigningKey(awsSecretKey, date, region, serviceName)));
        System.out.println(signature);
        System.out.println();

        System.out.println("Step 4: Add the signing info to the request");
        System.out.println(appendSignature(signature));
        System.out.println();

    }

    //Step 1: Create a canonical request date should be in format YYYYMMDD and dateTime
    //should be in format YYYYMMDDTHHMMSSZ
    public static String createCanonicalString(String user, String accessKey, String date,
String dateTime, String region, String expiryPeriod, String hostName, String port) throws
Exception {
        canonicalQueryParameters.put("Action", action);
        canonicalQueryParameters.put("DBUser", user);
        canonicalQueryParameters.put("X-Amz-Algorithm", "AWS4-HMAC-SHA256");
        canonicalQueryParameters.put("X-Amz-Credential", accessKey + "%2F" + date + "%2F" +
region + "%2F" + serviceName + "%2Faws4_request");
        canonicalQueryParameters.put("X-Amz-Date", dateTime);
        canonicalQueryParameters.put("X-Amz-Expires", expiryPeriod);
        canonicalQueryParameters.put("X-Amz-SignedHeaders", signedHeader);
    }
}

```

```

        String canonicalQueryString = "";
        while(!canonicalQueryParameters.isEmpty()) {
            String currentQueryParameter = canonicalQueryParameters.firstKey();
            String currentQueryParameterValue =
canonicalQueryParameters.remove(currentQueryParameter);
            canonicalQueryString = canonicalQueryString + currentQueryParameter + "=" +
currentQueryParameterValue;
            if (!currentQueryParameter.equals("X-Amz-SignedHeaders")) {
                canonicalQueryString += "&";
            }
        }
        String canonicalHeaders = "host:" + hostName + ":" + port + '\n';
        requestWithoutSignature = hostName + ":" + port + "/" + canonicalQueryString;

        String hashedPayload = BinaryUtils.toHex(hash(payload));
        return httpMethod + '\n' + canonicalURIParameter + '\n' + canonicalQueryString +
'\n' + canonicalHeaders + '\n' + signedHeader + '\n' + hashedPayload;

    }

    //Step 2: Create a string to sign using sig v4
    public static String createStringToSign(String date, String canonicalRequest,
String accessKey, String date, String region) throws Exception {
    String credentialScope = date + "/" + region + "/" + serviceName + "/aws4_request";
    return algorithm + '\n' + date + '\n' + canonicalRequest + '\n' +
BinaryUtils.toHex(hash(canonicalRequest));

}

//Step 3: Calculate signature
/**
 * Step 3 of the &AWS; Signature version 4 calculation. It involves deriving
 * the signing key and computing the signature. Refer to
 * http://docs.aws.amazon
 * .com/general/latest/gr/sigv4-calculate-signature.html
 */
public static byte[] calculateSignature(String stringToSign,
                                         byte[] signingKey) {
    return sign(stringToSign.getBytes(Charset.forName("UTF-8")), signingKey,
               SigningAlgorithm.HmacSHA256);
}

public static byte[] sign(byte[] data, byte[] key,
                         SigningAlgorithm algorithm) throwsSdkClientException {
    try {
        Mac mac = algorithm.getMac();
        mac.init(new SecretKeySpec(key, algorithm.toString()));
        return mac.doFinal(data);
    } catch (Exception e) {
        throw new SdkClientException(
            "Unable to calculate a request signature: "
            + e.getMessage(), e);
    }
}

public static byte[] newSigningKey(String secretKey,
                                   String dateStamp, String regionName, String serviceName)
{
    byte[] kSecret = ("AWS4" + secretKey).getBytes(Charset.forName("UTF-8"));
    byte[] kDate = sign(dateStamp, kSecret, SigningAlgorithm.HmacSHA256);
    byte[] kRegion = sign(regionName, kDate, SigningAlgorithm.HmacSHA256);
    byte[] kService = sign(serviceName, kRegion,
                           SigningAlgorithm.HmacSHA256);
    return sign(AWS4_TERMINATOR, kService, SigningAlgorithm.HmacSHA256);
}

```

```

public static byte[] sign(String stringData, byte[] key,
                         SigningAlgorithm algorithm) throws SdkClientException {
    try {
        byte[] data = stringData.getBytes(UTF8);
        return sign(data, key, algorithm);
    } catch (Exception e) {
        throw new SdkClientException(
            "Unable to calculate a request signature: "
            + e.getMessage(), e);
    }
}

//Step 4: append the signature
public static String appendSignature(String signature) {
    return requestWithoutSignature + "&X-Amz-Signature=" + signature;
}

public static byte[] hash(String s) throws Exception {
    try {
        MessageDigest md = MessageDigest.getInstance("SHA-256");
        md.update(s.getBytes(UTF8));
        return md.digest();
    } catch (Exception e) {
        throw new SdkClientException(
            "Unable to compute hash while signing request: "
            + e.getMessage(), e);
    }
}
}

```

Connecting to a DB cluster

The following code example shows how to generate an authentication token, and then use it to connect to a cluster running MariaDB or MySQL.

To run this code example, you need the [AWS SDK for Java](#), found on the AWS site. In addition, you need the following:

- MySQL Connector/J. This code example was tested with `mysql-connector-java-5.1.33-bin.jar`.
- An intermediate certificate for Amazon Aurora that is specific to an AWS Region. (For more information, see [Using SSL/TLS to encrypt a connection to a DB cluster \(p. 1714\)](#).) At runtime, the class loader looks for the certificate in the same directory as this Java code example, so that the class loader can find it.
- Modify the values of the following variables as needed:
 - `RDS_INSTANCE_HOSTNAME` – The host name of the DB cluster that you want to access.
 - `RDS_INSTANCE_PORT` – The port number used for connecting to your PostgreSQL DB cluster.
 - `REGION_NAME` – The AWS Region where the DB cluster is running.
 - `DB_USER` – The database account that you want to access.
 - `SSL_CERTIFICATE` – An SSL certificate for Amazon Aurora that is specific to an AWS Region.

To download a certificate for your AWS Region, see [Using SSL/TLS to encrypt a connection to a DB cluster \(p. 1714\)](#). Place the SSL certificate in the same directory as this Java program file, so that the class loader can find the certificate at runtime.

This code example obtains AWS credentials from the [default credential provider chain](#).

```
package com.amazonaws.samples;
```

```

import com.amazonaws.services.rds.auth.RdsIamAuthTokenGenerator;
import com.amazonaws.services.rds.auth.GetIamAuthTokenRequest;
import com.amazonaws.auth.BasicAWSCredentials;
import com.amazonaws.auth.DefaultAWSCredentialsProviderChain;
import com.amazonaws.auth.AWSStaticCredentialsProvider;

import java.io.File;
import java.io.FileOutputStream;
import java.io.InputStream;
import java.security.KeyStore;
import java.security.cert.CertificateFactory;
import java.security.cert.X509Certificate;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;
import java.util.Properties;

import java.net.URL;

public class IAMDatabaseAuthenticationTester {
    //AWS; Credentials of the IAM user with policy enabling IAM Database Authenticated
    access to the db by the db user.
    private static final DefaultAWSCredentialsProviderChain creds = new
DefaultAWSCredentialsProviderChain();
    private static final String AWS_ACCESS_KEY =
creds.getCredentials().getAWSAccessKeyId();
    private static final String AWS_SECRET_KEY = creds.getCredentials().getAWSSecretKey();

    //Configuration parameters for the generation of the IAM Database Authentication token
    private static final String RDS_INSTANCE_HOSTNAME = "rdsmysql.123456789012.us-
west-2.rds.amazonaws.com";
    private static final int RDS_INSTANCE_PORT = 3306;
    private static final String REGION_NAME = "us-west-2";
    private static final String DB_USER = "jane_doe";
    private static final String JDBC_URL = "jdbc:mysql://" + RDS_INSTANCE_HOSTNAME + ":" +
RDS_INSTANCE_PORT;

    private static final String SSL_CERTIFICATE = "rds-ca-2019-us-west-2.pem";

    private static final String KEY_STORE_TYPE = "JKS";
    private static final String KEY_STORE_PROVIDER = "SUN";
    private static final String KEY_STORE_FILE_PREFIX = "sys-connect-via-ssl-test-cacerts";
    private static final String KEY_STORE_FILE_SUFFIX = ".jks";
    private static final String DEFAULT_KEY_STORE_PASSWORD = "changeit";

    public static void main(String[] args) throws Exception {
        //get the connection
        Connection connection = getDBConnectionUsingIam();

        //verify the connection is successful
        Statement stmt= connection.createStatement();
        ResultSet rs=stmt.executeQuery("SELECT 'Success!' FROM DUAL;");
        while (rs.next()) {
            String id = rs.getString(1);
            System.out.println(id); //Should print "Success!"
        }

        //close the connection
        stmt.close();
        connection.close();

        clearSslProperties();
    }
}

```

```

/**
 * This method returns a connection to the db instance authenticated using IAM Database
Authentication
 * @return
 * @throws Exception
 */
private static Connection getDBConnectionUsingIam() throws Exception {
    setSslProperties();
    return DriverManager.getConnection(JDBC_URL, setMySqlConnectionProperties());
}

/**
 * This method sets the mysql connection properties which includes the IAM Database
Authentication token
 * as the password. It also specifies that SSL verification is required.
 * @return
 */
private static Properties setMySqlConnectionProperties() {
    Properties mysqlConnectionProperties = new Properties();
    mysqlConnectionProperties.setProperty("verifyServerCertificate","true");
    mysqlConnectionProperties.setProperty("useSSL", "true");
    mysqlConnectionProperties.setProperty("user",DB_USER);
    mysqlConnectionProperties.setProperty("password",generateAuthToken());
    return mysqlConnectionProperties;
}

/**
 * This method generates the IAM Auth Token.
 * An example IAM Auth Token would look like follows:
 * btus123.cmz7kenwo2ye.rds.cn-north-1.amazonaws.com.cn:3306/?Action=connect&DBUser=iamtestuser&X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-
Date=20171003T010726Z&X-Amz-SignedHeaders=host&X-Amz-Expires=899&X-Amz-
Credential=AKIAFPXHGVDI5RNFO4AQ%2F20171003%2Fcn-north-1%2Frds-db%2Faws4_request&X-Amz-
Signature=f9f45ef96c1f770cdad11a53e33ffa4c3730bc03fdee820cfdf1322eed15483b
 * @return
 */
private static String generateAuthToken() {
    BasicAWSCredentials awsCredentials = new BasicAWSCredentials(AWS_ACCESS_KEY,
AWS_SECRET_KEY);

    RdsIamAuthTokenGenerator generator = RdsIamAuthTokenGenerator.builder()
        .credentials(new
AWSStaticCredentialsProvider(awsCredentials)).region(REGION_NAME).build();
    return generator.getAuthToken(GetIamAuthTokenRequest.builder()

.hostname(RDS_INSTANCE_HOSTNAME).port(RDS_INSTANCE_PORT).userName(DB_USER).build());
}

/**
 * This method sets the SSL properties which specify the key store file, its type and
password:
 * @throws Exception
 */
private static void setSslProperties() throws Exception {
    System.setProperty("javax.net.ssl.trustStore", createKeyStoreFile());
    System.setProperty("javax.net.ssl.trustStoreType", KEY_STORE_TYPE);
    System.setProperty("javax.net.ssl.trustStorePassword", DEFAULT_KEY_STORE_PASSWORD);
}

/**
 * This method returns the path of the Key Store File needed for the SSL verification
during the IAM Database Authentication to
 * the db instance.
 * @return
 * @throws Exception

```

```

/*
private static String createKeyStoreFile() throws Exception {
    return createKeyStoreFile(createCertificate()).getPath();
}

/**
 * This method generates the SSL certificate
 * @return
 * @throws Exception
 */
private static X509Certificate createCertificate() throws Exception {
    CertificateFactory certFactory = CertificateFactory.getInstance("X.509");
    URL url = new File(SSL_CERTIFICATE).toURI().toURL();
    if (url == null) {
        throw new Exception();
    }
    try (InputStream certInputStream = url.openStream()) {
        return (X509Certificate) certFactory.generateCertificate(certInputStream);
    }
}

/**
 * This method creates the Key Store File
 * @param rootX509Certificate - the SSL certificate to be stored in the KeyStore
 * @return
 * @throws Exception
 */
private static File createKeyStoreFile(X509Certificate rootX509Certificate) throws
Exception {
    File keyStoreFile = File.createTempFile(KEY_STORE_FILE_PREFIX,
KEY_STORE_FILE_SUFFIX);
    try (FileOutputStream fos = new FileOutputStream(keyStoreFile.getPath())) {
        KeyStore ks = KeyStore.getInstance(KEY_STORE_TYPE, KEY_STORE_PROVIDER);
        ks.load(null);
        ks.setCertificateEntry("rootCaCertificate", rootX509Certificate);
        ks.store(fos, DEFAULT_KEY_STORE_PASSWORD.toCharArray());
    }
    return keyStoreFile;
}

/**
 * This method clears the SSL properties.
 * @throws Exception
 */
private static void clearSslProperties() throws Exception {
    System.clearProperty("javax.net.ssl.trustStore");
    System.clearProperty("javax.net.ssl.trustStoreType");
    System.clearProperty("javax.net.ssl.trustStorePassword");
}
}

```

Connecting to your DB cluster using IAM authentication and the AWS SDK for Python (Boto3)

You can connect to an Aurora MySQL or Aurora PostgreSQL DB cluster with the AWS SDK for Python (Boto3) as described following.

The following are prerequisites for connecting to your DB cluster using IAM authentication:

- [Enabling and disabling IAM database authentication \(p. 1744\)](#)
- [Creating and using an IAM policy for IAM database access \(p. 1746\)](#)
- [Creating a database account using IAM authentication \(p. 1749\)](#)

In addition, make sure the imported libraries in the sample code exist on your system.

The code examples use profiles for shared credentials. For information about specifying credentials, see [Credentials](#) in the AWS SDK for Python (Boto3) documentation.

The following code examples show how to generate an authentication token, and then use it to connect to a DB cluster.

To run this code example, you need the [AWS SDK for Python \(Boto3\)](#), found on the AWS site.

Modify the values of the following variables as needed:

- **ENDPOINT** – The endpoint of the DB cluster that you want to access
- **PORT** – The port number used for connecting to your DB cluster
- **USER** – The database account that you want to access.
- **REGION** – The AWS Region where the DB cluster is running
- **DBNAME** – The database that you want to access
- **SSLCERTIFICATE** – The full path to the SSL certificate for Amazon Aurora

To download a certificate, see [Using SSL/TLS to encrypt a connection to a DB cluster \(p. 1714\)](#).

For `ssl_ca`, specify an SSL certificate. To download an SSL certificate, see [Using SSL/TLS to encrypt a connection to a DB cluster \(p. 1714\)](#).

This code connects to an Aurora MySQL DB cluster.

Before running this code, install Connector/Python by following the instructions in [Connector/Python Installation](#) in the MySQL documentation.

```
import mysql.connector
import sys
import boto3
import os

ENDPOINT="mysqlcluster.cluster-123456789012.us-east-1.rds.amazonaws.com"
PORT="3306"
USER="jane_doe"
REGION="us-east-1"
DBNAME="mydb"
os.environ['LIBMYSQL_ENABLE_CLEARTEXT_PLUGIN'] = '1'

#gets the credentials from .aws/credentials
session = boto3.Session(profile_name='default')
client = session.client('rds')

token = client.generate_db_auth_token(DBHostname=ENDPOINT, Port=PORT, DBUsername=USER,
Region=REGION)

try:
    conn = mysql.connector.connect(host=ENDPOINT, user=USER, passwd=token, port=PORT,
database=DBNAME, ssl_ca='SSLCERTIFICATE')
    cur = conn.cursor()
    cur.execute("""SELECT now()""")
    query_results = cur.fetchall()
    print(query_results)
except Exception as e:
    print("Database connection failed due to {}".format(e))
```

This code connects to an Aurora PostgreSQL DB cluster.

Before running this code, install `psycopg2` by following the instructions in [Psycopg documentation](#).

```
import psycopg2
import sys
import boto3
import os

ENDPOINT="postgresmycluster.cluster-123456789012.us-east-1.rds.amazonaws.com"
PORT="5432"
USER="jane_doe"
REGION="us-east-1"
DBNAME="mydb"

#gets the credentials from .aws/credentials
session = boto3.Session(profile_name='RDSCreds')
client = session.client('rds')

token = client.generate_db_auth_token(DBHostname=ENDPOINT, Port=PORT, DBUsername=USER,
Region=REGION)

try:
    conn = psycopg2.connect(host=ENDPOINT, port=PORT, database=DBNAME, user=USER,
password=token, sslrootcert="SSLCERTIFICATE")
    cur = conn.cursor()
    cur.execute("""SELECT now()""")
    query_results = cur.fetchall()
    print(query_results)
except Exception as e:
    print("Database connection failed due to {}".format(e))
```

Troubleshooting Amazon Aurora identity and access

Use the following information to help you diagnose and fix common issues that you might encounter when working with Aurora and IAM.

Topics

- [I'm not authorized to perform an action in Aurora \(p. 1769\)](#)
- [I'm not authorized to perform iam:PassRole \(p. 1770\)](#)
- [I want to view my access keys \(p. 1770\)](#)
- [I'm an administrator and want to allow others to access Aurora \(p. 1770\)](#)
- [I want to allow people outside of my AWS account to access my Aurora resources \(p. 1770\)](#)

I'm not authorized to perform an action in Aurora

If the AWS Management Console tells you that you're not authorized to perform an action, then you must contact your administrator for assistance. Your administrator is the person that provided you with your user name and password.

The following example error occurs when the `mateojackson` IAM user tries to use the console to view details about a `widget` but does not have `rds:GetWidget` permissions.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
rds:GetWidget on resource: my-example-widget
```

In this case, Mateo asks his administrator to update his policies to allow him to access the `my-example-widget` resource using the `rds:GetWidget` action.

I'm not authorized to perform iam:PassRole

If you receive an error that you're not authorized to perform the `iam:PassRole` action, then you must contact your administrator for assistance. Your administrator is the person that provided you with your user name and password. Ask that person to update your policies to allow you to pass a role to Aurora.

Some AWS services allow you to pass an existing role to that service, instead of creating a new service role or service-linked role. To do this, you must have permissions to pass the role to the service.

The following example error occurs when an IAM user named `marymajor` tries to use the console to perform an action in Aurora. However, the action requires the service to have permissions granted by a service role. Mary does not have permissions to pass the role to the service.

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform: iam:PassRole
```

In this case, Mary asks her administrator to update her policies to allow her to perform the `iam:PassRole` action.

I want to view my access keys

After you create your IAM user access keys, you can view your access key ID at any time. However, you can't view your secret access key again. If you lose your secret key, you must create a new access key pair.

Access keys consist of two parts: an access key ID (for example, `AKIAIOSFODNN7EXAMPLE`) and a secret access key (for example, `wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY`). Like a user name and password, you must use both the access key ID and secret access key together to authenticate your requests. Manage your access keys as securely as you do your user name and password.

Important

Do not provide your access keys to a third party, even to help [find your canonical user ID](#). By doing this, you might give someone permanent access to your account.

When you create an access key pair, you are prompted to save the access key ID and secret access key in a secure location. The secret access key is available only at the time you create it. If you lose your secret access key, you must add new access keys to your IAM user. You can have a maximum of two access keys. If you already have two, you must delete one key pair before creating a new one. To view instructions, see [Managing access keys](#) in the *IAM User Guide*.

I'm an administrator and want to allow others to access Aurora

To enable others to access Aurora, you must create an IAM entity (user or role) for the person or application that needs access. They use the credentials for that entity to access AWS. You must then attach a policy to the entity that grants them the correct permissions in Aurora.

To get started right away, see [Creating your first IAM delegated user and group](#) in the *IAM User Guide*.

I want to allow people outside of my AWS account to access my Aurora resources

You can create a role that users in other accounts or people outside of your organization can use to access your resources. You can specify who is trusted to assume the role. For services that support resource-based policies or access control lists (ACLs), you can use those policies to grant people access to your resources.

To learn more, consult the following:

- To learn whether Aurora supports these features, see [How Amazon Aurora works with IAM \(p. 1727\)](#).
- To learn how to provide access to your resources across AWS accounts that you own, see [Providing access to an IAM user in another AWS account that you own](#) in the *IAM User Guide*.
- To learn how to provide access to your resources to third-party AWS accounts, see [Providing access to AWS accounts owned by third parties](#) in the *IAM User Guide*.
- To learn how to provide access through identity federation, see [Providing access to externally authenticated users \(identity federation\)](#) in the *IAM User Guide*.
- To learn the difference between using roles and resource-based policies for cross-account access, see [How IAM roles differ from resource-based policies](#) in the *IAM User Guide*.

Logging and monitoring in Amazon Aurora

Monitoring is an important part of maintaining the reliability, availability, and performance of Amazon Aurora and your AWS solutions. You should collect monitoring data from all of the parts of your AWS solution so that you can more easily debug a multi-point failure if one occurs. AWS provides several tools for monitoring your Amazon Aurora resources and responding to potential incidents:

Amazon CloudWatch Alarms

Using Amazon CloudWatch alarms, you watch a single metric over a time period that you specify. If the metric exceeds a given threshold, a notification is sent to an Amazon SNS topic or AWS Auto Scaling policy. CloudWatch alarms do not invoke actions because they are in a particular state. Rather the state must have changed and been maintained for a specified number of periods.

AWS CloudTrail Logs

CloudTrail provides a record of actions taken by a user, role, or an AWS service in Amazon Aurora. CloudTrail captures all API calls for Amazon Aurora as events, including calls from the console and from code calls to Amazon RDS API operations. Using the information collected by CloudTrail, you can determine the request that was made to Amazon Aurora, the IP address from which the request was made, who made the request, when it was made, and additional details. For more information, see [Monitoring Amazon Aurora API calls in AWS CloudTrail \(p. 710\)](#).

Enhanced Monitoring

Amazon Aurora provides metrics in real time for the operating system (OS) that your DB cluster runs on. You can view the metrics for your DB cluster using the console, or consume the Enhanced Monitoring JSON output from Amazon CloudWatch Logs in a monitoring system of your choice. For more information, see [Monitoring OS metrics with Enhanced Monitoring \(p. 626\)](#).

Amazon RDS Performance Insights

Performance Insights expands on existing Amazon Aurora monitoring features to illustrate your database's performance and help you analyze any issues that affect it. With the Performance Insights dashboard, you can visualize the database load and filter the load by waits, SQL statements, hosts, or users. For more information, see [Monitoring DB load with Performance Insights on Amazon Aurora \(p. 573\)](#).

Database Logs

You can view, download, and watch database logs using the AWS Management Console, AWS CLI, or RDS API. For more information, see [Monitoring Amazon Aurora log files \(p. 695\)](#).

Amazon Aurora Recommendations

Amazon Aurora provides automated recommendations for database resources. These recommendations provide best practice guidance by analyzing DB cluster configuration, usage, and performance data. For more information, see [Viewing Amazon Aurora recommendations \(p. 558\)](#).

Amazon Aurora Event Notification

Amazon Aurora uses the Amazon Simple Notification Service (Amazon SNS) to provide notification when an Amazon Aurora event occurs. These notifications can be in any notification form supported by Amazon SNS for an AWS Region, such as an email, a text message, or a call to an HTTP endpoint. For more information, see [Using Amazon RDS event notification \(p. 675\)](#).

AWS Trusted Advisor

Trusted Advisor draws upon best practices learned from serving hundreds of thousands of AWS customers. Trusted Advisor inspects your AWS environment and then makes recommendations when opportunities exist to save money, improve system availability and performance, or help close security gaps. All AWS customers have access to five Trusted Advisor checks. Customers with a Business or Enterprise support plan can view all Trusted Advisor checks.

Trusted Advisor has the following Amazon Aurora-related checks:

- Amazon Aurora Idle DB Instances
- Amazon Aurora Security Group Access Risk
- Amazon Aurora Backups
- Amazon Aurora Multi-AZ
- Aurora DB Instance Accessibility

For more information on these checks, see [Trusted Advisor best practices \(checks\)](#).

Database activity streams

Database activity streams can protect your databases from internal threats by controlling DBA access to the database activity streams. Thus, the collection, transmission, storage, and subsequent processing of the database activity stream is beyond the access of the DBAs that manage the database. Database activity streams can provide safeguards for your database and meet compliance and regulatory requirements. For more information, see [Monitoring Amazon Aurora with Database Activity Streams \(p. 714\)](#).

For more information about monitoring Aurora see [Monitoring metrics in an Amazon Aurora cluster \(p. 541\)](#).

Compliance validation for Amazon Aurora

Third-party auditors assess the security and compliance of Amazon Aurora as part of multiple AWS compliance programs. These include SOC, PCI, FedRAMP, HIPAA, and others.

For a list of AWS services in scope of specific compliance programs, see [AWS services in scope by compliance program](#). For general information, see [AWS compliance programs](#).

You can download third-party audit reports using AWS Artifact. For more information, see [Downloading reports in AWS Artifact](#).

Your compliance responsibility when using Amazon Aurora is determined by the sensitivity of your data, your organization's compliance objectives, and applicable laws and regulations. AWS provides the following resources to help with compliance:

- [Security and compliance quick start guides](#) – These deployment guides discuss architectural considerations and provide steps for deploying security- and compliance-focused baseline environments on AWS.
- [Architecting for HIPAA security and compliance whitepaper](#) – This whitepaper describes how companies can use AWS to create HIPAA-compliant applications.
- [AWS compliance resources](#) – This collection of workbooks and guides that might apply to your industry and location.
- [AWS Config](#) – This AWS service assesses how well your resource configurations comply with internal practices, industry guidelines, and regulations.
- [AWS Security Hub](#) – This AWS service provides a comprehensive view of your security state within AWS that helps you check your compliance with security industry standards and best practices.

Resilience in Amazon Aurora

The AWS global infrastructure is built around AWS Regions and Availability Zones. AWS Regions provide multiple physically separated and isolated Availability Zones, which are connected with low-latency, high-throughput, and highly redundant networking. With Availability Zones, you can design and operate applications and databases that automatically fail over between Availability Zones without interruption. Availability Zones are more highly available, fault tolerant, and scalable than traditional single or multiple data center infrastructures.

For more information about AWS Regions and Availability Zones, see [AWS global infrastructure](#).

In addition to the AWS global infrastructure, Aurora offers features to help support your data resiliency and backup needs.

Backup and restore

Aurora backs up your cluster volume automatically and retains restore data for the length of the *backup retention period*. Aurora backups are continuous and incremental so you can quickly restore to any point within the backup retention period. No performance impact or interruption of database service occurs as backup data is being written. You can specify a backup retention period, from 1 to 35 days, when you create or modify a DB cluster.

If you want to retain a backup beyond the backup retention period, you can also take a snapshot of the data in your cluster volume. Aurora retains incremental restore data for the entire backup retention period. Thus, you need to create a snapshot only for data that you want to retain beyond the backup retention period. You can create a new DB cluster from the snapshot.

You can recover your data by creating a new Aurora DB cluster from the backup data that Aurora retains, or from a DB cluster snapshot that you have saved. You can quickly create a new copy of a DB cluster from backup data to any point in time during your backup retention period. The continuous and incremental nature of Aurora backups during the backup retention period means you don't need to take frequent snapshots of your data to improve restore times.

For more information, see [Backing up and restoring an Amazon Aurora DB cluster \(p. 490\)](#).

Replication

Aurora Replicas are independent endpoints in an Aurora DB cluster, best used for scaling read operations and increasing availability. Up to 15 Aurora Replicas can be distributed across the Availability Zones that a DB cluster spans within an AWS Region. The DB cluster volume is made up of multiple copies of the data for the DB cluster. However, the data in the cluster volume is represented as a single, logical volume to the primary DB instance and to Aurora Replicas in the DB cluster. If the primary DB instance fails, an Aurora Replica is promoted to be the primary DB instance.

Aurora also supports replication options that are specific to Aurora MySQL and Aurora PostgreSQL.

For more information, see [Replication with Amazon Aurora \(p. 70\)](#).

Failover

Aurora stores copies of the data in a DB cluster across multiple Availability Zones in a single AWS Region. This storage occurs regardless of whether the DB instances in the DB cluster span multiple Availability Zones. When you create Aurora Replicas across Availability Zones, Aurora automatically provisions and maintains them synchronously. The primary DB instance is synchronously replicated across Availability Zones to Aurora Replicas to provide data redundancy, eliminate I/O freezes, and minimize latency spikes during system backups. Running a DB cluster with high availability can enhance availability during

planned system maintenance, and help protect your databases against failure and Availability Zone disruption.

For more information, see [High availability for Amazon Aurora \(p. 68\)](#).

Infrastructure security in Amazon Aurora

As a managed service, Amazon RDS is protected by the AWS global network security procedures that are described in the [Amazon Web Services: Overview of security processes](#) whitepaper.

You use AWS published API calls to access Amazon Aurora through the network. Clients must support Transport Layer Security (TLS) 1.0. We recommend TLS 1.2 or later. Clients must also support cipher suites with perfect forward secrecy (PFS) such as Ephemeral Diffie-Hellman (DHE) or Elliptic Curve Ephemeral Diffie-Hellman (ECDHE). Most modern systems such as Java 7 and later support these modes.

Additionally, requests must be signed by using an access key ID and a secret access key that is associated with an IAM principal. Or you can use the [AWS Security Token Service \(AWS STS\)](#) to generate temporary security credentials to sign requests.

In addition, Aurora offers features to help support infrastructure security.

Security groups

Security groups control the access that traffic has in and out of a DB instance. By default, network access is turned off to a DB instance. You can specify rules in a security group that allow access from an IP address range, port, or security group. After ingress rules are configured, the same rules apply to all DB instances that are associated with that security group.

For more information, see [Controlling access with security groups \(p. 1780\)](#).

Public accessibility

When you launch a DB instance inside a virtual private cloud (VPC) based on the Amazon VPC service, you can turn on or off public accessibility for that instance. To designate whether the DB instance that you create has a DNS name that resolves to a public IP address, you use the *Public accessibility* parameter. By using this parameter, you can designate whether there is public access to the DB instance. You can modify a DB instance to turn on or off public accessibility by modifying the *Public accessibility* parameter.

For more information, see [Hiding a DB instance in a VPC from the internet \(p. 1789\)](#).

Note

If your DB instance is in a VPC but isn't publicly accessible, you can also use an AWS Site-to-Site VPN connection or an AWS Direct Connect connection to access it from a private network. For more information, see [Internetwork traffic privacy \(p. 1723\)](#).

Amazon RDS API and interface VPC endpoints (AWS PrivateLink)

You can establish a private connection between your VPC and Amazon RDS API endpoints by creating an *interface VPC endpoint*. Interface endpoints are powered by [AWS PrivateLink](#).

AWS PrivateLink enables you to privately access Amazon RDS API operations without an internet gateway, NAT device, VPN connection, or AWS Direct Connect connection. Instances in your VPC don't need public IP addresses to communicate with Amazon RDS API endpoints to launch, modify, or terminate DB clusters. Your instances also don't need public IP addresses to use any of the available RDS API operations. Traffic between your VPC and Amazon RDS doesn't leave the Amazon network.

Each interface endpoint is represented by one or more elastic network interfaces in your subnets. For more information on elastic network interfaces, see [Elastic network interfaces](#) in the *Amazon EC2 User Guide*.

For more information about VPC endpoints, see [Interface VPC endpoints \(AWS PrivateLink\)](#) in the *Amazon VPC User Guide*. For more information about RDS API operations, see [Amazon RDS API Reference](#).

Considerations for VPC endpoints

Before you set up an interface VPC endpoint for Amazon RDS API endpoints, ensure that you review [Interface endpoint properties and limitations](#) in the *Amazon VPC User Guide*.

All RDS API operations relevant to managing Amazon Aurora resources are available from your VPC using AWS PrivateLink.

VPC endpoint policies are supported for RDS API endpoints. By default, full access to RDS API operations is allowed through the endpoint. For more information, see [Controlling access to services with VPC endpoints](#) in the *Amazon VPC User Guide*.

Availability

Amazon RDS API currently supports VPC endpoints in the following AWS Regions:

- US East (Ohio)
- US East (N. Virginia)
- US West (N. California)
- US West (Oregon)
- Africa (Cape Town)
- Asia Pacific (Hong Kong)
- Asia Pacific (Mumbai)
- Asia Pacific (Osaka)
- Asia Pacific (Seoul)
- Asia Pacific (Singapore)
- Asia Pacific (Sydney)
- Asia Pacific (Tokyo)
- Canada (Central)
- Europe (Frankfurt)
- Europe (Ireland)
- Europe (London)

- Europe (Paris)
- Europe (Stockholm)
- Europe (Milan)
- Middle East (Bahrain)
- South America (São Paulo)
- China (Beijing)
- China (Ningxia)
- AWS GovCloud (US-East)
- AWS GovCloud (US-West)

Creating an interface VPC endpoint for Amazon RDS API

You can create a VPC endpoint for the Amazon RDS API using either the Amazon VPC console or the AWS Command Line Interface (AWS CLI). For more information, see [Creating an interface endpoint](#) in the *Amazon VPC User Guide*.

Create a VPC endpoint for Amazon RDS API using the service name `com.amazonaws.region.rds`.

Excluding AWS Regions in China, if you enable private DNS for the endpoint, you can make API requests to Amazon RDS with the VPC endpoint using its default DNS name for the AWS Region, for example `rds.us-east-1.amazonaws.com`. For the China (Beijing) and China (Ningxia) AWS Regions, you can make API requests with the VPC endpoint using `rds-api.cn-north-1.amazonaws.com.cn` and `rds-api.cn-northwest-1.amazonaws.com.cn`, respectively.

For more information, see [Accessing a service through an interface endpoint](#) in the *Amazon VPC User Guide*.

Creating a VPC endpoint policy for Amazon RDS API

You can attach an endpoint policy to your VPC endpoint that controls access to Amazon RDS API. The policy specifies the following information:

- The principal that can perform actions.
- The actions that can be performed.
- The resources on which actions can be performed.

For more information, see [Controlling access to services with VPC endpoints](#) in the *Amazon VPC User Guide*.

Example: VPC endpoint policy for Amazon RDS API actions

The following is an example of an endpoint policy for Amazon RDS API. When attached to an endpoint, this policy grants access to the listed Amazon RDS API actions for all principals on all resources.

```
{  
    "Statement": [  
        {  
            "Principal": "*",  
            "Effect": "Allow",  
            "Action": [  
                "rds:CreateDBInstance",  
                "rds:ModifyDBInstance",  
            ]  
        }  
    ]  
}
```

```
        "rds:CreateDBSnapshot"
    ],
    "Resource": "*"
}
]
```

Example: VPC endpoint policy that denies all access from a specified AWS account

The following VPC endpoint policy denies AWS account 123456789012 all access to resources using the endpoint. The policy allows all actions from other accounts.

```
{
  "Statement": [
    {
      "Action": "*",
      "Effect": "Allow",
      "Resource": "*",
      "Principal": "*"
    },
    {
      "Action": "*",
      "Effect": "Deny",
      "Resource": "*",
      "Principal": {
        "AWS": [
          "123456789012"
        ]
      }
    }
  ]
}
```

Security best practices for Amazon Aurora

Use AWS Identity and Access Management (IAM) accounts to control access to Amazon RDS API operations, especially operations that create, modify, or delete Amazon Aurora resources. Such resources include DB clusters, security groups, and parameter groups. Also use IAM to control actions that perform common administrative actions such as backing up and restoring DB clusters.

- Create an individual IAM user for each person who manages Amazon Aurora resources, including yourself. Don't use AWS root credentials to manage Amazon Aurora resources.
- Grant each user the minimum set of permissions required to perform his or her duties.
- Use IAM groups to effectively manage permissions for multiple users.
- Rotate your IAM credentials regularly.
- Configure AWS Secrets Manager to automatically rotate the secrets for Amazon Aurora. For more information, see [Rotating your AWS Secrets Manager secrets](#) in the *AWS Secrets Manager User Guide*. You can also retrieve the credential from AWS Secrets Manager programmatically. For more information, see [Retrieving the secret value](#) in the *AWS Secrets Manager User Guide*.

For more information about Amazon Aurora security, see [Security in Amazon Aurora \(p. 1706\)](#). For more information about IAM, see [AWS Identity and Access Management](#). For information on IAM best practices, see [IAM best practices](#).

Use the AWS Management Console, the AWS CLI, or the RDS API to change the password for your master user. If you use another tool, such as a SQL client, to change the master user password, it might result in privileges being revoked for the user unintentionally.

Controlling access with security groups

Security groups control the access that traffic has in and out of a DB instance. Aurora supports VPC security groups.

VPC security groups

Each VPC security group rule enables a specific source to access a DB instance in a VPC that is associated with that VPC security group. The source can be a range of addresses (for example, 203.0.113.0/24), or another VPC security group. By specifying a VPC security group as the source, you allow incoming traffic from all instances (typically application servers) that use the source VPC security group. VPC security groups can have rules that govern both inbound and outbound traffic, though the outbound traffic rules typically do not apply to DB instances. Outbound traffic rules only apply if the DB instance acts as a client. You must use the [Amazon EC2 API](#) or the **Security Group** option on the VPC Console to create VPC security groups.

When you create rules for your VPC security group that allow access to the instances in your VPC, you must specify a port for each range of addresses that the rule allows access for. For example, if you want to enable SSH access to instances in the VPC, then you create a rule allowing access to TCP port 22 for the specified range of addresses.

You can configure multiple VPC security groups that allow access to different ports for different instances in your VPC. For example, you can create a VPC security group that allows access to TCP port 80 for web servers in your VPC. You can then create another VPC security group that allows access to TCP port 3306 for Aurora MySQL DB instances in your VPC.

Note

In an Aurora DB cluster, the VPC security group associated with the DB cluster is also associated with all of the DB instances in the DB cluster. If you change the VPC security group for the DB cluster or for a DB instance, the change is applied automatically to all of the DB instances in the DB cluster.

For more information on VPC security groups, see [Security groups](#) in the *Amazon Virtual Private Cloud User Guide*.

Note

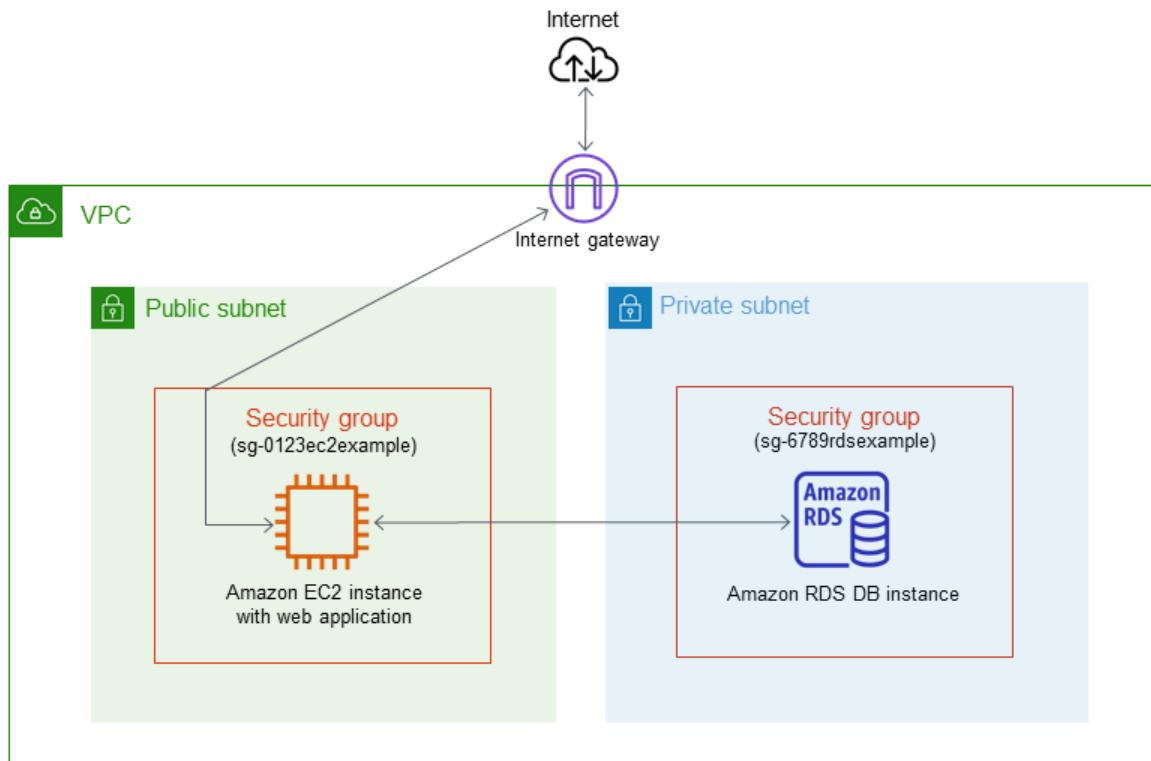
If your DB cluster is in a VPC but isn't publicly accessible, you can also use an AWS Site-to-Site VPN connection or an AWS Direct Connect connection to access it from a private network. For more information, see [Internetwork traffic privacy \(p. 1723\)](#).

Security group scenario

A common use of a DB instance in a VPC is to share data with an application server running in an Amazon EC2 instance in the same VPC, which is accessed by a client application outside the VPC. For this scenario, you use the RDS and VPC pages on the AWS Management Console or the RDS and EC2 API operations to create the necessary instances and security groups:

1. Create a VPC security group (for example, sg-0123ec2example) and define inbound rules that use the IP addresses of the client application as the source. This security group allows your client application to connect to EC2 instances in a VPC that uses this security group.
2. Create an EC2 instance for the application and add the EC2 instance to the VPC security group (sg-0123ec2example) that you created in the previous step.
3. Create a second VPC security group (for example, sg-6789rdsexample) and create a new rule by specifying the VPC security group that you created in step 1 (sg-0123ec2example) as the source.
4. Create a new DB instance and add the DB instance to the VPC security group (sg-6789rdsexample) that you created in the previous step. When you create the DB instance, use the same port number as the one specified for the VPC security group (sg-6789rdsexample) rule that you created in step 3.

The following diagram shows this scenario.



For more information about using a VPC, see [Amazon Virtual Private Cloud VPCs and Amazon Aurora \(p. 1787\)](#).

Creating a VPC security group

You can create a VPC security group for a DB instance by using the VPC console. For information about creating a security group, see [Provide access to the DB cluster in the VPC by creating a security group \(p. 87\)](#) and [Security groups](#) in the *Amazon Virtual Private Cloud User Guide*.

Associating a security group with a DB instance

You can associate a security group with a DB instance by using **Modify** on the RDS console, the `ModifyDBInstance` Amazon RDS API, or the `modify-db-instance` AWS CLI command.

For information about modifying a DB instance in a DB cluster, see [Modify a DB instance in a DB cluster \(p. 373\)](#). For security group considerations when you restore a DB instance from a DB snapshot, see [Security group considerations \(p. 497\)](#).

Associating a security group with a DB cluster

You can associate a security group with a DB cluster by using **Modify cluster** on the RDS console, the `ModifyDBCluster` Amazon RDS API, or the `modify-db-cluster` AWS CLI command.

For information about modifying a DB cluster, see [Modifying an Amazon Aurora DB cluster \(p. 372\)](#).

Master user account privileges

When you create a new DB cluster, the default master user that you use gets certain privileges for that DB cluster. The following table shows the privileges and database roles the master user gets for each of the database engines.

Important

We strongly recommend that you do not use the master user directly in your applications. Instead, adhere to the best practice of using a database user created with the minimal privileges required for your application.

Note

If you accidentally delete the permissions for the master user, you can restore them by modifying the DB cluster and setting a new master user password. For more information about modifying a DB cluster, see [Modifying an Amazon Aurora DB cluster \(p. 372\)](#).

Database engine	System privilege	Database role
Amazon Aurora MySQL	CREATE, DROP, GRANT OPTION, REFERENCES, EVENT, ALTER, DELETE, INDEX, INSERT, SELECT, UPDATE, CREATE TEMPORARY TABLES, LOCK TABLES, TRIGGER, CREATE VIEW, SHOW VIEW, LOAD FROM S3, SELECT INTO S3, ALTER ROUTINE, CREATE ROUTINE, EXECUTE, CREATE USER, PROCESS, SHOW DATABASES , RELOAD, REPLICATION CLIENT, REPLICATION SLAVE	—
Amazon Aurora PostgreSQL	LOGIN, NOSUPERUSER, INHERIT, CREATEDB, CREATEROLE, NOREPLICATION, VALID UNTIL 'infinity'	RDS_SUPERUSER

Using service-linked roles for Amazon Aurora

Amazon Aurora uses AWS Identity and Access Management (IAM) [service-linked roles](#). A service-linked role is a unique type of IAM role that is linked directly to Amazon Aurora. Service-linked roles are predefined by Amazon Aurora and include all the permissions that the service requires to call other AWS services on your behalf.

A service-linked role makes using Amazon Aurora easier because you don't have to manually add the necessary permissions. Amazon Aurora defines the permissions of its service-linked roles, and unless defined otherwise, only Amazon Aurora can assume its roles. The defined permissions include the trust policy and the permissions policy, and that permissions policy cannot be attached to any other IAM entity.

You can delete the roles only after first deleting their related resources. This protects your Amazon Aurora resources because you can't inadvertently remove permission to access the resources.

For information about other services that support service-linked roles, see [AWS services that work with IAM](#) and look for the services that have **Yes** in the **Service-Linked Role** column. Choose a **Yes** with a link to view the service-linked role documentation for that service.

Service-linked role permissions for Amazon Aurora

Amazon Aurora uses the service-linked role named AWSServiceRoleForRDS to allow Amazon RDS to call AWS services on behalf of your DB clusters.

The AWSServiceRoleForRDS service-linked role trusts the following services to assume the role:

- rds.amazonaws.com

This service-linked role has a permissions policy attached to it called [AmazonRDSServiceRolePolicy](#) that grants it permissions to operate in your account. The role permissions policy allows Amazon Aurora to complete the following actions on the specified resources:

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": [  
                "ec2:AuthorizeSecurityGroupIngress",  
                "ec2>CreateNetworkInterface",  
                "ec2>CreateSecurityGroup",  
                "ec2>DeleteNetworkInterface",  
                "ec2>DeleteSecurityGroup",  
                "ec2:DescribeAvailabilityZones",  
                "ec2:DescribeInternetGateways",  
                "ec2:DescribeSecurityGroups",  
                "ec2:DescribeSubnets",  
                "ec2:DescribeVpcAttribute",  
                "ec2:DescribeVpcs",  
                "ec2:ModifyNetworkInterfaceAttribute",  
                "ec2:ModifyVpcEndpoint",  
                "ec2:RevokeSecurityGroupIngress",  
                "ec2>CreateVpcEndpoint",  
                "ec2:DescribeVpcEndpoints",  
                "ec2>DeleteVpcEndpoints",  
                "ec2:AssignPrivateIpAddresses",  
                "ec2:UnassignPrivateIpAddresses"  
            ]  
        }  
    ]  
}
```

```

        ],
        "Resource": "*"
    },
    {
        "Effect": "Allow",
        "Action": [
            "sns:Publish"
        ],
        "Resource": "*"
    },
    {
        "Effect": "Allow",
        "Action": [
            "logs>CreateLogGroup"
        ],
        "Resource": [
            "arn:aws:logs:***:log-group:/aws/rds/*",
            "arn:aws:logs:***:log-group:/aws/docdb/*",
            "arn:aws:logs:***:log-group:/aws/neptune/*"
        ]
    },
    {
        "Effect": "Allow",
        "Action": [
            "logs>CreateLogStream",
            "logs:PutLogEvents",
            "logs:DescribeLogStreams"
        ],
        "Resource": [
            "arn:aws:logs:***:log-group:/aws/rds/*:log-stream:*",
            "arn:aws:logs:***:log-group:/aws/docdb/*:log-stream:*",
            "arn:aws:logs:***:log-group:/aws/neptune/*:log-stream:*

```

Note

You must configure permissions to allow an IAM entity (such as a user, group, or role) to create, edit, or delete a service-linked role. If you encounter the following error message:

Unable to create the resource. Verify that you have permission to create service linked role. Otherwise wait and try again later.

Make sure you have the following permissions enabled:

```
{
    "Action": "iam>CreateServiceLinkedRole",
    "Effect": "Allow",
```

```
"Resource": "arn:aws:iam::*:role/aws-service-role/rds.amazonaws.com/  
AWSServiceRoleForRDS",  
    "Condition": {  
        "StringLike": {  
            "iam:AWSServiceName": "rds.amazonaws.com"  
        }  
    }  
}
```

For more information, see [Service-linked role permissions in the IAM User Guide](#).

Creating a service-linked role for Amazon Aurora

You don't need to manually create a service-linked role. When you create a DB cluster, Amazon Aurora creates the service-linked role for you.

Important

If you were using the Amazon Aurora service before December 1, 2017, when it began supporting service-linked roles, then Amazon Aurora created the AWSServiceRoleForRDS role in your account. To learn more, see [A new role appeared in my AWS account](#).

If you delete this service-linked role, and then need to create it again, you can use the same process to recreate the role in your account. When you create a DB cluster, Amazon Aurora creates the service-linked role for you again.

Editing a service-linked role for Amazon Aurora

Amazon Aurora does not allow you to edit the AWSServiceRoleForRDS service-linked role. After you create a service-linked role, you cannot change the name of the role because various entities might reference the role. However, you can edit the description of the role using IAM. For more information, see [Editing a service-linked role](#) in the *IAM User Guide*.

Deleting a service-linked role for Amazon Aurora

If you no longer need to use a feature or service that requires a service-linked role, we recommend that you delete that role. That way you don't have an unused entity that is not actively monitored or maintained. However, you must delete all of your DB clusters before you can delete the service-linked role.

Cleaning up a service-linked role

Before you can use IAM to delete a service-linked role, you must first confirm that the role has no active sessions and remove any resources used by the role.

To check whether the service-linked role has an active session in the IAM console

1. Sign in to the AWS Management Console and open the IAM console at <https://console.aws.amazon.com/iam/>.
2. In the navigation pane of the IAM console, choose **Roles**. Then choose the name (not the check box) of the AWSServiceRoleForRDS role.
3. On the **Summary** page for the chosen role, choose the **Access Advisor** tab.
4. On the **Access Advisor** tab, review recent activity for the service-linked role.

Note

If you are unsure whether Amazon Aurora is using the AWSServiceRoleForRDS role, you can try to delete the role. If the service is using the role, then the deletion fails and you can view the AWS Regions where the role is being used. If the role is being used, then you must wait

for the session to end before you can delete the role. You cannot revoke the session for a service-linked role.

If you want to remove the AWSServiceRoleForRDS role, you must first delete *all* of your DB clusters.

[Deleting all of your clusters](#)

Use one of the following procedures to delete a single cluster. Repeat the procedure for each of your clusters.

To delete a cluster (console)

1. Open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. In the **Databases** list, choose the cluster that you want to delete.
3. For **Cluster Actions**, choose **Delete**.
4. Choose **Delete**.

To delete a cluster (CLI)

See [delete-db-cluster](#) in the *AWS CLI Command Reference*.

To delete a cluster (API)

See [DeleteDBCluster](#) in the *Amazon RDS API Reference*.

You can use the IAM console, the IAM CLI, or the IAM API to delete the AWSServiceRoleForRDS service-linked role. For more information, see [Deleting a service-linked role](#) in the *IAM User Guide*.

Amazon Virtual Private Cloud VPCs and Amazon Aurora

Amazon Virtual Private Cloud (Amazon VPC) enables you to launch AWS resources, such as Aurora DB clusters, into a virtual private cloud (VPC).

When you use an Amazon VPC, you have control over your virtual networking environment: you can choose your own IP address range, create subnets, and configure routing and access control lists. There is no additional cost to run your DB instance in an Amazon VPC.

Accounts that support only the *EC2-VPC* platform have a default VPC. All new DB instances are created in the default VPC unless you specify otherwise. If you are a new Amazon Aurora customer, if you have never created a DB instance before, or if you are creating a DB instance in an AWS Region you have not used before, you are most likely on the *EC2-VPC* platform and have a default VPC.

Topics

- [Working with a DB instance in a VPC \(p. 1787\)](#)
- [How to create a VPC for use with Amazon Aurora \(p. 1793\)](#)
- [Scenarios for accessing a DB instance in a VPC \(p. 1800\)](#)
- [Tutorial: Create an Amazon VPC for use with a DB instance \(p. 1805\)](#)

This documentation only discusses VPC functionality relevant to Amazon Aurora DB clusters. For more information about Amazon VPC, see [Amazon VPC Getting Started Guide](#) and [Amazon VPC User Guide](#). For information about using a network address translation (NAT) gateway, see [NAT gateways](#) in the [Amazon Virtual Private Cloud User Guide](#).

Working with a DB instance in a VPC

Your DB instance is in a virtual private cloud (VPC). A VPC is a virtual network that is logically isolated from other virtual networks in the AWS Cloud. Amazon VPC lets you launch AWS resources, such as an Amazon Aurora DB instance or Amazon EC2 instance, into a VPC. The VPC can either be a default VPC that comes with your account or one that you create. All VPCs are associated with your AWS account.

Your default VPC has three subnets you can use to isolate resources inside the VPC. The default VPC also has an internet gateway that can be used to provide access to resources inside the VPC from outside the VPC.

For a list of scenarios involving Amazon Aurora DB instances in a VPC , see [Scenarios for accessing a DB instance in a VPC \(p. 1800\)](#).

For a tutorial that shows you how to create a VPC that you can use with a common Amazon Aurora scenario, see [Tutorial: Create an Amazon VPC for use with a DB instance \(p. 1805\)](#).

To learn how to work with DB instances inside a VPC, see the following:

Topics

- [Working with a DB instance in a VPC \(p. 1788\)](#)
- [Working with DB subnet groups \(p. 1788\)](#)
- [Hiding a DB instance in a VPC from the internet \(p. 1789\)](#)
- [Creating a DB instance in a VPC \(p. 1790\)](#)

Working with a DB instance in a VPC

Here are some tips on working with a DB instance in a VPC:

- Your VPC must have at least two subnets. These subnets must be in two different Availability Zones in the AWS Region where you want to deploy your DB instance. A subnet is a segment of a VPC's IP address range that you can specify and that lets you group instances based on your security and operational needs.
- If you want your DB instance in the VPC to be publicly accessible, you must enable the VPC attributes *DNS hostnames* and *DNS resolution*.
- Your VPC must have a DB subnet group that you create (for more information, see the next section). You create a DB subnet group by specifying the subnets you created. Amazon Aurora chooses a subnet and an IP address within that subnet to associate with your DB instance. The DB instance uses the Availability Zone that contains the subnet.
- Your VPC must have a VPC security group that allows access to the DB instance.
- The CIDR blocks in each of your subnets must be large enough to accommodate spare IP addresses for Amazon Aurora to use during maintenance activities, including failover and compute scaling.
- A VPC can have an *instance tenancy* attribute of either *default* or *dedicated*. All default VPCs have the instance tenancy attribute set to default, and a default VPC can support any DB instance class.

If you choose to have your DB instance in a dedicated VPC where the instance tenancy attribute is set to dedicated, the DB instance class of your DB instance must be one of the approved Amazon EC2 dedicated instance types. For example, the m3.medium EC2 dedicated instance corresponds to the db.m3.medium DB instance class. For information about instance tenancy in a VPC, see [Dedicated instances in the Amazon Elastic Compute Cloud User Guide](#).

For more information about the instance types that can be in a dedicated instance, see [Amazon EC2 dedicated instances](#) on the EC2 pricing page.

Note

When you set the instance tenancy attribute to dedicated for an Amazon RDS DB instance, it doesn't guarantee that the DB instance will run on a dedicated host.

Working with DB subnet groups

Subnets are segments of a VPC's IP address range that you designate to group your resources based on security and operational needs. A DB subnet group is a collection of subnets (typically private) that you create in a VPC and that you then designate for your DB instances. A DB subnet group allows you to specify a particular VPC when creating DB instances using the CLI or API; if you use the console, you can just choose the VPC and subnets you want to use.

Each DB subnet group should have subnets in at least two Availability Zones in a given AWS Region. When creating a DB instance in a VPC, you must choose a DB subnet group. From the DB subnet group, Amazon Aurora chooses a subnet and an IP address within that subnet to associate with your DB instance. The DB instance uses the Availability Zone that contains the subnet. If the primary DB instance of a Multi-AZ deployment fails, Amazon Aurora can promote the corresponding standby and subsequently create a new standby using an IP address of the subnet in one of the other Availability Zones.

The subnets in a DB subnet group are either public or private. They can't be a mix of both public and private subnets. The subnets are public or private, depending on the configuration that you set for their network access control lists (network ACLs) and routing tables.

When Amazon Aurora creates a DB instance in a VPC, it assigns a network interface to your DB instance by using an IP address from your DB subnet group. However, we strongly recommend that you use the DNS name to connect to your DB instance because the underlying IP address changes during failover.

Note

For each DB instance that you run in a VPC, make sure to reserve at least one address in each subnet in the DB subnet group for use by Amazon Aurora for recovery actions.

Hiding a DB instance in a VPC from the internet

One common Amazon Aurora scenario is to have a VPC in which you have an EC2 instance with a public-facing web application and a DB instance with a database that is not publicly accessible. For example, you can create a VPC that has a public subnet and a private subnet. Amazon EC2 instances that function as web servers can be deployed in the public subnet, and the DB instances are deployed in the private subnet. In such a deployment, only the web servers have access to the DB instances. For an illustration of this scenario, see [A DB instance in a VPC accessed by an EC2 instance in the same VPC \(p. 1800\)](#).

When you launch a DB instance inside a VPC, the DB instance has a private IP address for traffic inside the VPC. This private IP address isn't publicly accessible. You can use the *Public access* option to designate whether the DB instance also has a public IP address in addition to the private IP address. If the DB instance is designated as publicly accessible, its DNS endpoint resolves to the private IP address from within the DB instance's VPC, and to the public IP address from outside of the DB instance's VPC. Access to the DB instance is ultimately controlled by the security group it uses, and that public access is not permitted if the security group assigned to the DB instance doesn't permit it.

You can modify a DB instance to turn on or off public accessibility by modifying the *Public access* option. For more information, see the modifying section for your DB engine.

The following illustration shows the **Public access** option in the **Additional connectivity configuration** section. To set the option, open the **Additional connectivity configuration** section in the **Connectivity** section.

Connectivity

Subnet group
default ▾

Security group
List of DB security groups to associate with this DB instance.
Choose security groups ▾
default X

Certificate authority
rds-ca-2019 ▾

▼ Additional connectivity configuration

Public access

Publicly accessible
EC2 instances and devices outside the VPC can connect to the instance. You define the security groups for supported devices and instances.

Not publicly accessible
No IP address is assigned to the DB instance. EC2 instances and devices outside the VPC can't connect.

⚠ Setting your DB instance accessibility to private might result in the loss of connectivity to your database. [Learn more](#)

Database port
Specify the TCP/IP port that the DB instance will use for application connections. The application connection string must specify the port number. The DB security group and your firewall must allow connections to the port. [Learn more](#) ▾
3306 ▾

For information about modifying a DB instance to set the **Public access** option, see [Modify a DB instance in a DB cluster \(p. 373\)](#).

Creating a DB instance in a VPC

The following procedures help you create a DB instance in a VPC. If your account has a default VPC, you can begin with step 3 because the VPC and DB subnet group have already been created for you. If your AWS account doesn't have a default VPC, or if you want to create an additional VPC, you can create a new VPC.

Note

If you want your DB instance in the VPC to be publicly accessible, you must update the DNS information for the VPC by enabling the VPC attributes *DNS hostnames* and *DNS resolution*. For information about updating the DNS information for a VPC instance, see [Updating DNS support for your VPC](#).

Follow these steps to create a DB instance in a VPC:

- [Step 1: Create a VPC \(p. 1791\)](#)
- [Step 2: Add subnets to the VPC \(p. 1791\)](#)
- [Step 3: Create a DB subnet group \(p. 1791\)](#)
- [Step 4: Create a VPC security group \(p. 1793\)](#)
- [Step 5: Create a DB instance in the VPC \(p. 1793\)](#)

Step 1: Create a VPC

If your AWS account does not have a default VPC or if you want to create an additional VPC, follow the instructions for creating a new VPC. See [Create a VPC with private and public subnets \(p. 1805\)](#), or see [Step 1: Create a VPC](#) in the Amazon VPC documentation.

Step 2: Add subnets to the VPC

Once you have created a VPC, you need to create subnets in at least two Availability Zones. You use these subnets when you create a DB subnet group. If you have a default VPC, a subnet is automatically created for you in each Availability Zone in the AWS Region.

For instructions on how to create subnets in a VPC, see [Create a VPC with private and public subnets \(p. 1805\)](#).

Step 3: Create a DB subnet group

A DB subnet group is a collection of subnets (typically private) that you create for a VPC and that you then designate for your DB instances. A DB subnet group allows you to specify a particular VPC when you create DB instances using the CLI or API. If you use the console, you can just choose the VPC and subnets you want to use. Each DB subnet group must have at least one subnet in at least two Availability Zones in the AWS Region.

For a DB instance to be publicly accessible, the subnets in the DB subnet group must have an internet gateway. For more information about internet gateways for subnets, see [Internet gateways](#) in the Amazon VPC documentation.

When you create a DB instance in a VPC, make sure to choose a DB subnet group. Amazon Aurora chooses a subnet and an IP address within that subnet to associate with your DB instance. Amazon Aurora creates and associates an Elastic Network Interface to your DB instance with that IP address. The DB instance uses the Availability Zone that contains the subnet. For Multi-AZ deployments, defining a subnet for two or more Availability Zones in an AWS Region allows Amazon Aurora to create a new standby in another Availability Zone should the need arise. You need to do this even for Single-AZ deployments, just in case you want to convert them to Multi-AZ deployments at some point.

In this step, you create a DB subnet group and add the subnets that you created for your VPC.

To create a DB subnet group

1. Open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. In the navigation pane, choose **Subnet groups**.
3. Choose **Create DB Subnet Group**.
4. For **Name**, type the name of your DB subnet group.
5. For **Description**, type a description for your DB subnet group.
6. For **VPC**, choose the VPC that you created.
7. In the **Add subnets** section, choose the Availability Zones that include the subnets from **Availability Zones**, and then choose the subnets from **Subnets**.

Create DB Subnet Group

To create a new subnet group, give it a name and a description, and choose an existing VPC. You will then be able to add subnets related to that VPC.

Subnet group details

Name

You won't be able to modify the name after your subnet group has been created.

mydbsubnetgroup

Must contain from 1 to 255 characters. Alphanumeric characters, spaces, hyphens, underscores, and periods are allowed.

Description

My DB Subnet Group

VPC

Choose a VPC identifier that corresponds to the subnets you want to use for your DB subnet group. You won't be able to choose a different VPC identifier after your subnet group has been created.

tutorial-vpc (vpc-068fe388385afc014)

Add subnets

Availability Zones

Choose the Availability Zones that include the subnets you want to add.

Choose an availability zone

us-east-1a X us-east-1c X

Subnets

Choose the subnets that you want to add. The list includes the subnets in the selected Availability Zones.

Select subnets

subnet-079bd4b8953aee1dd (10.0.0.0/24) X

subnet-057e85b72c46fdd9a (10.0.1.0/24) X

Subnets selected (2)

Availability zone	Subnet ID	CIDR block
us-east-1a	subnet-079bd4b8953aee1dd	10.0.0.0/24
us-east-1c	subnet-057e85b72c46fdd9a	10.0.1.0/24

Cancel

Create

8. Choose **Create**.

Your new DB subnet group appears in the DB subnet groups list on the RDS console. You can choose the DB subnet group to see details, including all of the subnets associated with the group, in the details pane at the bottom of the window.

Step 4: Create a VPC security group

Before you create your DB instance, you must create a VPC security group to associate with your DB instance. For instructions on how to create a security group for your DB instance, see [Create a VPC security group for a private DB instance \(p. 1808\)](#), or see [Security groups for your VPC](#) in the Amazon VPC documentation.

Step 5: Create a DB instance in the VPC

In this step, you create a DB instance and use the VPC name, the DB subnet group, and the VPC security group you created in the previous steps.

Note

If you want your DB instance in the VPC to be publicly accessible, you must enable the VPC attributes *DNS hostnames* and *DNS resolution*. For information on updating the DNS information for a VPC instance, see [Updating DNS support for your VPC](#).

For details on how to create a DB instance, see [Creating an Amazon Aurora DB cluster \(p. 125\)](#).

When prompted in the **Connectivity** section, enter the VPC name, the DB subnet group, and the VPC security group you created in the previous steps.

Note

Updating VPCs is not currently supported for Aurora clusters.

How to create a VPC for use with Amazon Aurora

The following sections discuss how to create a VPC for use with Amazon Aurora.

Note

For a helpful and detailed guide on connecting to an Amazon Aurora DB cluster, you can see [Aurora MySQL database administrator's handbook – Connection management](#).

Create a VPC and subnets

You can only create an Amazon Aurora DB cluster in a Virtual Private Cloud (VPC) that spans two Availability Zones, and each zone must contain at least one subnet. You can create an Aurora DB cluster in the default VPC for your AWS account, or you can create a user-defined VPC. For information, see [Amazon Virtual Private Cloud VPCs and Amazon Aurora \(p. 1787\)](#).

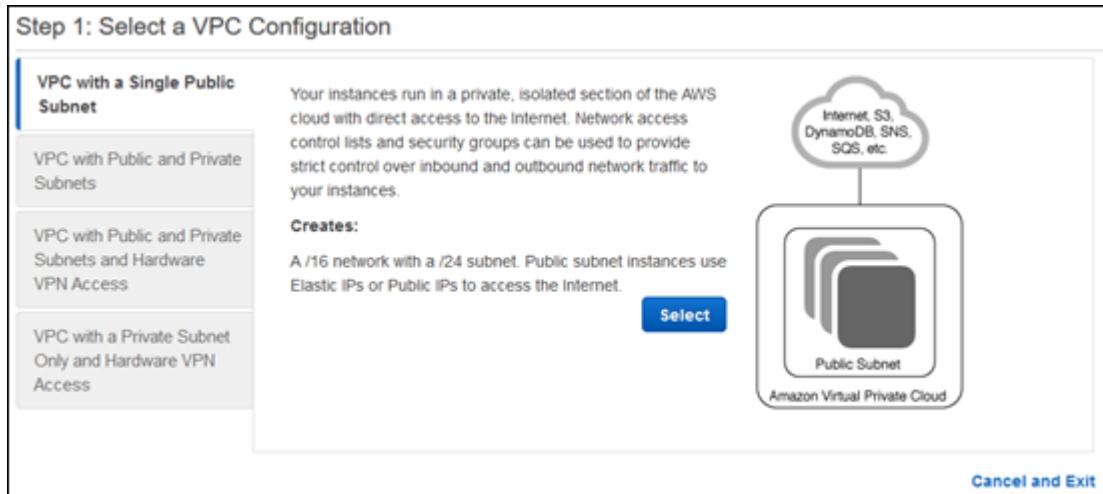
Amazon Aurora optionally can create a VPC and subnet group for you to use with your DB cluster. Doing this can be helpful if you have never created a VPC, or if you would like to create a new VPC that is separate from your other VPCs. If you want Amazon Aurora to create a VPC and subnet group for you, then skip this procedure and see [Create an Aurora MySQL DB cluster \(p. 89\)](#) or [Create an Aurora PostgreSQL DB cluster \(p. 97\)](#).

Note

All VPC and EC2 resources that you use with your Aurora DB cluster must be in one of the regions listed in [Regions and Availability Zones \(p. 11\)](#).

To create a VPC for use with an Aurora DB cluster

1. Sign in to the AWS Management Console and open the Amazon VPC console at <https://console.aws.amazon.com/vpc/>.
2. In the top-right corner of the AWS Management Console, choose the AWS Region to create your VPC in. This example uses the US East (Ohio) Region.
3. In the upper-left corner, choose **VPC Dashboard**. Choose **Start VPC Wizard** to begin creating a VPC.
4. In the Create VPC wizard, choose **VPC with a Single Public Subnet**. Choose **Select**.



5. Set the following values in the **Create VPC** panel:

- **IP CIDR block:** 10.0.0.0/16
- **VPC name:** gs-cluster-vpc
- **Public subnet:** 10.0.0.0/24
- **Availability Zone:** us-east-1a
- **Subnet name:** gs-subnet1
- **Enable DNS hostnames:** Yes
- **Hardware tenancy:** Default

Step 2: VPC with a Single Public Subnet

IPv4 CIDR block*: (65531 IP addresses available)

IPv6 CIDR block: No IPv6 CIDR Block
 Amazon provided IPv6 CIDR block

VPC name:

Public subnet's IPv4 CIDR*: (251 IP addresses available)

Availability Zone*:

Subnet name:

You can add more subnets after AWS creates the VPC.

Service endpoints

Enable DNS hostnames*: Yes No

Hardware tenancy*:

6. Choose **Create VPC**.

7. When your VPC has been created, choose **Close** on the notification page.

To create additional subnets

1. To add the second subnet to your VPC, in the VPC Dashboard choose **Subnets**, and then choose **Create Subnet**. An Amazon Aurora DB cluster requires at least two VPC subnets.
2. Set the following values in the **Create Subnet** panel:
 - **Name tag:** gs-subnet2
 - **VPC:** Choose the VPC that you created in the previous step, for example: vpc-a464d1c1 (10.0.0.0/16) | gs-cluster-vpc.
 - **Availability Zone:** us-east-1c
 - **CIDR block:** 10.0.1.0/24

VPC CIDRs	CIDR	Status	Status Reason
	10.0.0.0/16	associated	

Availability Zone: us-east-1c
 IPv4 CIDR block: 10.0.1.0/24

[Cancel](#) [Yes, Create](#)

3. Choose **Yes Create**.
4. To ensure that the second subnet that you created uses the same route table as the first subnet, in the VPC Dashboard, choose **Subnets**, and then choose the first subnet that was created for the VPC, **gs-subnet1**. Choose the **Route Table** tab, and note the **Current Route Table**, for example: **rtb-c16ce5bc**.
5. In the list of subnets, clear the first subnet and choose the second subnet, **gs-subnet2**. Choose the **Route Table** tab, and then choose **Edit**. In the **Change to** list, choose the route table from the previous step, for example: **rtb-c16ce5bc**. Choose **Save** to save your choice.

Destination	Target
10.0.0.0/16	local
0.0.0.0/0	igw-3c84af45

Current Route Table: [rtb-c16ce5bb](#)
 Change to: [rtb-c16ce5bc](#)

[Cancel](#) [Save](#)

Create a security group and add inbound rules

After you've created your VPC and subnets, the next step is to create a security group and add inbound rules.

To create a security group

The last step in creating a VPC for use with your Amazon Aurora DB cluster is to create a VPC security group, which identifies which network addresses and protocols are allowed to access DB instances in your VPC.

1. In the VPC Dashboard, choose **Security Groups**, and then choose **Create Security Group**.
2. Set the following values in the **Create Security Group** panel:

- **Name tag:** gs-securitygroup1
- **Group name:** gs-securitygroup1
- **Description:** Getting Started Security Group
- **VPC:** Choose the VPC that you created earlier, for example: vpc-b5754bcd | gs-cluster-vpc.



3. Choose **Yes, Create** to create the security group.

To add inbound rules to the security group

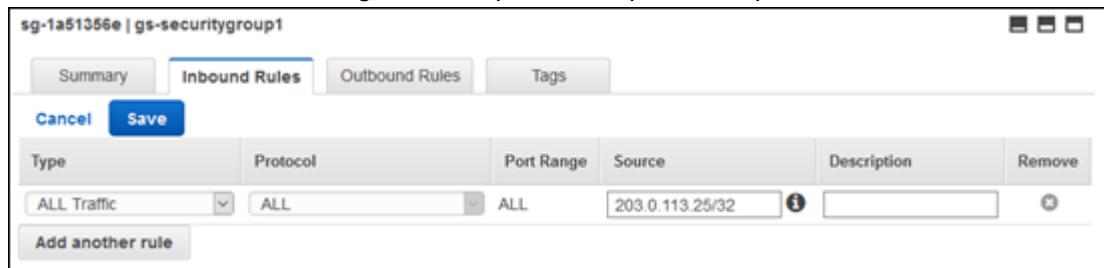
To connect to your Aurora DB cluster, you need to add an inbound rule to your VPC security group that allows inbound traffic to connect.

1. Determine the IP address to use to connect to the Aurora cluster. You can use the service at <https://checkip.amazonaws.com> to determine your public IP address. If you are connecting through an ISP or from behind your firewall without a static IP address, you need to find out the range of IP addresses used by client computers.

Warning

If you use 0.0.0.0/0, you enable all IP addresses to access your DB cluster. This is acceptable for a short time in a test environment, but it's unsafe for production environments. In production, you'll authorize only a specific IP address or range of addresses to access your DB cluster.

2. In the VPC Dashboard, choose **Security Groups**, and then choose the gs-securitygroup1 security group that you created in the previous procedure.
3. Choose the **Inbound** tab, and then choose the **Edit** button.
4. Set the following values for your new inbound rule:
 - **Type:** All Traffic
 - **Source:** The IP address or range from the previous step, for example 203.0.113.25/32.



5. Choose **Save** to save your settings.

Create a DB subnet group

The last thing that you need before you can create an Aurora DB cluster is a DB subnet group. Your DB subnet group identifies the subnets that your DB cluster uses from the VPC that you created in the previous steps. Your DB subnet group must include at least one subnet in at least two of the Availability Zones in the AWS Region where you want to deploy your DB cluster.

To create a DB subnet group for use with your Aurora DB cluster

1. Open the Amazon Aurora console at <https://console.aws.amazon.com/rds>.
2. Choose **Subnet Groups**, and then choose **Create DB Subnet Group**.
3. Set the following values for your new DB subnet group:
 - **Name:** gs-subnetgroup1
 - **Description:** Getting Started Subnet Group
 - **VPC ID:** Choose the VPC that you created in the previous procedure, for example, gs-cluster-vpc (vpc-b5754bcd).
4. In the **Add subnets** section, choose the Availability Zones that include the subnets from **Availability Zones**, and then choose the subnets from **Subnets**.

Create DB Subnet Group

To create a new subnet group, give it a name and a description, and choose an existing VPC. You will then be able to add subnets related to that VPC.

Subnet group details

Name

You won't be able to modify the name after your subnet group has been created.

gs-subnetgroup1

Must contain from 1 to 255 characters. Alphanumeric characters, spaces, hyphens, underscores, and periods are allowed.

Description

Getting Started Subnet Group

VPC

Choose a VPC identifier that corresponds to the subnets you want to use for your DB subnet group. You won't be able to choose a different VPC identifier after your subnet group has been created.

gs-cluster-vpc (vpc-068fe388385afc014) ▾

Add subnets

Availability Zones

Choose the Availability Zones that include the subnets you want to add.

Choose an availability zone ▾

us-east-1a X

us-east-1c X

Subnets

Choose the subnets that you want to add. The list includes the subnets in the selected Availability Zones.

Select subnets ▾

subnet-079bd4b8953aee1dd (10.0.0.0/24) X

subnet-057e85b72c46fdd9a (10.0.1.0/24) X

Subnets selected (2)

Availability zone	Subnet ID	CIDR block
us-east-1a	subnet-079bd4b8953aee1dd	10.0.0.0/24
us-east-1c	subnet-057e85b72c46fdd9a	10.0.1.0/24

Cancel

Create

5. Choose **Create** to create the subnet group.

Scenarios for accessing a DB instance in a VPC

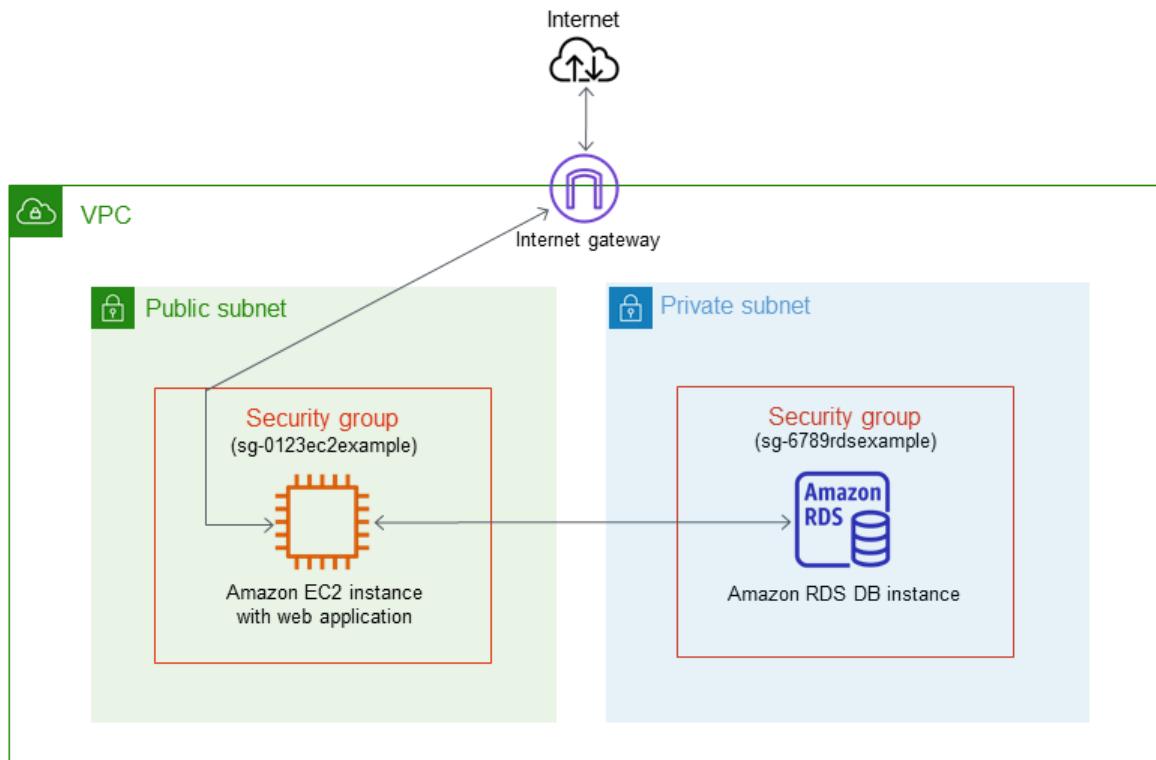
Amazon Aurora supports the following scenarios for accessing a DB instance in a VPC:

- An EC2 instance in the same VPC (p. 1800)
- An EC2 instance in a different VPC (p. 1801)
- A client application through the internet (p. 1802)
- A private network (p. 1803)
- An EC2 instance not in a VPC (p. 1803)

A DB instance in a VPC accessed by an EC2 instance in the same VPC

A common use of a DB instance in a VPC is to share data with an application server that is running in an EC2 instance in the same VPC. This is the user scenario created if you use AWS Elastic Beanstalk to create an EC2 instance and a DB instance in the same VPC.

The following diagram shows this scenario.



The simplest way to manage access between EC2 instances and DB instances in the same VPC is to do the following:

- Create a VPC security group for your DB instances to be in. This security group can be used to restrict access to the DB instances. For example, you can create a custom rule for this security group that

allows TCP access using the port you assigned to the DB instance when you created it and an IP address you use to access the DB instance for development or other purposes.

- Create a VPC security group for your EC2 instances (web servers and clients) to be in. This security group can, if needed, allow access to the EC2 instance from the internet by using the VPC's routing table. For example, you can set rules on this security group to allow TCP access to the EC2 instance over port 22.
- Create custom rules in the security group for your DB instances that allow connections from the security group you created for your EC2 instances. This would allow any member of the security group to access the DB instances.

For a tutorial that shows you how to create a VPC with both public and private subnets for this scenario, see [Tutorial: Create an Amazon VPC for use with a DB instance \(p. 1805\)](#).

To create a rule in a VPC security group that allows connections from another security group, do the following:

1. Sign in to the AWS Management Console and open the Amazon VPC console at <https://console.aws.amazon.com/vpc>.
2. In the navigation pane, choose **Security Groups**.
3. Choose or create a security group for which you want to allow access to members of another security group. In the scenario preceding, this is the security group that you use for your DB instances. Choose the **Inbound rules** tab, and then choose **Edit inbound rules**.
4. On the **Edit inbound rules** page, choose **Add rule**.
5. From **Type**, choose the entry that corresponds to the port you used when you created your DB instance, such as **MySQL/Aurora**.
6. In the **Source** box, start typing the ID of the security group, which lists the matching security groups. Choose the security group with members that you want to have access to the resources protected by this security group. In the scenario preceding, this is the security group that you use for your EC2 instance.
7. If required, repeat the steps for the TCP protocol by creating a rule with **All TCP** as the **Type** and your security group in the **Source** box. If you intend to use the UDP protocol, create a rule with **All UDP** as the **Type** and your security group in the **Source** box.
8. Choose **Save rules** when you are done.

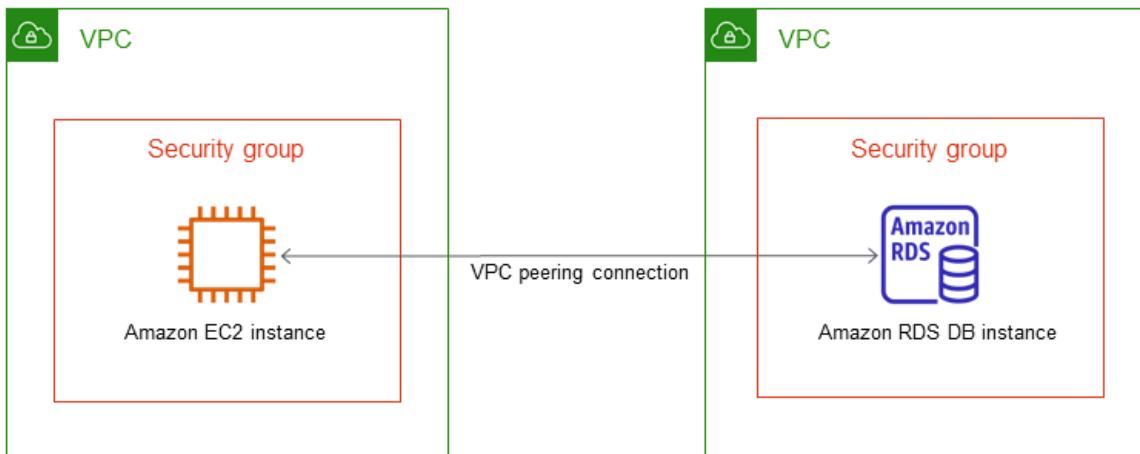
The following screen shows an inbound rule with a security group for its source.

Details	Inbound rules	Outbound rules	Tags
Inbound rules			
Type	Protocol	Port range	Source
MySQL/Aurora	TCP	3306	sg-00bd2328e37926844 (tutorial-securitygroup)

A DB instance in a VPC accessed by an EC2 instance in a different VPC

When your DB instance is in a different VPC from the EC2 instance you are using to access it, you can use VPC peering to access the DB instance.

The following diagram shows this scenario.

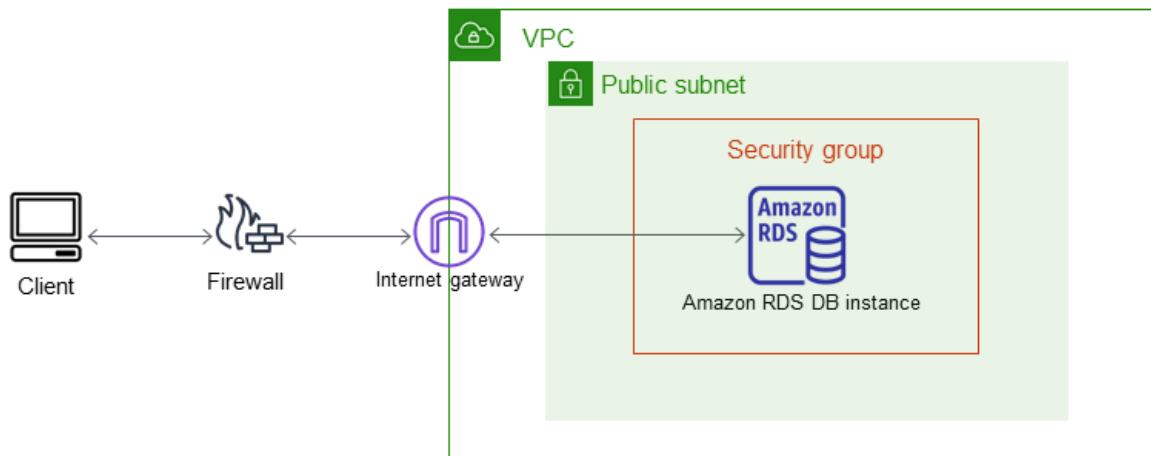


A VPC peering connection is a networking connection between two VPCs that enables you to route traffic between them using private IP addresses. Instances in either VPC can communicate with each other as if they are within the same network. You can create a VPC peering connection between your own VPCs, with a VPC in another AWS account, or with a VPC in a different AWS Region. To learn more about VPC peering, see [VPC peering](#) in the *Amazon Virtual Private Cloud User Guide*.

A DB instance in a VPC accessed by a client application through the internet

To access a DB instance in a VPC from a client application through the internet, you configure a VPC with a single public subnet, and an internet gateway to enable communication over the internet.

The following diagram shows this scenario.



We recommend the following configuration:

- A VPC of size /16 (for example CIDR: 10.0.0.0/16). This size provides 65,536 private IP addresses.
- A subnet of size /24 (for example CIDR: 10.0.0.0/24). This size provides 256 private IP addresses.
- An Amazon Aurora DB instance that is associated with the VPC and the subnet. Amazon RDS assigns an IP address within the subnet to your DB instance.
- An internet gateway which connects the VPC to the internet and to other AWS products.

- A security group associated with the DB instance. The security group's inbound rules allow your client application to access to your DB instance.

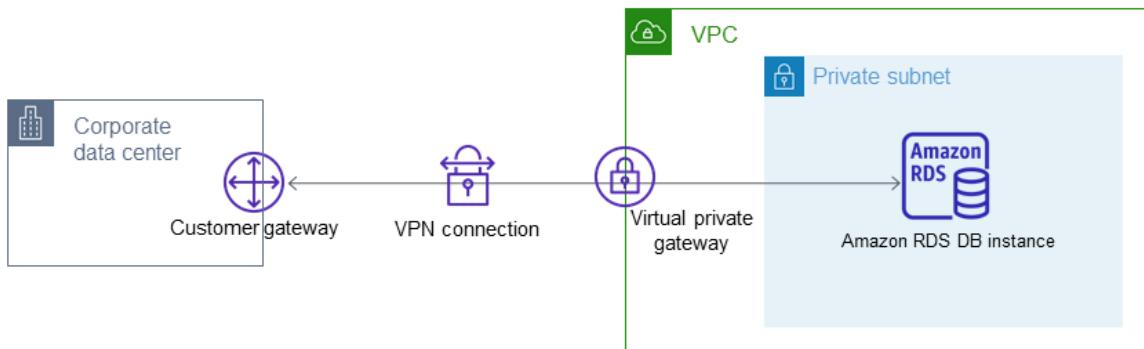
For information about creating a DB instance in a VPC, see [Creating a DB instance in a VPC \(p. 1790\)](#).

A DB instance in a VPC accessed by a private network

If your DB instance isn't publicly accessible, you have the following options for accessing it from a private network:

- An AWS Site-to-Site VPN connection. For more information, see [What is AWS Site-to-Site VPN?](#)
- An AWS Direct Connect connection. For more information, see [What is AWS Direct Connect?](#)

The following diagram shows a scenario with an AWS Site-to-Site VPN connection.



For more information, see [Internetwork traffic privacy \(p. 1723\)](#).

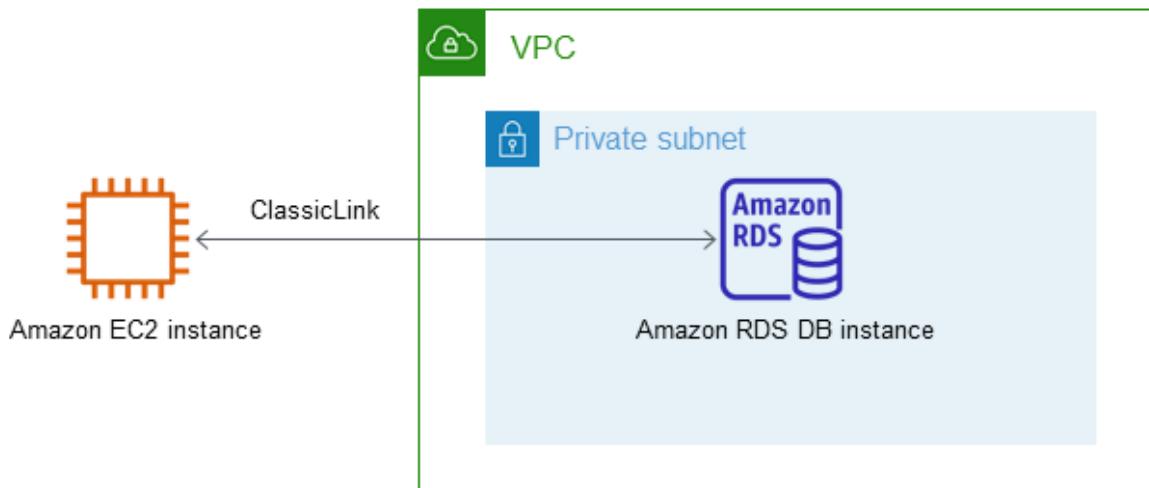
A DB instance in a VPC accessed by an EC2 instance not in a VPC

You can communicate between an Amazon Aurora DB instance that is in a VPC and an EC2 instance that is not in an Amazon VPC by using *ClassicLink*. When you use ClassicLink, an application on the EC2 instance can connect to the DB instance by using the endpoint for the DB instance. ClassicLink is available at no charge.

Important

If your EC2 instance was created after 2013, it is probably in a VPC.

The following diagram shows this scenario.



Using ClassicLink, you can connect an EC2 instance to a logically isolated database where you define the IP address range and control the access control lists (ACLs) to manage network traffic. You don't have to use public IP addresses or tunneling to communicate with the DB instance in the VPC. This arrangement provides you with higher throughput and lower latency connectivity for inter-instance communications.

To enable ClassicLink between a DB instance in a VPC and an EC2 instance not in a VPC

1. Sign in to the AWS Management Console and open the Amazon VPC console at <https://console.aws.amazon.com/vpc>.
2. In the navigation pane, choose **Your VPCs**.
3. Choose the VPC used by the DB instance.
4. In **Actions**, choose **Enable ClassicLink**. In the confirmation dialog box, choose **Yes, Enable**.
5. On the EC2 console, choose the EC2 instance you want to connect to the DB instance in the VPC.
6. In **Actions**, choose **ClassicLink**, and then choose **Link to VPC**.
7. On the **Link to VPC** page, choose the security group you want to use, and then choose **Link to VPC**.

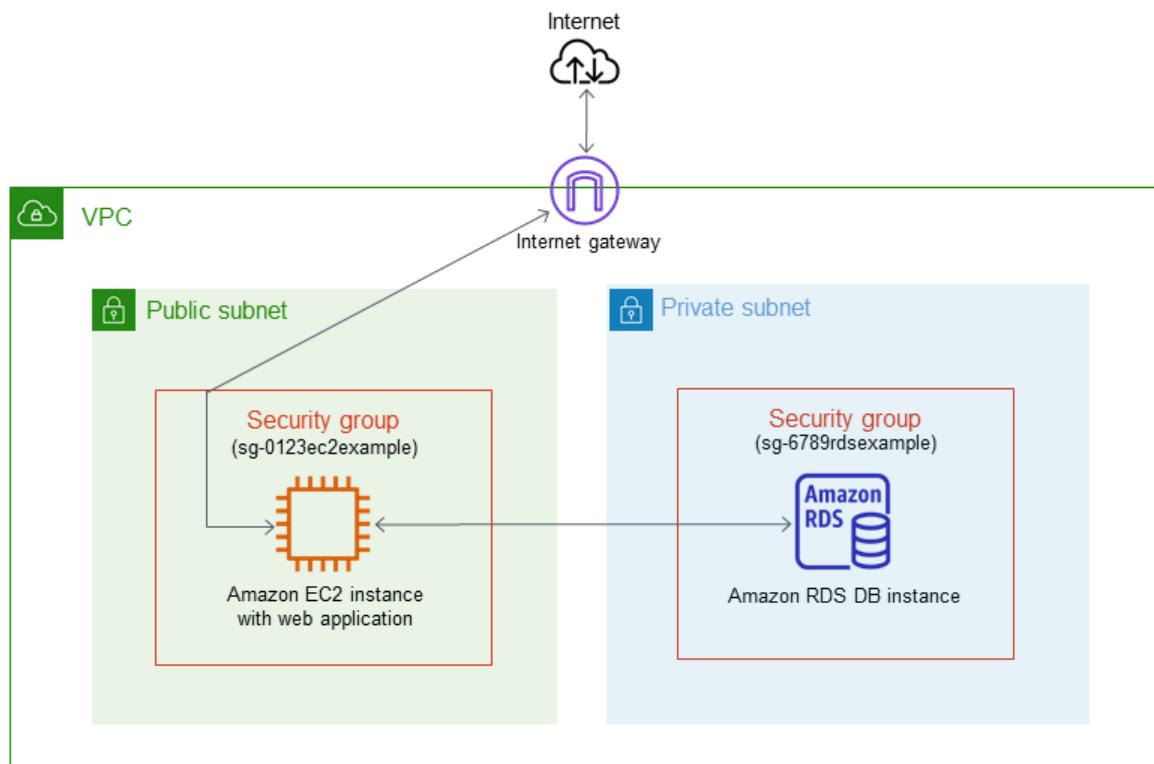
Note

The ClassicLink features are only visible in the consoles for accounts and regions that support EC2-Classic. For more information, see [ClassicLink](#) in the *Amazon EC2 User Guide for Linux Instances*.

Tutorial: Create an Amazon VPC for use with a DB instance

A common scenario includes a DB instance in an Amazon VPC, that shares data with a web server that is running in the same VPC. In this tutorial you create the VPC for this scenario.

The following diagram shows this scenario. For information about other scenarios, see [Scenarios for accessing a DB instance in a VPC \(p. 1800\)](#).



Because your DB instance only needs to be available to your web server, and not to the public Internet, you create a VPC with both public and private subnets. The web server is hosted in the public subnet, so that it can reach the public Internet. The DB instance is hosted in a private subnet. The web server is able to connect to the DB instance because it is hosted within the same VPC, but the DB instance is not available to the public Internet, providing greater security.

This tutorial describes configuring a VPC for Amazon Aurora DB clusters. For more information about Amazon VPC, see [Amazon VPC Getting Started Guide](#) and [Amazon VPC User Guide](#).

Note

For a tutorial that shows you how to create a web server for this VPC scenario, see [Tutorial: Create a web server and an Amazon Aurora DB cluster \(p. 103\)](#).

Create a VPC with private and public subnets

Use the following procedure to create a VPC with both public and private subnets.

To create a VPC and subnets

1. If you don't have an Elastic IP address to associate with a network address translation (NAT) gateway, allocate one now. A NAT gateway is required for this tutorial. If you have an available Elastic IP address, move on to the next step.
 - a. Open the Amazon EC2 console at <https://console.aws.amazon.com/ec2/>.
 - b. In the top-right corner of the AWS Management Console, choose the Region to allocate your Elastic IP address in. The Region of your Elastic IP address should be the same as the Region where you want to create your VPC. This example uses the US West (Oregon) Region.
 - c. In the navigation pane, choose **Elastic IPs**.
 - d. Choose **Allocate Elastic IP address**.
 - e. If the console shows the **Network Border Group** field, keep the default value for it.
 - f. For **Public IPv4 address pool**, choose **Amazon's pool of IPv4 addresses**.
 - g. Choose **Allocate**.

Note the allocation ID of the new Elastic IP address because you'll need this information when you create your VPC.

For more information about Elastic IP addresses, see [Elastic IP addresses](#) in the *Amazon EC2 User Guide*. For more information about NAT gateways, see [NAT gateways](#) in the *Amazon VPC User Guide*.

2. Open the Amazon VPC console at <https://console.aws.amazon.com/vpc/>.
3. In the top-right corner of the AWS Management Console, choose the Region to create your VPC in. This example uses the US West (Oregon) Region.
4. In the upper-left corner, choose **VPC Dashboard**. To begin creating a VPC, choose **Launch VPC Wizard**.
5. On the **Step 1: Select a VPC Configuration** page, choose **VPC with Public and Private Subnets**, and then choose **Select**.
6. On the **Step 2: VPC with Public and Private Subnets** page, set these values:
 - **IPv4 CIDR block:** 10.0.0.0/16
 - **IPv6 CIDR block:** No IPv6 CIDR Block
 - **VPC name:** tutorial-vpc
 - **Public subnet's IPv4 CIDR:** 10.0.0.0/24
 - **Availability Zone:** us-west-2a
 - **Public subnet name:** Tutorial public
 - **Private subnet's IPv4 CIDR:** 10.0.1.0/24
 - **Availability Zone:** us-west-2b
 - **Private subnet name:** Tutorial private 1
 - **Elastic IP Allocation ID:** An Elastic IP address to associate with the NAT gateway
 - **Service endpoints:** Skip this field.
 - **Enable DNS hostnames:** Yes
 - **Hardware tenancy:** Default
7. Choose **Create VPC**.

Create additional subnets

You must have either two private subnets or two public subnets available to create a DB subnet group for a DB instance to use in a VPC. Because the DB instance for this tutorial is private, add a second private subnet to the VPC.

To create an additional subnet

1. Open the Amazon VPC console at <https://console.aws.amazon.com/vpc/>.
2. To add the second private subnet to your VPC, choose **VPC Dashboard**, choose **Subnets**, and then choose **Create subnet**.
3. On the **Create subnet** page, set these values:
 - **VPC ID:** Choose the VPC that you created in the previous step, for example: vpc-*identifier* (tutorial-vpc)
 - **Subnet name:** Tutorial private 2
 - **Availability Zone:** us-west-2c

Note

Choose an Availability Zone that is different from the one that you chose for the first private subnet.

4. Choose **Create subnet**.
5. To ensure that the second private subnet that you created uses the same route table as the first private subnet, complete the following steps:
 - a. Choose **VPC Dashboard**, choose **Subnets**, and then choose the first private subnet that you created for the VPC, Tutorial private 1.
 - b. Below the list of subnets, choose the **Route table** tab, and note the value for **Route Table**—for example: rtb-98b613fd.
 - c. In the list of subnets, deselect the first private subnet.
 - d. In the list of subnets, choose the second private subnet Tutorial private 2, and choose the **Route table** tab.
 - e. If the current route table is not the same as the route table for the first private subnet, choose **Edit route table association**. For **Route table ID**, choose the route table that you noted earlier—for example: rtb-98b613fd. Next, to save your selection, choose **Save**.

Create a VPC security group for a public web server

Next you create a security group for public access. To connect to public instances in your VPC, you add inbound rules to your VPC security group that allow traffic to connect from the internet.

To create a VPC security group

1. Open the Amazon VPC console at <https://console.aws.amazon.com/vpc/>.
2. Choose **VPC Dashboard**, choose **Security Groups**, and then choose **Create security group**.
3. On the **Create security group** page, set these values:
 - **Security group name:** tutorial-securitygroup
 - **Description:** Tutorial Security Group
 - **VPC:** Choose the VPC that you created earlier, for example: vpc-*identifier* (tutorial-vpc)
4. Add inbound rules to the security group.
 - a. Determine the IP address to use to connect to instances in your VPC. To determine your public IP address, in a different browser window or tab, you can use the service at <https://checkip.amazonaws.com>. An example of an IP address is 203.0.113.25/32.

If you are connecting through an Internet service provider (ISP) or from behind your firewall without a static IP address, you need to find out the range of IP addresses used by client computers.

Warning

If you use 0.0.0.0/0, you enable all IP addresses to access your public instances.

This approach is acceptable for a short time in a test environment, but it's unsafe for production environments. In production, you'll authorize only a specific IP address or range of addresses to access your instances.

- b. In the **Inbound rules** section, choose **Add rule**.
- c. Set the following values for your new inbound rule to allow Secure Shell (SSH) access to your EC2 instance. If you do this, you can connect to your EC2 instance to install the web server and other utilities, and to upload content for your web server.
 - **Type:** SSH
 - **Source:** The IP address or range from Step a, for example: 203.0.113.25/32.
- d. Choose **Add rule**.
- e. Set the following values for your new inbound rule to allow HTTP access to your web server.
 - **Type:** HTTP
 - **Source:** 0.0.0.0/0

5. To create the security group, choose **Create security group**.

Note the security group ID because you need it later in this tutorial.

Create a VPC security group for a private DB instance

To keep your DB instance private, create a second security group for private access. To connect to private instances in your VPC, you add inbound rules to your VPC security group that allow traffic from your web server only.

To create a VPC security group

1. Open the Amazon VPC console at <https://console.aws.amazon.com/vpc/>.
2. Choose **VPC Dashboard**, choose **Security Groups**, and then choose **Create security group**.
3. On the **Create security group** page, set these values:
 - **Security group name:** tutorial-db-securitygroup
 - **Description:** Tutorial DB Instance Security Group
 - **VPC:** Choose the VPC that you created earlier, for example: vpc-*identifier* (tutorial-vpc)
4. Add inbound rules to the security group.
 - a. In the **Inbound rules** section, choose **Add rule**.
 - b. Set the following values for your new inbound rule to allow MySQL traffic on port 3306 from your EC2 instance. If you do this, you can connect from your web server to your DB instance to store and retrieve data from your web application to your database.
 - **Type:** MySQL/Aurora
 - **Source:** The identifier of the tutorial-securitygroup security group that you created previously in this tutorial, for example: sg-9edd5cfb.
5. To create the security group, choose **Create security group**.

Create a DB subnet group

A DB subnet group is a collection of subnets that you create in a VPC and that you then designate for your DB instances. A DB subnet group allows you to specify a particular VPC when creating DB instances.

To create a DB subnet group

1. Open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.

Note

Make sure you connect to the Amazon RDS console, not to the Amazon VPC console.

2. In the navigation pane, choose **Subnet groups**.
3. Choose **Create DB Subnet Group**.
4. On the **Create DB subnet group** page, set these values in **Subnet group details**:

- **Name:** tutorial-db-subnet-group
- **Description:** Tutorial DB Subnet Group
- **VPC:** tutorial-vpc (`vpc-identifier`)

5. In the **Add subnets** section, choose the **Availability Zones** and **Subnets**.

For this tutorial, choose `us-west-2b` and `us-west-2c` for the **Availability Zones**. Next, for **Subnets**, choose the subnets for IPv4 CIDR block `10.0.1.0/24` and `10.0.2.0/24`.

6. Choose **Create**.

Your new DB subnet group appears in the DB subnet groups list on the RDS console. You can click the DB subnet group to see details, including all of the subnets associated with the group, in the details pane at the bottom of the window.

Note

If you created this VPC to complete [Tutorial: Create a web server and an Amazon Aurora DB cluster \(p. 103\)](#), create the DB cluster by following the instructions in [Create an Amazon Aurora DB cluster \(p. 104\)](#).

Deleting the VPC

After you create the VPC and other resources for this tutorial, you can delete them if they are no longer needed.

Note

If you added resources in the Amazon VPC you created for this tutorial, such as Amazon EC2 instances or Amazon RDS DB instances, you might need to delete these resources before you can delete the VPC. For more information, see [Delete your VPC](#) in the *Amazon VPC User Guide*.

To delete a VPC and related resources

1. Delete the DB subnet group.
 - a. Open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
 - b. In the navigation pane, choose **Subnet groups**.
 - c. Select the DB subnet group you want to delete, such as `tutorial-db-subnet-group`.
 - d. Choose **Delete**, and then choose **Delete** in the confirmation window.
2. Note the VPC ID.
 - a. Open the Amazon VPC console at <https://console.aws.amazon.com/vpc/>.
 - b. Choose **VPC Dashboard**, and then choose **VPCs**.
 - c. In the list, identify the VPC you created, such as `tutorial-vpc`.
 - d. Note the **VPC ID** of the VPC you created. You will need the VPC ID in subsequent steps.
3. Delete the security groups.
 - a. Open the Amazon VPC console at <https://console.aws.amazon.com/vpc/>.

- b. Choose **VPC Dashboard**, and then choose **Security Groups**.
 - c. Select the security group for the Amazon RDS DB instance, such as **tutorial-db-securitygroup**.
 - d. From **Actions**, choose **Delete security groups**, and then choose **Delete** on the confirmation page.
 - e. On the **Security Groups** page, select the security group for the Amazon EC2 instance, such as **tutorial-securitygroup**.
 - f. From **Actions**, choose **Delete security groups**, and then choose **Delete** on the confirmation page.
4. Delete the NAT gateway.
 - a. Open the Amazon VPC console at <https://console.aws.amazon.com/vpc/>.
 - b. Choose **VPC Dashboard**, and then choose **NAT Gateways**.
 - c. Select the NAT gateway of the VPC you created. Use the VPC ID to identify the correct NAT gateway.
 - d. From **Actions**, choose **Delete NAT gateway**.
 - e. On the confirmation page, enter **delete**, and then choose **Delete**.
 5. Delete the VPC.
 - a. Open the Amazon VPC console at <https://console.aws.amazon.com/vpc/>.
 - b. Choose **VPC Dashboard**, and then choose **VPCs**.
 - c. Select the VPC you want to delete, such as **tutorial-vpc**.
 - d. From **Actions**, choose **Delete VPC**.

The confirmation page shows other resources that are associated with the VPC that will also be deleted, including the subnets associated with it.

 - e. On the confirmation page, enter **delete**, and then choose **Delete**.
 6. Release the Elastic IP addresses.
 - a. Open the Amazon EC2 console at <https://console.aws.amazon.com/ec2/>.
 - b. Choose **EC2 Dashboard**, and then choose **Elastic IPs**.
 - c. Select the Elastic IP address you want to release.
 - d. From **Actions**, choose **Release Elastic IP addresses**.
 - e. On the confirmation page, choose **Release**.

Quotas and constraints for Amazon Aurora

Following, you can find a description of the resource quotas and naming constraints for Amazon Aurora.

Topics

- [Quotas in Amazon Aurora \(p. 1811\)](#)
- [Naming constraints in Amazon Aurora \(p. 1812\)](#)
- [Amazon Aurora size limits \(p. 1813\)](#)

Quotas in Amazon Aurora

Each AWS account has quotas, for each AWS Region, on the number of Amazon Aurora resources that can be created. After a quota for a resource has been reached, additional calls to create that resource fail with an exception.

The following table lists the resources and their quotas per AWS Region.

Name	Default	Adjustable
Authorizations per DB security group	Each supported Region: 20	No
DB instances	Each supported Region: 40	Yes
DB subnet groups	Each supported Region: 50	Yes
Event subscriptions	Each supported Region: 20	Yes
IAM roles per DB cluster	Each supported Region: 5	Yes
IAM roles per DB instance	Each supported Region: 5	Yes
Manual DB cluster snapshots	Each supported Region: 100	Yes
Manual DB instance snapshots	Each supported Region: 100	Yes
Option groups	Each supported Region: 20	Yes
Parameter groups	Each supported Region: 50	Yes
Proxies	Each supported Region: 20	Yes
Read replicas per master	Each supported Region: 5	Yes
Reserved DB instances	Each supported Region: 40	Yes
Rules per security group	Each supported Region: 20	No
Security groups	Each supported Region: 25	Yes

Name	Default	Adjustable
Security groups (VPC)	Each supported Region: 5	No
Subnets per DB subnet group	Each supported Region: 20	No
Tags per resource	Each supported Region: 50	No
Total storage for all DB instances	Each supported Region: 100,000 Gigabytes	Yes

Note

By default, you can have up to a total of 40 DB instances. RDS DB instances, Aurora DB instances, Amazon Neptune instances, and Amazon DocumentDB instances apply to this quota. If your application requires more DB instances, you can request additional DB instances by opening the [Service Quotas console](#). In the navigation pane, choose **AWS services**. Choose **Amazon Relational Database Service (Amazon RDS)**, choose a quota, and follow the directions to request a quota increase. For more information, see [Requesting a quota increase](#) in the *Service Quotas User Guide*.

Backups managed by AWS Backup are considered manual DB cluster snapshots, but don't count toward the manual cluster snapshot quota. For information about AWS Backup, see the [AWS Backup Developer Guide](#).

If you use any of the Amazon RDS APIs and exceed the default quota for the number of calls per second, the Amazon RDS API issues an error similar to the following: ClientError: An error occurred (ThrottlingException) when calling the *API_name* operation: Rate exceeded. Reduce the number of calls per second. The quota is meant to cover most use cases. If higher limits are needed, request a quota increase by contacting AWS Support. Open the [AWS Support Center](#) page, sign in if necessary, and choose **Create case**. Choose **Service limit increase**. Complete and submit the form.

Note

This quota can't be changed in the Amazon RDS Service Quotas console.

Naming constraints in Amazon Aurora

The following table describes naming constraints in Amazon Aurora.

Resource or item	Constraints
DB cluster identifier	Identifiers have these naming constraints: <ul style="list-style-type: none"> Must contain 1–63 alphanumeric characters or hyphens. First character must be a letter. Can't end with a hyphen or contain two consecutive hyphens. Must be unique for all DB instances per AWS account, per AWS Region.
Initial database name	Database name constraints differ between Aurora MySQL and PostgreSQL. For more information, see the available settings when creating each DB cluster.
Master user name	Master user name constraints differ for each database engine. For more information, see the available settings when creating each DB cluster.

Resource or item	Constraints
Master password	The password for the database master user can include any printable ASCII character except /, ", @, or a space. Master password length constraints differ for each database engine. For more information, see the available settings when creating each DB cluster.
DB parameter group name	These names have these constraints: <ul style="list-style-type: none">• Must contain 1–255 alphanumeric characters.• First character must be a letter.• Hyphens are allowed, but the name cannot end with a hyphen or contain two consecutive hyphens.
DB subnet group name	These names have these constraints: <ul style="list-style-type: none">• Must contain 1–255 characters.• Alphanumeric characters, spaces, hyphens, underscores, and periods are allowed.

Amazon Aurora size limits

Storage size limits

An Aurora cluster volume can grow to a maximum size of 128 tebibytes (TiB) for the following engine versions:

- Aurora MySQL versions 3.1 and higher (compatible with MySQL 8.0), 2.09 and higher (compatible with MySQL 5.7), and 1.23 and higher (compatible with MySQL 5.6)
- All Aurora PostgreSQL 13 versions, Aurora PostgreSQL versions 12.4 and higher, 11.7 and higher, 10.12 and higher, and 9.6.17 and higher

For lower engine versions, the maximum size of an Aurora cluster volume is 64 TiB. For more information, see [How Aurora storage automatically resizes \(p. 65\)](#).

SQL table size limits

For Aurora MySQL, the maximum table size is 64 tebibytes (TiB). For an Aurora PostgreSQL DB cluster, the maximum table size is 32 tebibytes (TiB). We recommend that you follow table design best practices, such as partitioning of large tables.

Troubleshooting for Aurora

Use the following sections to help troubleshoot problems you have with DB instances in Amazon RDS and Aurora.

Topics

- [Can't connect to Amazon RDS DB instance \(p. 1814\)](#)
- [Amazon RDS security issues \(p. 1816\)](#)
- [Resetting the DB instance owner password \(p. 1816\)](#)
- [Amazon RDS DB instance outage or reboot \(p. 1816\)](#)
- [Amazon RDS DB parameter changes not taking effect \(p. 1817\)](#)
- [Amazon Aurora MySQL out of memory issues \(p. 1817\)](#)
- [Amazon Aurora MySQL replication issues \(p. 1818\)](#)

For information about debugging problems using the Amazon RDS API, see [Troubleshooting applications on Aurora \(p. 1823\)](#).

Can't connect to Amazon RDS DB instance

When you can't connect to a DB instance, the following are common causes:

- **Inbound rules** – The access rules enforced by your local firewall and the IP addresses authorized to access your DB instance might not match. The problem is most likely the inbound rules in your security group.

By default, DB instances don't allow access. Access is granted through a security group associated with the VPC that allows traffic into and out of the DB instance. If necessary, add inbound and outbound rules for your particular situation to the security group. You can specify an IP address, a range of IP addresses, or another VPC security group.

Note

When adding a new inbound rule, you can choose **My IP** for **Source** to allow access to the DB instance from the IP address detected in your browser.

For more information about setting up security groups, see [Provide access to the DB cluster in the VPC by creating a security group \(p. 87\)](#).

Note

Client connections from IP addresses within the range 169.254.0.0/16 aren't permitted. This is the Automatic Private IP Addressing Range (APIPA), which is used for local-link addressing.

- **Public accessibility** – To connect to your DB instance from outside of the VPC, such as by using a client application, the instance must have a public IP address assigned to it.

To make the instance publicly accessible, modify it and choose **Yes** under **Public accessibility**. For more information, see [Hiding a DB instance in a VPC from the internet \(p. 1789\)](#).

- **Port** – The port that you specified when you created the DB instance can't be used to send or receive communications due to your local firewall restrictions. To determine if your network allows the specified port to be used for inbound and outbound communication, check with your network administrator.

- **Availability** – For a newly created DB instance, the DB instance has a status of *creating* until the DB instance is ready to use. When the state changes to *available*, you can connect to the DB instance. Depending on the size of your DB instance, it can take up to 20 minutes before an instance is available.
- **Internet gateway** – For a DB instance to be publicly accessible, the subnets in its DB subnet group must have an internet gateway.

To configure an internet gateway for a subnet

1. Sign in to the AWS Management Console and open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. In the navigation pane, choose **Databases**, and then choose the name of the DB instance.
3. In the **Connectivity & security** tab, write down the values of the VPC ID under **VPC** and the subnet ID under **Subnets**.
4. Open the Amazon VPC console at <https://console.aws.amazon.com/vpc/>.
5. In the navigation pane, choose **Internet Gateways**. Verify that there is an internet gateway attached to your VPC. Otherwise, choose **Create Internet Gateway** to create an internet gateway. Select the internet gateway, and then choose **Attach to VPC** and follow the directions to attach it to your VPC.
6. In the navigation pane, choose **Subnets**, and then select your subnet.
7. On the **Route Table** tab, verify that there is a route with `0.0.0.0/0` as the destination and the internet gateway for your VPC as the target.
 - a. Choose the ID of the route table (`rtb-xxxxxx`) to navigate to the route table.
 - b. On the **Routes** tab, choose **Edit routes**. Choose **Add route**, use `0.0.0.0/0` as the destination and the internet gateway as the target.
 - c. Choose **Save routes**.

For more information, see [Working with a DB instance in a VPC \(p. 1787\)](#).

Testing a connection to a DB instance

You can test your connection to a DB instance using common Linux or Microsoft Windows tools.

From a Linux or Unix terminal, you can test the connection by entering the following (replace `DB-instance-endpoint` with the endpoint and `port` with the port of your DB instance).

```
nc -zv DB-instance-endpoint port
```

For example, the following shows a sample command and the return value.

```
nc -zv postgresql1.c6c8mn7fake0.us-west-2.rds.amazonaws.com 8299
Connection to postgresql1.c6c8mn7fake0.us-west-2.rds.amazonaws.com 8299 port [tcp/vvr-data] succeeded!
```

Windows users can use Telnet to test the connection to a DB instance. Telnet actions aren't supported other than for testing the connection. If a connection is successful, the action returns no message. If a connection isn't successful, you receive an error message such as the following.

```
C:\>telnet sg-postgresql1.c6c8mn7fake0.us-west-2.rds.amazonaws.com 819
Connecting To sg-postgresql1.c6c8mn7fake0.us-west-2.rds.amazonaws.com...Could not open
connection to the host, on port 819: Connect failed
```

If Telnet actions return success, your security group is properly configured.

Note

Amazon RDS doesn't accept internet control message protocol (ICMP) traffic, including ping.

Troubleshooting connection authentication

If you can connect to your DB instance but you get authentication errors, you might want to reset the master user password for the DB instance. You can do this by modifying the RDS instance.

Amazon RDS security issues

To avoid security issues, never use your master AWS user name and password for a user account. Best practice is to use your master AWS account to create AWS Identity and Access Management (IAM) users and assign those to DB user accounts. You can also use your master account to create other user accounts, if necessary.

For more information on creating IAM users, see [Create an IAM user \(p. 84\)](#).

Error message "failed to retrieve account attributes, certain console functions may be impaired."

You can get this error for several reasons. It might be because your account is missing permissions, or your account hasn't been properly set up. If your account is new, you might not have waited for the account to be ready. If this is an existing account, you might lack permissions in your access policies to perform certain actions such as creating a DB instance. To fix the issue, your IAM administrator needs to provide the necessary roles to your account. For more information, see [the IAM documentation](#).

Resetting the DB instance owner password

If you get locked out of your DB cluster, you can log in as the master user. Then you can reset the credentials for other administrative users or roles. If you can't log in as the master user, the AWS account owner can reset the master user password. For details of which administrative accounts or roles you might need to reset, see [Master user account privileges \(p. 1782\)](#).

You can change the DB instance password by using the Amazon RDS console, the AWS CLI command `modify-db-instance`, or by using the `ModifyDBInstance` API operation. For more information about modifying a DB instance in a DB cluster, see [Modify a DB instance in a DB cluster \(p. 373\)](#).

Amazon RDS DB instance outage or reboot

A DB instance outage can occur when a DB instance is rebooted. It can also occur when the DB instance is put into a state that prevents access to it, and when the database is restarted. A reboot can occur when you either manually reboot your DB instance or change a DB instance setting that requires a reboot before it can take effect.

A DB instance reboot occurs when you change a setting that requires a reboot, or when you manually cause a reboot. A reboot can occur immediately if you change a setting and request that the change take effect immediately or it can occur during the DB instance's maintenance window.

A DB instance reboot occurs immediately when one of the following occurs:

- You change the backup retention period for a DB instance from 0 to a nonzero value or from a nonzero value to 0 and set **Apply Immediately** to true.
- You change the DB instance class, and **Apply Immediately** is set to true.

A DB instance reboot occurs during the maintenance window when one of the following occurs:

- You change the backup retention period for a DB instance from 0 to a nonzero value or from a nonzero value to 0, and **Apply Immediately** is set to false.
- You change the DB instance class, and **Apply Immediately** is set to false.

When you change a static parameter in a DB parameter group, the change doesn't take effect until the DB instance associated with the parameter group is rebooted. The change requires a manual reboot. The DB instance isn't automatically rebooted during the maintenance window.

Amazon RDS DB parameter changes not taking effect

In some cases, you might change a parameter in a DB parameter group but don't see the changes take effect. If so, you likely need to reboot the DB instance associated with the DB parameter group. When you change a dynamic parameter, the change takes effect immediately. When you change a static parameter, the change doesn't take effect until you reboot the DB instance associated with the parameter group.

You can reboot a DB instance using the RDS console or explicitly calling the [RebootDBInstance](#) API operation (without failover, if the DB instance is in a Multi-AZ deployment). The requirement to reboot the associated DB instance after a static parameter change helps mitigate the risk of a parameter misconfiguration affecting an API call. An example of this might be calling [ModifyDBInstance](#) to change the DB instance class. For more information, see [Modifying parameters in a DB parameter group \(p. 347\)](#).

Amazon Aurora MySQL out of memory issues

The Aurora MySQL `aurora_oom_response` instance-level parameter can enable the DB instance to monitor the system memory and estimate the memory consumed by various statements and connections. If the system runs low on memory, it can perform a list of actions to release that memory in an attempt to avoid out-of-memory (OOM) and database restart. The instance-level parameter takes a string of comma-separated actions that a DB instance should take when its memory is low. Valid actions include `print`, `tune`, `decline`, `kill_query`, or any combination of these. An empty string means that no action should be taken and effectively disables the feature.

Note

This parameter only applies to Aurora MySQL version 1.18 and higher. It isn't used in Aurora MySQL version 2.

The following are usage examples for the `aurora_oom_response` parameter:

- `print` – Only prints the queries taking high amount of memory.
- `tune` – Tunes the internal table caches to release some memory back to the system.
- `decline` – Declines new queries once the instance is low on memory.

- `kill_query` – Ends the queries in descending order of memory consumption until the instance memory surfaces above the low threshold. Data definition language (DDL) statements aren't ended.
- `print`, `tune` – Performs actions described for both `print` and `tune`.
- `tune`, `decline`, `kill_query` – Performs the actions described for `tune`, `decline`, and `kill_query`.

For the db.t2.small DB instance class, the `aurora_oom_response` parameter is set to `print`, `tune` by default. For all other DB instance classes, the parameter value is empty by default (disabled).

Amazon Aurora MySQL replication issues

Some MySQL replication issues also apply to Aurora MySQL. You can diagnose and correct these.

Topics

- [Diagnosing and resolving lag between read replicas \(p. 1818\)](#)
- [Diagnosing and resolving a MySQL read replication failure \(p. 1819\)](#)
- [Replication stopped error \(p. 1820\)](#)

Diagnosing and resolving lag between read replicas

After you create a MySQL read replica and the replica is available, Amazon RDS first replicates the changes made to the source DB instance from the time the read replica create operation started. During this phase, the replication lag time for the read replica is greater than 0. You can monitor this lag time in Amazon CloudWatch by viewing the Amazon RDS `AuroraBinlogReplicaLag` metric.

The `AuroraBinlogReplicaLag` metric reports the value of the `Seconds_Behind_Master` field of the MySQL `SHOW SLAVE STATUS` command. For more information, see [SHOW SLAVE STATUS](#). When the `AuroraBinlogReplicaLag` metric reaches 0, the replica has caught up to the source DB instance. If the `AuroraBinlogReplicaLag` metric returns -1, replication might not be active. To troubleshoot a replication error, see [Diagnosing and resolving a MySQL read replication failure \(p. 1819\)](#). A `AuroraBinlogReplicaLag` value of -1 can also mean that the `Seconds_Behind_Master` value can't be determined or is `NULL`.

Note

Previous versions of Aurora MySQL used `SHOW SLAVE STATUS` instead of `SHOW REPLICA STATUS`. If you are using Aurora MySQL version 1 or 2, then use `SHOW SLAVE STATUS`. Use `SHOW REPLICA STATUS` for Aurora MySQL version 3 and higher.

The `AuroraBinlogReplicaLag` metric returns -1 during a network outage or when a patch is applied during the maintenance window. In this case, wait for network connectivity to be restored or for the maintenance window to end before you check the `AuroraBinlogReplicaLag` metric again.

The MySQL read replication technology is asynchronous. Thus, you can expect occasional increases for the `BinLogDiskUsage` metric on the source DB instance and for the `AuroraBinlogReplicaLag` metric on the read replica. For example, consider a situation where a high volume of write operations to the source DB instance occur in parallel. At the same time, write operations to the read replica are serialized using a single I/O thread. Such a situation can lead to a lag between the source instance and read replica.

For more information about read replicas and MySQL, see [Replication implementation details](#) in the MySQL documentation.

You can reduce the lag between updates to a source DB instance and the subsequent updates to the read replica by doing the following:

- Set the DB instance class of the read replica to have a storage size comparable to that of the source DB instance.
- Make sure that parameter settings in the DB parameter groups used by the source DB instance and the read replica are compatible. For more information and an example, see the discussion of the `max_allowed_packet` parameter in the next section.
- Disable the query cache. For tables that are modified often, using the query cache can increase replica lag because the cache is locked and refreshed often. If this is the case, you might see less replica lag if you disable the query cache. You can disable the query cache by setting the `query_cache_type` parameter to 0 in the DB parameter group for the DB instance. For more information on the query cache, see [Query cache configuration](#).
- Warm the buffer pool on the read replica for InnoDB for MySQL. For example, suppose that you have a small set of tables that are being updated often and you're using the InnoDB or XtraDB table schema. In this case, dump those tables on the read replica. Doing this causes the database engine to scan through the rows of those tables from the disk and then cache them in the buffer pool. This approach can reduce replica lag. The following shows an example.

For Linux, macOS, or Unix:

```
PROMPT> mysqldump \
-h <endpoint> \
--port=<port> \
-u=<username> \
-p <password> \
database_name table1 table2 > /dev/null
```

For Windows:

```
PROMPT> mysqldump ^
-h <endpoint> ^
--port=<port> ^
-u=<username> ^
-p <password> ^
database_name table1 table2 > /dev/null
```

Diagnosing and resolving a MySQL read replication failure

Amazon RDS monitors the replication status of your read replicas and updates the **Replication State** field of the read replica instance to **Error** if replication stops for any reason. You can review the details of the associated error thrown by the MySQL engines by viewing the **Replication Error** field. Events that indicate the status of the read replica are also generated, including [RDS-EVENT-0045 \(p. 679\)](#), [RDS-EVENT-0046 \(p. 679\)](#), and [RDS-EVENT-0047 \(p. 678\)](#). For more information about events and subscribing to events, see [Using Amazon RDS event notification \(p. 675\)](#). If a MySQL error message is returned, check the error in the [MySQL error message documentation](#).

Common situations that can cause replication errors include the following:

- The value for the `max_allowed_packet` parameter for a read replica is less than the `max_allowed_packet` parameter for the source DB instance.

The `max_allowed_packet` parameter is a custom parameter that you can set in a DB parameter group. The `max_allowed_packet` parameter is used to specify the maximum size of data manipulation language (DML) that can be run on the database. If the `max_allowed_packet` value for the source DB instance is larger than the `max_allowed_packet` value for the read replica, the replication process can throw an error and stop replication. The most common error is `packet`

bigger than 'max_allowed_packet' bytes. You can fix the error by having the source and read replica use DB parameter groups with the same `max_allowed_packet` parameter values.

- Writing to tables on a read replica. If you're creating indexes on a read replica, you need to have the `read_only` parameter set to `0` to create the indexes. If you're writing to tables on the read replica, it can break replication.
- Using a nontransactional storage engine such as MyISAM. Read replicas require a transactional storage engine. Replication is only supported for the following storage engines: InnoDB for MySQL or MariaDB.

You can convert a MyISAM table to InnoDB with the following command:

```
alter table <schema>.<table_name> engine=innodb;
```

- Using unsafe nondeterministic queries such as `SYSDATE()`. For more information, see [Determination of safe and unsafe statements in binary logging](#) in the MySQL documentation.

The following steps can help resolve your replication error:

- If you encounter a logical error and you can safely skip the error, follow the steps described in [Skipping the current replication error](#). Your Aurora MySQL DB instance must be running a version that includes the `mysql_rds_skip_repl_error` procedure. For more information, see [mysql_rds_skip_repl_error](#).
- If you encounter a binary log (binlog) position issue, you can change the replica replay position with the `mysql.rds_next_master_log` (Aurora MySQL version 1 and 2) or `mysql.rds_next_source_log` (Aurora MySQL version 3 and higher) command. Your Aurora MySQL DB instance must be running a version that supports this command to change the replica replay position. For version information, see [mysql_rds_next_master_log](#).
- If you encounter a temporary performance issue due to high DML load, you can set the `innodb_flush_log_at_trx_commit` parameter to `2` in the DB parameter group on the read replica. Doing this can help the read replica catch up, though it temporarily reduces atomicity, consistency, isolation, and durability (ACID).
- You can delete the read replica and create an instance using the same DB instance identifier so that the endpoint remains the same as that of your old read replica.

If a replication error is fixed, the **Replication State** changes to **replicating**. For more information, see [Troubleshooting a MySQL read replica problem](#).

Replication stopped error

When you call the `mysql.rds_skip_repl_error` command, you might receive an error message stating that replication is down or disabled.

This error message appears because replication is stopped and can't be restarted.

If you need to skip a large number of errors, the replication lag can increase beyond the default retention period for binary log files. In this case, you might encounter a fatal error due to binary log files being purged before they have been replayed on the replica. This purge causes replication to stop, and you can no longer call the `mysql.rds_skip_repl_error` command to skip replication errors.

You can mitigate this issue by increasing the number of hours that binary log files are retained on your replication source. After you have increased the binlog retention time, you can restart replication and call the `mysql.rds_skip_repl_error` command as needed.

To set the binlog retention time, use the `mysql.rds_set_configuration` procedure. Specify a configuration parameter of 'binlog retention hours' along with the number of hours to retain binlog files on the DB cluster, up to 2160 (90 days). The default for Aurora MySQL is 24 (1 day). The following example sets the retention period for binlog files to 48 hours.

```
CALL mysql.rds_set_configuration('binlog retention hours', 48);
```

Amazon RDS application programming interface (API) reference

In addition to the AWS Management Console, and the AWS Command Line Interface (AWS CLI), Amazon Relational Database Service (Amazon RDS) also provides an application programming interface (API). You can use the API to automate tasks for managing your DB instances and other objects in Amazon RDS.

- For an alphabetical list of API operations, see [Actions](#).
- For an alphabetical list of data types, see [Data types](#).
- For a list of common query parameters, see [Common parameters](#).
- For descriptions of the error codes, see [Common errors](#).

For more information about the AWS CLI, see [AWS Command Line Interface reference for Amazon RDS](#).

Topics

- [Using the Query API \(p. 1822\)](#)
- [Troubleshooting applications on Aurora \(p. 1823\)](#)

Using the Query API

The following sections briefly discuss the parameters and request authentication used with the Query API.

For general information about how the Query API works, see [Query requests](#) in the *Amazon EC2 API Reference*.

Query parameters

HTTP Query-based requests are HTTP requests that use the HTTP verb GET or POST and a Query parameter named `Action`.

Each Query request must include some common parameters to handle authentication and selection of an action.

Some operations take lists of parameters. These lists are specified using the `param.n` notation. Values of `n` are integers starting from 1.

For information about Amazon RDS regions and endpoints, go to [Amazon Relational Database Service \(RDS\)](#) in the Regions and Endpoints section of the *Amazon Web Services General Reference*.

Query request authentication

You can only send Query requests over HTTPS, and you must include a signature in every Query request. You must use either AWS signature version 4 or signature version 2. For more information, see [Signature Version 4 signing process](#) and [Signature version 2 signing process](#).

Troubleshooting applications on Aurora

Amazon RDS provides specific and descriptive errors to help you troubleshoot problems while interacting with the Amazon RDS API.

Topics

- [Retrieving errors \(p. 1823\)](#)
- [Troubleshooting tips \(p. 1823\)](#)

For information about troubleshooting for Amazon RDS DB instances, see [Troubleshooting for Aurora \(p. 1814\)](#).

Retrieving errors

Typically, you want your application to check whether a request generated an error before you spend any time processing results. The easiest way to find out if an error occurred is to look for an `Error` node in the response from the Amazon RDS API.

XPath syntax provides a simple way to search for the presence of an `Error` node, as well as an easy way to retrieve the error code and message. The following code snippet uses Perl and the `XML::XPath` module to determine if an error occurred during a request. If an error occurred, the code prints the first error code and message in the response.

```
use XML::XPath;
my $xp = XML::XPath->new(xml =>$response);
if ( $xp->find("//Error") )
{print "There was an error processing your request:\n", " Error code: ",
$xp->findvalue("//Error[1]/Code"), "\n", " ",
$xp->findvalue("//Error[1]/Message"), "\n\n"; }
```

Troubleshooting tips

We recommend the following processes to diagnose and resolve problems with the Amazon RDS API.

- Verify that Amazon RDS is operating normally in the AWS Region you are targeting by visiting <http://status.aws.amazon.com>.
- Check the structure of your request

Each Amazon RDS operation has a reference page in the *Amazon RDS API Reference*. Double-check that you are using parameters correctly. In order to give you ideas regarding what might be wrong, look at the sample requests or user scenarios to see if those examples are doing similar operations.

- Check the forum

Amazon RDS has a development community forum where you can search for solutions to problems others have experienced along the way. To view the forum, go to [AWS Discussion Forums](#).

Document history

Current API version: 2014-10-31

The following table describes important changes to the *Amazon Aurora User Guide*. For notification about updates to this documentation, you can subscribe to an RSS feed. For information about Amazon Relational Database Service (Amazon RDS), see the [Amazon Relational Database Service User Guide](#).

Note

Before August 31, 2018, Amazon Aurora was documented in the *Amazon Relational Database Service User Guide*. For earlier Aurora document history, see [Document history](#) in the *Amazon Relational Database Service User Guide*.

You can filter new Amazon Aurora features on the [What's New with Database?](#) page. For **Products**, choose **Amazon Aurora**. Then search using keywords such as **global database** or **Serverless**.

update-history-change	update-history-description	update-history-date
Aurora PostgreSQL 13.5 supports Babelfish for Aurora PostgreSQL 1.1.0 (p. 1824)	Babelfish 1.1.0 is supported in Aurora PostgreSQL 13.5. For more information, see Babelfish version 1.1.0 .	February 28, 2022
Aurora PostgreSQL releases 13.5, 12.9, 11.14, and 10.19 (p. 1824)	New releases of Amazon Aurora PostgreSQL-Compatible Edition support compatibility with PostgreSQL 13.5, PostgreSQL 12.9, PostgreSQL 11.14, and PostgreSQL 10.19. For more information, see Database engine versions for Amazon Aurora PostgreSQL .	February 25, 2022
Amazon Aurora supports Database Activity Streams in the Asia Pacific (Jakarta) Region (p. 1824)	For more information, see Support for AWS Regions for database activity streams .	February 16, 2022
Aurora MySQL version 2.10.2 (p. 1110)	Aurora MySQL version 2.10.2 is available.	January 26, 2022
Performance Insights supports new APIs (p. 1824)	Performance Insights supports the following APIs: <code>GetResourceMetadata</code> , <code>ListAvailableResourceDimensions</code> , and <code>ListAvailableResourceMetrics</code> . For more information, see Retrieving metrics with the Performance Insights API in this manual and the Amazon RDS Performance Insights API Reference .	January 12, 2022
RDS Proxy supports events (p. 1824)	RDS Proxy now generates events that you can subscribe	January 11, 2022

	<p>to and view in CloudWatch Events or configure to send to Amazon EventBridge. For more information, see Working with RDS Proxy events.</p>	
Aurora MySQL version 2.08.4 (p. 1131)	Aurora MySQL version 2.08.4 is available.	January 6, 2022
RDS Proxy available in additional AWS Regions (p. 1824)	RDS Proxy is now available in the following Regions: Africa (Cape Town), Asia Pacific (Hong Kong), Asia Pacific (Osaka), Europe (Milan), Europe (Paris), Europe (Stockholm), Middle East (Bahrain), and South America (São Paulo). For more information about RDS Proxy, see Using Amazon RDS Proxy .	January 5, 2022
Amazon Aurora available in the Asia Pacific (Jakarta) Region (p. 1824)	Aurora is now available in the Asia Pacific (Jakarta) Region. For more information, see Regions and Availability Zones .	December 13, 2021
DevOps Guru for Amazon RDS provides detailed insights and recommendations for Amazon Aurora (p. 1824)	DevOps Guru for RDS mines Performance Insights for performance-related data. Using this data, the service analyzes the performance of your Amazon Aurora DB instances and can help you resolve performance issues. To learn more, see Analyzing performance anomalies with DevOps Guru for RDS in this guide and see Overview of DevOps Guru for RDS in the Amazon DevOps Guru User Guide .	December 1, 2021
Aurora MySQL version 2.07.7 (p. 1140)	Aurora MySQL version 2.07.7 is available.	November 24, 2021
Aurora PostgreSQL supports RDS Proxy with PostgreSQL 12 (p. 1824)	You can now create an RDS Proxy with an Aurora PostgreSQL 12 database cluster. For more information about RDS Proxy, see Using Amazon RDS Proxy .	November 22, 2021
Aurora MySQL version 3.01.0, compatible with MySQL 8.0.23 (p. 1108)	Aurora MySQL version 3.01.0 is available. This version is compatible with MySQL 8.0.23. For full details, see Aurora MySQL version 3 compatible with MySQL 8.0 .	November 18, 2021
Aurora MySQL version 2.09.3 (p. 1119)	Aurora MySQL version 2.09.3 is available.	November 12, 2021

Aurora supports AWS Graviton2 instance classes for Database Activity Streams (p. 1824)	You can use database activity streams with the db.r6g instance class for Aurora MySQL and Aurora PostgreSQL. For more information, see Supported DB instance classes .	November 3, 2021
Amazon Aurora support for cross-account AWS KMS keys (p. 1824)	You can use a KMS key from a different AWS account for encryption when exporting DB snapshots to Amazon S3. For more information, see Exporting DB snapshot data to Amazon S3 .	November 3, 2021
Aurora PostgreSQL releases 13.4, 12.8, 11.13, and 10.18 (p. 1824)	New releases of Amazon Aurora PostgreSQL-Compatible Edition support compatibility with PostgreSQL 13.4, PostgreSQL 12.8, PostgreSQL 11.13, and PostgreSQL 10.18. For more information, see Database engine versions for Amazon Aurora PostgreSQL .	October 28, 2021
Amazon Aurora supports Babelfish for Aurora PostgreSQL (p. 1824)	Babelfish for Aurora PostgreSQL extends your Amazon Aurora PostgreSQL-Compatible Edition database with the ability to accept database connections from Microsoft SQL Server clients. For more information, see Working with Babelfish for Aurora PostgreSQL .	October 28, 2021
Aurora MySQL version 2.10.1 (p. 1113)	Aurora MySQL version 2.10.1 is available.	October 21, 2021
Amazon Aurora supports Performance Insights in additional AWS Regions (p. 1824)	Performance Insights is available in the Middle East (Bahrain), Africa (Cape Town), Europe (Milan), and Asia Pacific (Osaka) Regions. For more information, see AWS Region support for Performance Insights .	October 5, 2021
Aurora MySQL version 1.23.4 (p. 1197)	Aurora MySQL version 1.23.4 is available.	September 30, 2021

Configurable autoscaling timeout for Aurora Serverless (p. 1824)	You can choose how long Aurora Serverless waits to find an autoscaling point. If no autoscaling point is found during that period, Aurora Serverless cancels the scaling event or forces the capacity change, depending on the timeout action that you selected. For more information, see Autoscaling for Aurora Serverless .	September 10, 2021
Aurora supports X2g and T4g instance classes (p. 1824)	Both Aurora MySQL and Aurora PostgreSQL can now use X2g and T4g instance classes. The instance classes that you can use depend on the version of Aurora MySQL or Aurora PostgreSQL. For information about supported instance types, see DB instance classes .	September 10, 2021
Aurora MySQL version 2.07.6 (p. 1142)	Aurora MySQL version 2.07.6 is available.	September 2, 2021
Aurora PostgreSQL supports version 13.3 (p. 1824)	Aurora PostgreSQL now supports PostgreSQL version 13.3. For more information, see Engine versions for Amazon Aurora PostgreSQL . The release supports Intel based instance types R5 and T3, and deprecates R4 instance types. For information about supported instance types, see DB instance classes .	August 26, 2021
Aurora PostgreSQL releases 12.7, 11.12, 10.17, and 9.6.22 (p. 1824)	New releases of Amazon Aurora PostgreSQL-Compatible Edition support compatibility with PostgreSQL 12.7, PostgreSQL 11.12, PostgreSQL 10.17, and PostgreSQL 9.6.22 For more information, see Database engine versions for Amazon Aurora PostgreSQL .	August 19, 2021
Amazon RDS supports RDS Proxy in a shared VPC (p. 1824)	You can now create an RDS Proxy in a shared VPC. For more information about RDS Proxy, see "Managing Connections with Amazon RDS Proxy" in the Amazon RDS User Guide or the Aurora User Guide .	August 6, 2021

Aurora MySQL Serverless version 1.22.3 (p. 1249)	Aurora Serverless with MySQL 5.6 compatibility is available. The release includes features and bug fixes based on Aurora MySQL version 1.22.3. For more information about Aurora Serverless, see Using Amazon Aurora Serverless .	July 16, 2021
Aurora version policy page (p. 1824)	The <i>Amazon Aurora User Guide</i> now includes a section with general information about Aurora versions and associated policies. For details, see Amazon Aurora versions .	July 14, 2021
Aurora MySQL version 2.07.5 (p. 1143)	Aurora MySQL version 2.07.5 is available.	July 6, 2021
Exclude Data API events from an AWS CloudTrail trail (p. 1824)	You can exclude Data API events from a CloudTrail trail. For more information, see Excluding Data API events from an AWS CloudTrail trail .	July 2, 2021
Aurora MySQL version 1.23.3 (p. 1198)	Aurora MySQL version 1.23.3 is available.	June 28, 2021
Aurora PostgreSQL releases 4.1.0, 3.5.0, 2.8.0, and 1.10.0 compatible with PostgreSQL 12.6, 11.11, 10.16, and 9.6.21 (p. 1824)	New versions of Amazon Aurora PostgreSQL-Compatible Edition include 4.1.0 (compatible with PostgreSQL 12.6), 3.5.0 (compatible with PostgreSQL 11.11), 2.8.0 (compatible with PostgreSQL 10.16), and 1.10.0 (compatible with PostgreSQL 9.6.21). For more information, see Database engine versions for Amazon Aurora PostgreSQL-Compatible Edition .	June 17, 2021
Amazon Aurora PostgreSQL-Compatible Edition supports additional extensions (p. 1824)	Newly supported extensions include pg_bigm, pg_cron, pg_partman, and pg_proctab. For more information, see Extension versions for Amazon Aurora PostgreSQL-Compatible Edition .	June 17, 2021
Cloning for Aurora Serverless v1 clusters (p. 1824)	You can now create cloned clusters that are Aurora Serverless v1. For information about cloning, see Cloning a volume for an Aurora DB cluster .	June 16, 2021
Aurora MySQL version 1.22.5 (p. 1204)	Aurora MySQL version 1.22.5 is available.	June 3, 2021

Aurora MySQL version 2.10.0 (p. 1115)	Aurora MySQL version 2.10.0 is available. Some of the highlights include higher availability of reader instances during writer restarts , improvements to zero-downtime patching (ZDP) , improvements to zero-downtime restart (ZDR) , and the binlog I/O cache optimization .	May 25, 2021
Aurora global databases available in China (Beijing) and China (Ningxia) Regions (p. 1824)	You can now create Aurora global databases in the China (Beijing) and China (Ningxia) Regions. For information about Aurora global databases, see Working with Amazon Aurora global databases .	May 19, 2021
FIPS 140-2 support for Data API (p. 1824)	The Data API supports the Federal Information Processing Standard Publication 140-2 (FIPS 140-2) for SSL/TLS connections. For more information, see Data API availability .	May 14, 2021
Aurora PostgreSQL patch releases 3.2.7, 2.5.7, 1.7.7 compatible with PostgreSQL 11.7, 10.12, 9.6.17 (p. 1824)	New patch releases of Amazon Aurora PostgreSQL-Compatible Edition include release 3.2.7 compatible with PostgreSQL 11.7, release 2.5.7 compatible with PostgreSQL 10.12, and release 1.7.7 compatible with PostgreSQL 9.6.17. For more information, see Amazon Aurora PostgreSQL releases and engine versions .	May 11, 2021
Aurora PostgreSQL patch releases 3.1.4, 2.4.4, 1.6.4 compatible with PostgreSQL 11.6, 10.11, 9.6.16 (p. 1824)	New patch releases of Amazon Aurora PostgreSQL-Compatible Edition include release 3.1.4 compatible with PostgreSQL 11.6, release 2.4.4 compatible with PostgreSQL 10.11, and release 1.6.4 compatible with PostgreSQL 9.6.16. For more information, see Amazon Aurora PostgreSQL releases and engine versions .	May 11, 2021

AWS JDBC Driver for PostgreSQL (preview) (p. 1824)	The AWS JDBC Driver for PostgreSQL, now available in preview, is a client driver designed for the high availability of Aurora PostgreSQL. For more information, see Connecting with the Amazon Web Services JDBC Driver for PostgreSQL (preview) .	April 27, 2021
Aurora PostgreSQL patch releases 4.0.2, 3.4.2, 2.7.2, 1.9.2 compatible with PostgreSQL 12.4, 11.9, 10.14, 9.6.19 (p. 1824)	New patch releases of Amazon Aurora PostgreSQL-Compatible Edition include release 4.0.2 compatible with PostgreSQL 12.4, release 3.4.2 compatible with PostgreSQL 11.9, release 2.7.2 compatible with PostgreSQL 10.14, and release 1.9.2 compatible with PostgreSQL 9.6.19. For more information, see Amazon Aurora PostgreSQL releases and engine versions .	April 23, 2021
The Data API available in additional AWS Regions (p. 1824)	The Data API is now available in the Asia Pacific (Seoul) and Canada (Central) Regions. For more information, see Availability of the Data API .	April 9, 2021
Aurora MySQL version 1.23.2 (p. 1198)	Aurora MySQL version 1.23.2 is available.	March 18, 2021
Aurora PostgreSQL patch releases 4.0.1, 3.4.1, 2.7.1, 1.9.1 compatible with PostgreSQL 12.4, 11.9, 10.14, 9.6.19 (p. 1824)	New patch releases of Amazon Aurora PostgreSQL-Compatible Edition include release 4.0.1 compatible with PostgreSQL 12.4, release 3.4.1 compatible with PostgreSQL 11.9, release 2.7.1 compatible with PostgreSQL 10.14, and release 1.9.1 compatible with PostgreSQL 9.6.19. For more information, see Engine versions for Amazon Aurora PostgreSQL .	March 12, 2021
Amazon Aurora supports the Graviton2 DB instance classes (p. 1824)	You can now use the Graviton2 DB instance classes db.r6g.x to create DB clusters running MySQL or PostgreSQL. For more information, see DB instance class types .	March 12, 2021

RDS Proxy endpoint enhancements (p. 1824)	You can create additional endpoints associated with each RDS proxy. Creating an endpoint in a different VPC enables cross-VPC access for the proxy. Proxies for Aurora MySQL clusters can also have read-only endpoints. These reader endpoints connect to reader DB instances in the clusters and can improve read scalability and availability for query-intensive applications. For more information about RDS Proxy, see "Managing Connections with Amazon RDS Proxy" in the Amazon RDS User Guide or the Aurora user guide .	March 8, 2021
Aurora MySQL version 2.07.4 (p. 1145)	Aurora MySQL version 2.07.4 is available.	March 4, 2021
Aurora MySQL version 1.22.4 (p. 1204)	Aurora MySQL version 1.22.4 is available.	March 4, 2021
Amazon Aurora available in the Asia Pacific (Osaka) Region (p. 1824)	Aurora is now available in the Asia Pacific (Osaka) Region. For more information, see Regions and Availability Zones .	March 1, 2021
Aurora MySQL version 2.09.2 (p. 1122)	Aurora MySQL version 2.09.2 is available.	February 26, 2021
Aurora PostgreSQL supports enabling both IAM and Kerberos authentication on the same DB cluster (p. 1824)	Aurora PostgreSQL now supports enabling both IAM authentication and Kerberos authentication on the same DB cluster. For more information, see Database authentication with Amazon Aurora .	February 24, 2021
Aurora PostgreSQL patch releases 3.3.2, 2.6.2, 1.8.2 compatible with PostgreSQL 11.8, 10.13, 9.6.18 (p. 1824)	New patch releases of Amazon Aurora PostgreSQL-Compatible Edition include release 3.3.2 compatible with PostgreSQL 11.8, release 2.6.2 compatible with PostgreSQL 10.13, and release 1.8.2 compatible with PostgreSQL 9.6.18. For more information, see Engine versions for Amazon Aurora PostgreSQL .	February 12, 2021

Aurora global database now supports managed planned failover (p. 1824)	Aurora global database now supports managed planned failover, allowing you to more easily change the primary AWS Region of your Aurora global database. You can use managed planned failover with healthy Aurora global databases only. To learn more, see Disaster recovery and Amazon Aurora global databases . For reference information, see FailoverGlobalCluster in the <i>Amazon RDS API Reference</i> .	February 11, 2021
Data API for Aurora Serverless v1 now supports more data types (p. 1824)	With the Data API for Aurora Serverless v1, you can now use <code>UUID</code> and <code>JSON</code> data types as input to your database. Also with the Data API for Aurora Serverless v1, you can now have a <code>LONG</code> type value returned from your database as a <code>STRING</code> value. To learn more, see Calling the Data API . For reference information about supported data types, see SqlParameter in the <i>Amazon RDS Data Service API Reference</i> .	February 2, 2021
Aurora PostgreSQL supports major version upgrades to PostgreSQL 12 (p. 1824)	With Aurora PostgreSQL, you can now upgrade the DB engine to major version 12. For more information, see Upgrading the PostgreSQL DB engine for Aurora PostgreSQL .	January 28, 2021
Aurora PostgreSQL release 4.0 compatible with PostgreSQL 12.4 (p. 1824)	Amazon Aurora PostgreSQL release 4.0 is available and compatible with PostgreSQL 12.4. For more information, see Database engine versions for Amazon Aurora PostgreSQL .	January 28, 2021

Aurora MySQL supports in-place upgrade (p. 1824)	You can upgrade your Aurora MySQL 1.x cluster to Aurora MySQL 2.x, preserving the DB instances, endpoints, and so on of the original cluster. This in-place upgrade technique avoids the inconvenience of setting up a whole new cluster by restoring a snapshot. It also avoids the overhead of copying all your table data into a new cluster. For more information, see Upgrading the major version of an Aurora MySQL DB cluster from 1.x to 2.x .	January 11, 2021
AWS JDBC Driver for MySQL (preview) (p. 1824)	The AWS JDBC Driver for MySQL, now available in preview, is a client driver designed for the high availability of Aurora MySQL. For more information, see Connecting with the Amazon Web Services JDBC Driver for MySQL (preview) .	January 7, 2021
Aurora supports database activity streams on secondary clusters of a global database (p. 1824)	You can start a database a database activity stream on a primary or secondary cluster of Aurora PostgreSQL or Aurora MySQL. For supported engine versions, see Limitations of Aurora global databases .	December 22, 2020
Multi-master clusters with 4 DB instances (p. 1824)	The maximum number of DB instances in an Aurora MySQL multi-master cluster is now four. Formerly, the maximum was two DB instances. For more information, see Working with Aurora Multi-Master Clusters .	December 17, 2020
Aurora PostgreSQL supports AWS Lambda functions (p. 1824)	You can now invoke AWS Lambda function for your Aurora PostgreSQL DB clusters. For more information, see Invoking a Lambda function from an Aurora PostgreSQL DB cluster .	December 11, 2020

Aurora PostgreSQL releases 3.4.0, 2.7.0, and 1.9.0 (p. 1824)	New releases of Amazon Aurora PostgreSQL-Compatible Edition include release 3.4.0 (compatible with PostgreSQL 11.9), release 2.7.0 (compatible with PostgreSQL 10.14), and release 1.9.0 (compatible with PostgreSQL 9.6.19). For more information, see Database engine versions for Amazon Aurora PostgreSQL .	December 11, 2020
Aurora MySQL version 2.09.1 (p. 1124)	Aurora MySQL version 2.09.1 is available.	December 11, 2020
Amazon Aurora supports the Graviton2 DB instance classes in preview (p. 1824)	You can now use the Graviton2 DB instance classes db.r6g.x in preview to create DB clusters running MySQL or PostgreSQL. For more information, see DB instance class types .	December 11, 2020
Amazon Aurora Serverless v2 (preview) is now available in preview. (p. 1824)	Amazon Aurora Serverless v2 (preview) is available in preview. To work with Amazon Aurora Serverless v2 (preview), you must apply for access. For more information, see the Aurora Serverless v2 (preview) page.	December 1, 2020
Aurora PostgreSQL is now available for Aurora Serverless v1 in more AWS Regions. (p. 1824)	Aurora PostgreSQL is now available for Aurora Serverless v1 in more AWS Regions. You can now choose to run Aurora PostgreSQL Serverless in the same AWS Regions that offer Aurora MySQL Serverless. Additional AWS Regions with Aurora Serverless v1 support include US West (N. California), Asia Pacific (Singapore) Asia Pacific (Sydney) Asia Pacific (Seoul) Asia Pacific (Mumbai) Canada (Central) Europe (London) and Europe (Paris). For a list of all Regions and supported Aurora DB engines for Aurora Serverless v1, see Aurora Serverless . Amazon RDS Data API for Aurora Serverless v1 is also now available in these same AWS Regions. For a list of all Regions with support for the Data API for Aurora Serverless v1, see Data API for Aurora Serverless	November 24, 2020

Aurora MySQL version 1.23.1 (p. 1199)	Aurora MySQL version 1.23.1 is available.	November 24, 2020
Amazon RDS Performance Insights introduces new dimensions (p. 1824)	You can group database load according to the dimension groups for database, application (PostgreSQL), and session type (PostgreSQL). Amazon RDS also supports the dimensions db.name, db.application.name (PostgreSQL), and db.session_type.name (PostgreSQL). For more information, see Top load table .	November 24, 2020
Aurora PostgreSQL releases 3.2.6, 2.5.6, and 1.7.6 (p. 1824)	New releases of Amazon Aurora PostgreSQL-Compatible Edition include release 3.2.6 (compatible with PostgreSQL 11.7), release 2.5.6 (compatible with PostgreSQL 10.12), and release 1.7.6 (compatible with PostgreSQL 9.6.17). For more information, see Database engine versions for Amazon Aurora PostgreSQL .	November 13, 2020
Aurora MySQL version 2.08.3 (p. 1133)	Aurora MySQL version 2.08.3 is available.	November 12, 2020
Aurora MySQL version 2.07.3 (p. 1147)	Aurora MySQL version 2.07.3 is available.	November 10, 2020
Aurora MySQL version 1.22.3 (p. 1205)	Aurora MySQL version 1.22.3 is available.	November 9, 2020
Aurora Serverless v1 supports Aurora PostgreSQL version 10.12 (p. 1824)	Aurora PostgreSQL for Aurora Serverless v1 has been upgraded to Aurora PostgreSQL version 10.12 throughout the AWS Regions where Aurora PostgreSQL for Aurora Serverless v1 is supported. For more information, see Aurora Serverless v1 .	November 4, 2020
The Data API now supports tag-based authorization (p. 1824)	The Data API supports tag-based authorization. If you've labeled your RDS cluster resources with tags, you can use these tags in your policy statements to control access through the Data API. For more information, see Authorizing access to the Data API .	October 27, 2020

Amazon Aurora extends support for exporting snapshots to Amazon S3 (p. 1824)	You can now export DB snapshot data to Amazon S3 in all commercial AWS Regions. For more information, see Exporting DB snapshot data to Amazon S3 .	October 22, 2020
Aurora global database supports cloning (p. 1824)	You can now create clones of the primary and secondary DB clusters of your Aurora global databases. You can do so by using the AWS Management Console and choosing the Create clone menu option. You can also use the AWS CLI and run the <code>restore-db-cluster-to-point-in-time</code> command with the <code>--restore-type copy-on-write</code> option. Using the AWS Management Console or the AWS CLI, you can also clone DB clusters from your Aurora global databases across AWS accounts. For more information about cloning, see Cloning an Aurora DB cluster volume .	October 19, 2020
Amazon Aurora supports dynamic resizing for the cluster volume (p. 1824)	Starting with Aurora MySQL 1.23 and 2.09, and Aurora PostgreSQL 3.3.0 and Aurora PostgreSQL 2.6.0, Aurora reduces the size of the cluster volume after you remove data through operations such as <code>DROP TABLE</code> . To take advantage of this enhancement, upgrade to one of the appropriate versions depending on the database engine that your cluster uses. For information about this feature and how to check used and available storage space for an Aurora cluster, see Managing Performance and Scaling for Aurora DB Clusters .	October 13, 2020
Aurora PostgreSQL supports the pglogical extension (p. 1824)	Aurora PostgreSQL now supports the PostgreSQL pglogical extension version 2.2.2. For more information, see the Aurora PostgreSQL releases 3.3.0 and 2.6.0 at Database engine versions for Amazon Aurora PostgreSQL .	September 22, 2020

Amazon Aurora supports volume sizes up to 128 TiB (p. 1824)	New and existing Aurora cluster volumes can now grow to a maximum size of 128 tebibytes (TiB). For more information, see How Aurora storage grows .	September 22, 2020
Aurora PostgreSQL bug fix for very specific queries that use NOT EXISTS (p. 1824)	Fixed a bug for very specific queries that use the NOT EXISTS operator on Aurora PostgreSQL releases that were released on or after May 24, 2020. The fix is available in Aurora PostgreSQL release 2.5.4 (compatible with PostgreSQL 10.12), Aurora PostgreSQL release 2.6.1 (compatible with PostgreSQL 10.13), Aurora PostgreSQL release 3.2.4 (compatible with PostgreSQL 11.7), and Aurora PostgreSQL release 3.3.1 (compatible with PostgreSQL 11.8).	September 17, 2020
Aurora MySQL version 2.09.0 (p. 1127)	Aurora MySQL version 2.09.0 is available.	September 17, 2020
Aurora PostgreSQL supports the db.r5 and db.t3 DB instance classes in the China (Ningxia) Region (p. 1824)	You can now create Aurora PostgreSQL DB clusters in the China (Ningxia) Region that use the db.r5 and db.t3 DB instance classes. For more information, see DB instance classes .	September 3, 2020
Aurora PostgreSQL releases 3.3.0, 2.6.0, and 1.8.0 (p. 1824)	New releases of Amazon Aurora PostgreSQL-Compatible Edition include Aurora PostgreSQL release 3.3.0 (compatible with PostgreSQL 11.8), Aurora PostgreSQL release 2.6.0 (compatible with PostgreSQL 10.13), and Aurora PostgreSQL release 1.8.0 (compatible with PostgreSQL 9.6.18). For more information, see Database engine versions for Amazon Aurora PostgreSQL .	September 3, 2020

Aurora parallel query enhancements (p. 1824)	Starting with Aurora MySQL 2.09 and 1.23, you can take advantage of enhancements to the parallel query feature. Creating a parallel query cluster no longer requires a special engine mode. You can now turn parallel query on and off using the <code>aurora_parallel_query</code> configuration option for any provisioned cluster that's running a compatible Aurora MySQL version. You can upgrade an existing cluster to a compatible Aurora MySQL version and use parallel query, instead of creating a new cluster and importing data into it. You can use Performance Insights for parallel query clusters. You can stop and start parallel query clusters. You can create Aurora parallel query clusters that are compatible with MySQL 5.7. Parallel query works for tables that use the <code>DYNAMIC</code> row format. Parallel query clusters can use AWS Identity and Access Management (IAM) authentication. Reader DB instances in parallel query clusters can take advantage of the <code>READ COMMITTED</code> isolation level. You can also now create parallel query clusters in additional AWS Regions. For more information about the parallel query feature and these enhancements, see Working with parallel query for Aurora MySQL .	September 2, 2020
Aurora MySQL version 1.23.0 (p. 1200)	Aurora MySQL version 1.23.0 is available.	September 2, 2020
Aurora MySQL version 2.08.2 (p. 1135)	Aurora MySQL version 2.08.2 is available.	August 28, 2020

Aurora PostgreSQL releases 3.2.3, 2.5.3, and 1.7.3 (p. 1824)	New releases of Amazon Aurora PostgreSQL-Compatible Edition include Aurora PostgreSQL release 3.2.3 (compatible with PostgreSQL 11.7), Aurora PostgreSQL release 2.5.3 (compatible with PostgreSQL 10.12), and Aurora PostgreSQL release 1.7.3 (compatible with PostgreSQL 9.6.17). For more information, see Database engine versions for Amazon Aurora PostgreSQL .	August 27, 2020
Changes to Aurora MySQL parameter binlog_rows_query_log_events (p. 1041)	You can now change the value of the Aurora MySQL configuration parameter <code>binlog_rows_query_log_events</code> . Formerly, this parameter wasn't modifiable.	August 26, 2020
Aurora MySQL version 2.04.9 (p. 1159)	Aurora MySQL version 2.04.9 is available.	August 14, 2020
Aurora MySQL Serverless version 1.21.0 (p. 1249)	Aurora Serverless with MySQL 5.6 compatibility is available. The release includes features and bug fixes based on Aurora MySQL version 1.21.0. For more information about Aurora Serverless, see Using Amazon Aurora Serverless .	August 14, 2020
Support for automatic minor version upgrades for Aurora MySQL (p. 1824)	With Aurora MySQL, the setting Enable auto minor version upgrade now takes effect when you specify it for an Aurora MySQL DB cluster. When you enable auto minor version upgrade, Aurora automatically upgrades to new minor versions as they are released. The automatic upgrades occur during the maintenance window for the database. For Aurora MySQL, this feature applies only to Aurora MySQL version 2, which is compatible with MySQL 5.7. Initially, the automatic upgrade procedure brings Aurora MySQL DB clusters to version 2.07.2. For more information about how this feature works with Aurora MySQL, see Database Upgrades and Patches for Amazon Aurora MySQL .	August 3, 2020

Aurora PostgreSQL supports major version upgrades to PostgreSQL version 11 (p. 1824)	With Aurora PostgreSQL, you can now upgrade the DB engine to major version 11. For more information, see Upgrading the PostgreSQL DB engine for Aurora PostgreSQL .	July 28, 2020
Aurora PostgreSQL releases 3.1.3, 2.4.3, and 1.6.3 (p. 1824)	New patch releases of Aurora PostgreSQL include Aurora PostgreSQL release 3.1.3 (compatible with PostgreSQL 11.6), Aurora PostgreSQL release 2.4.3 (compatible with PostgreSQL 10.11), and Aurora PostgreSQL release 1.6.3 (compatible with PostgreSQL 9.6.16). For more information, see Database engine versions for Amazon Aurora PostgreSQL .	July 27, 2020
Aurora PostgreSQL releases 3.2.2, 2.5.2, and 1.7.2 (p. 1824)	New releases of Amazon Aurora PostgreSQL-Compatible Edition include Aurora PostgreSQL release 3.2.2 (compatible with PostgreSQL 11.7), Aurora PostgreSQL release 2.5.2 (compatible with PostgreSQL 10.12), and Aurora PostgreSQL release 1.7.2 (compatible with PostgreSQL 9.6.17). For more information, see Database engine versions for Amazon Aurora PostgreSQL .	July 9, 2020
Amazon Aurora supports AWS PrivateLink (p. 1824)	Amazon Aurora now supports creating Amazon VPC endpoints for Amazon RDS API calls to keep traffic between applications and Aurora in the AWS network. For more information, see Amazon Aurora and interface VPC endpoints (AWS PrivateLink) .	July 9, 2020
RDS Proxy generally available (p. 1824)	RDS Proxy is now generally available. You can use RDS Proxy with RDS for MySQL, Aurora MySQL, RDS for PostgreSQL, and Aurora PostgreSQL for production workloads. For more information about RDS Proxy, see "Managing Connections with Amazon RDS Proxy" in the Amazon RDS User Guide or the Aurora user guide .	June 30, 2020

Aurora Serverless version 2.08.3 (p. 1248)	Aurora Serverless with MySQL 5.7 compatibility is available. The release includes features and bug fixes based on Aurora MySQL version 2.08.3. For more information about Aurora Serverless, see Using Amazon Aurora Serverless .	June 24, 2020
Aurora Serverless version 2.07.1 (p. 1248)	Aurora Serverless with MySQL 5.7 compatibility is available. The release includes features and bug fixes based on Aurora MySQL version 2.07.1. For more information about Aurora Serverless, see Using Amazon Aurora Serverless .	June 24, 2020
Aurora global database write forwarding (p. 1824)	You can now enable write capability on secondary clusters in a global database. With write forwarding, you issue DML statements on a secondary cluster, Aurora forwards the write to the primary cluster, and the updated data is replicated to all the secondary clusters. For more information, see Write forwarding for secondary AWS Regions with an Aurora global database .	June 18, 2020
Aurora MySQL version 2.08.1 (p. 1136)	Aurora MySQL version 2.08.1 is available.	June 18, 2020
Aurora MySQL version 1.22.2 for parallel query clusters (p. 1206)	Aurora MySQL version 1.22.2 is available when you create a parallel query cluster.	June 18, 2020
Aurora MySQL version 1.20.1 for parallel query clusters (p. 1212)	Aurora MySQL version 1.20.1 is available when you create a parallel query cluster.	June 11, 2020
Aurora supports integration with AWS Backup (p. 1824)	You can use AWS Backup to manage backups of Aurora DB clusters. For more information, see Overview of backing up and restoring an Aurora DB cluster .	June 10, 2020
Aurora PostgreSQL supports db.t3.large DB instance classes (p. 1824)	You can now create Aurora PostgreSQL DB clusters that use the db.t3.large DB instance classes. For more information, see DB instance classes .	June 5, 2020

Aurora global database supports PostgreSQL version 11.7 and managed recovery point objective (RPO) (p. 1824)	You can now create Aurora global databases for the PostgreSQL database engine version 11.7. You can also manage how a PostgreSQL global database recovers from a failure using a recovery point objective (RPO). For more information, see Cross-Region Disaster Recovery for Aurora global databases .	June 4, 2020
Aurora PostgreSQL releases 3.2.1, 2.5.1, and 1.7.1 (p. 1824)	New releases of Amazon Aurora PostgreSQL-Compatible Edition include Aurora PostgreSQL release 3.2.1 (compatible with PostgreSQL 11.7), Aurora PostgreSQL release 2.5.1 (compatible with PostgreSQL 10.12), and Aurora PostgreSQL release 1.7.1 (compatible with PostgreSQL 9.6.17). For more information, see Database engine versions for Amazon Aurora PostgreSQL .	June 4, 2020
Aurora MySQL version 2.08.0 (p. 1137)	Aurora MySQL version 2.08.0 is available.	June 2, 2020
Aurora MySQL version 1.19.6 for parallel query clusters (p. 1214)	Aurora MySQL version 1.19.6 is available when you create a parallel query cluster.	June 2, 2020
Aurora MySQL supports database monitoring with database activity streams (p. 1824)	Aurora MySQL now includes database activity streams, which provide a near-real-time data stream of the database activity in your relational database. For more information, see Using database activity streams .	June 2, 2020
The query editor available in additional AWS Regions (p. 1824)	The query editor for Aurora Serverless is now available in additional AWS Regions. For more information, see Availability of the query editor .	May 28, 2020
The Data API available in additional AWS Regions (p. 1824)	The Data API is now available in additional AWS Regions. For more information, see Availability of the Data API .	May 28, 2020
RDS Proxy available in Canada (Central) Region (p. 1824)	You can now use the RDS Proxy preview in the Canada (Central) Region. For more information about RDS Proxy, see Managing connections with Amazon RDS proxy (preview) .	May 28, 2020

Aurora global database and cross-Region read replicas (p. 1824)	For an Aurora global database, you can now create an Aurora MySQL cross-Region read replica from the primary cluster in the same region as a secondary cluster. For more information about Aurora Global Database and cross-Region read replicas, see Working with Amazon Aurora global database and Replicating Amazon Aurora MySQL DB .	May 18, 2020
RDS Proxy available in more AWS Regions (p. 1824)	You can now use the RDS Proxy preview in the US West (N. California) Region, the Europe (London) Region, the Europe (Frankfurt) Region, the Asia Pacific (Seoul) Region, the Asia Pacific (Mumbai) Region, the Asia Pacific (Singapore) Region, and the Asia Pacific (Sydney) Region. For more information about RDS Proxy, see Managing connections with Amazon RDS proxy (preview) .	May 13, 2020
Aurora PostgreSQL-Compatible Edition supports on-premises or self-hosted Microsoft active directory (p. 1824)	You can now use an on-premises or self-hosted Active Directory for Kerberos authentication of users when they connect to your Aurora PostgreSQL DB clusters. For more information, see Using Kerberos authentication with Aurora PostgreSQL .	May 7, 2020
Aurora MySQL multi-master clusters available in more AWS Regions (p. 1824)	You can now create Aurora multi-master clusters in the Asia Pacific (Seoul) Region, the Asia Pacific (Tokyo) Region, the Asia Pacific (Mumbai) Region, and the Europe (Frankfurt) Region. For more information about multi-master clusters, see Working with Aurora multi-master clusters .	May 7, 2020
Performance Insights supports analyzing statistics of running Aurora MySQL queries (p. 1824)	You can now analyze statistics of running queries with Performance Insights for Aurora MySQL DB instances. For more information, see Analyzing statistics of running queries .	May 5, 2020

Java client library for Data API generally available (p. 1824)	The Java client library for the Data API is now generally available. You can download and use a Java client library for Data API. It enables you to map your client-side classes to requests and responses of the Data API. For more information, see Using the Java client library for Data API .	April 30, 2020
Amazon Aurora available in the Europe (Milan) Region (p. 1824)	Aurora is now available in the Europe (Milan) Region. For more information, see Regions and Availability Zones .	April 28, 2020
Amazon Aurora available in the Europe (Milan) Region (p. 1824)	Aurora is now available in the Europe (Milan) Region. For more information, see Regions and Availability Zones .	April 27, 2020
Amazon Aurora available in the Africa (Cape Town) Region (p. 1824)	Aurora is now available in the Africa (Cape Town) Region. For more information, see Regions and Availability Zones .	April 22, 2020
Aurora PostgreSQL releases 3.1.2, 2.4.2, and 1.6.2 (p. 1824)	New patch releases of Aurora PostgreSQL include Aurora PostgreSQL release 3.1.2 (compatible with PostgreSQL 11.6), Aurora PostgreSQL release 2.4.2 (compatible with PostgreSQL 10.11), and Aurora PostgreSQL release 1.6.2 (compatible with PostgreSQL 9.6.16). For more information, see Database engine versions for Amazon Aurora PostgreSQL .	April 17, 2020
Aurora MySQL version 2.07.2 (p. 1149)	Aurora MySQL version 2.07.2 is available.	April 17, 2020
Aurora PostgreSQL releases 3.1.1, 2.4.1, and 1.6.1 (p. 1824)	New patch releases of Aurora PostgreSQL include Aurora PostgreSQL release 3.1.1 (compatible with PostgreSQL 11.6), Aurora PostgreSQL release 2.4.1 (compatible with PostgreSQL 10.11), and Aurora PostgreSQL release 1.6.1 (compatible with PostgreSQL 9.6.16). For more information, see Database engine versions for Amazon Aurora PostgreSQL .	April 16, 2020

Aurora PostgreSQL now supports db.r5.16xlarge and db.r5.8xlarge DB instance classes (p. 1824)	You can now create Aurora PostgreSQL DB clusters running PostgreSQL that use the db.r5.16xlarge and db.r5.8xlarge DB instance classes. For more information, see Hardware specifications for DB instance classes for Aurora .	April 8, 2020
Amazon RDS proxy for PostgreSQL (p. 1824)	Amazon RDS Proxy is now available for PostgreSQL. You can use RDS Proxy to reduce the overhead of connection management on your cluster and also the chance of "too many connections" errors. The RDS Proxy is currently in public preview for PostgreSQL. For more information, see Managing connections with Amazon RDS proxy (preview) .	April 8, 2020
Aurora global databases now support Aurora PostgreSQL (p. 1824)	You can now create Aurora global databases for the PostgreSQL database engine. An Aurora global database spans multiple AWS Regions, enabling low latency global reads and disaster recovery from region-wide outages. For more information, see Working with Amazon Aurora global database .	March 10, 2020
Aurora MySQL version 1.22.2 (p. 1206)	Aurora MySQL version 1.22.2 is available.	March 5, 2020
Aurora MySQL version 1.20.1 (p. 1212)	Aurora MySQL version 1.20.1 is available.	March 5, 2020
Aurora MySQL version 1.19.6 (p. 1214)	Aurora MySQL version 1.19.6 is available.	March 5, 2020
Aurora MySQL version 1.17.9 (p. 1219)	Aurora MySQL version 1.17.9 is available.	March 5, 2020
Support for major version upgrades for Aurora PostgreSQL (p. 1824)	With Aurora PostgreSQL, you can now upgrade the DB engine to a major version. By doing so, you can skip ahead to a newer major version when you upgrade select PostgreSQL engine versions. For more information, see Upgrading the PostgreSQL DB engine for Aurora PostgreSQL .	March 4, 2020

Aurora PostgreSQL supports Kerberos authentication (p. 1824)	You can now use Kerberos authentication to authenticate users when they connect to your Aurora PostgreSQL DB clusters. For more information, see Using Kerberos authentication with Aurora PostgreSQL .	February 28, 2020
Aurora PostgreSQL releases 3.1, 2.4, and 1.6 (p. 1824)	New releases of Amazon Aurora PostgreSQL-Compatible Edition include Aurora PostgreSQL release 3.1 (compatible with PostgreSQL 11.6), Aurora PostgreSQL release 2.4 (compatible with PostgreSQL 10.11), and Aurora PostgreSQL release 1.6 (compatible with PostgreSQL 9.6.16). For more information, see Database engine versions for Amazon Aurora PostgreSQL .	February 11, 2020
The Data API supports AWS PrivateLink (p. 1824)	The Data API now supports creating Amazon VPC endpoints for Data API calls to keep traffic between applications and the Data API in the AWS network. For more information, see Creating an Amazon VPC endpoint (AWS PrivateLink) for the Data API .	February 6, 2020
Aurora machine learning support in Aurora PostgreSQL (p. 1824)	The <code>aws_ml</code> Aurora PostgreSQL extension provides functions you use in your database queries to call Amazon Comprehend for sentiment analysis and SageMaker to run your own machine learning models. For more information, see Using machine learning (ML) capabilities with Aurora .	February 5, 2020
Aurora PostgreSQL supports exporting data to Amazon S3 (p. 1824)	You can query data from an Aurora PostgreSQL DB cluster and export it directly into files stored in an Amazon S3 bucket. For more information, see Exporting data from an Aurora PostgreSQL DB cluster to Amazon S3 .	February 5, 2020
Support for exporting DB snapshot data to Amazon S3 (p. 1824)	Amazon Aurora supports exporting DB snapshot data to Amazon S3 for MySQL and PostgreSQL. For more information, see Exporting DB snapshot data to Amazon S3 .	January 9, 2020

Aurora MySQL version 2.07.1 (p. 1151)	Aurora MySQL version 2.07.1 is available.	December 23, 2019
Aurora MySQL version 1.22.1 (p. 1207)	Aurora MySQL version 1.22.1 is available.	December 23, 2019
Aurora MySQL release notes in document history (p. 1824)	This section now includes history entries for Aurora MySQL-Compatible Edition release notes for versions released after August 31, 2018. For the full release notes for a specific version, choose the link in the first column of the history entry.	December 13, 2019
Amazon RDS proxy (p. 1824)	You can reduce the overhead of connection management on your cluster, and reduce the chance of "too many connections" errors, by using the Amazon RDS Proxy. You associate each proxy with an RDS DB instance or Aurora DB cluster. Then you use the proxy endpoint in the connection string for your application. The Amazon RDS Proxy is currently in a public preview state. It supports the Aurora MySQL database engine. For more information, see Managing connections with Amazon RDS proxy (preview) .	December 3, 2019
Data API for Aurora Serverless supports data type mapping hints (p. 1824)	You can now use a hint to instruct the Data API for Aurora Serverless to send a String value to the database as a different type. For more information, see Calling the data API .	November 26, 2019
Data API for Aurora Serverless supports a Java client library (preview) (p. 1824)	You can download and use a Java client library for Data API. It enables you to map your client-side classes to requests and responses of the Data API. For more information, see Using the Java client library for Data API .	November 26, 2019

Aurora PostgreSQL release 3.0 (p. 1824)	Amazon Aurora PostgreSQL release 3.0 is available and compatible with PostgreSQL 11.4. Supported AWS Regions include us-east-1, us-east-2, us-west-2, eu-west-1, ap-northeast-1, and ap-northeast-2. For more information, see Database engine versions for Amazon Aurora PostgreSQL .	November 26, 2019
Aurora PostgreSQL is FedRAMP HIGH eligible (p. 1824)	Aurora PostgreSQL is FedRAMP HIGH eligible. For details about AWS and compliance efforts, see AWS services in scope by compliance program .	November 26, 2019
READ COMMITTED isolation level enabled for Amazon Aurora MySQL replicas (p. 1824)	You can now enable the READ COMMITTED isolation level on Aurora MySQL Replicas. Doing so requires enabling the <code>aurora_read_replica_read_committed_isolation_enabled</code> configuration setting at the session level. Using the READ COMMITTED isolation level for long-running queries on OLTP clusters can help address issues with history list length. Before enabling this setting, be sure to understand how the isolation behavior on Aurora Replicas differs from the usual MySQL implementation of READ COMMITTED. For more information, see Aurora MySQL isolation levels .	November 25, 2019
Performance Insights supports analyzing statistics of running Aurora PostgreSQL queries (p. 1824)	You can now analyze statistics of running queries with Performance Insights for Aurora PostgreSQL DB instances. For more information, see Analyzing statistics of running queries .	November 25, 2019
More clusters in an Aurora global database (p. 1824)	You can now add multiple secondary regions to an Aurora global database. You can take advantage of low latency global reads and disaster recovery across a wider geographic area. For more information about Aurora global databases, see Working with Amazon Aurora global databases .	November 25, 2019

Aurora machine learning support in Aurora MySQL (p. 1824)	In Aurora MySQL 2.07 and higher, you can call Amazon Comprehend for sentiment analysis and SageMaker for a wide variety of machine learning algorithms. You use the results directly in your database application by embedding calls to stored functions in your queries. For more information, see Using machine learning (ML) capabilities with Aurora .	November 25, 2019
Aurora global database no longer requires engine mode setting (p. 1824)	You no longer need to specify <code>--engine-mode=global</code> when creating a cluster that is intended to be part of an Aurora global database. All Aurora clusters that meet the compatibility requirements are eligible to be part of a global database. For example, the cluster currently must use Aurora MySQL version 1 with MySQL 5.6 compatibility. For information about Aurora global databases, see Working with Amazon Aurora global databases .	November 25, 2019
Aurora global database is available for Aurora MySQL version 2 (p. 1824)	Starting in Aurora MySQL 2.07, you can create an Aurora global database with MySQL 5.7 compatibility. You don't need to specify the <code>global</code> engine mode for the primary or secondary clusters. You can add any new provisioned cluster with Aurora MySQL 2.07 or higher to an Aurora Global Database. For information about Aurora Global Database, see Working with Amazon Aurora global database .	November 25, 2019
Aurora MySQL version 2.07.0 (p. 1153)	Aurora MySQL version 2.07.0 is available.	November 25, 2019
Aurora MySQL version 1.22.0 (p. 1208)	Aurora MySQL version 1.22.0 is available.	November 25, 2019
Aurora MySQL version 1.21.0 (p. 1211)	Aurora MySQL version 1.21.0 is available.	November 25, 2019
Aurora MySQL version 2.06.0 (p. 1155)	Aurora MySQL version 2.06.0 is available.	November 22, 2019
Aurora MySQL version 2.04.8 (p. 1163)	Aurora MySQL version 2.04.8 is available.	November 20, 2019

Aurora MySQL hot row contention optimization available without lab mode (p. 1824)	The hot row contention optimization is now generally available for Aurora MySQL and does not require the Aurora lab mode setting to be ON. This feature substantially improves throughput for workloads with many transactions contending for rows on the same page. The improvement involves changing the lock release algorithm used by Aurora MySQL.	November 19, 2019
Aurora MySQL hash joins available without lab mode (p. 1824)	The hash join feature is now generally available for Aurora MySQL and does not require the Aurora lab mode setting to be ON. This feature can improve query performance when you need to join a large amount of data by using an equijoin. For more information about using this feature, see Working with hash joins in Aurora MySQL .	November 19, 2019
Aurora MySQL 2.* support for more db.r5 instance classes (p. 1824)	Aurora MySQL clusters now support the instance types db.r5.8xlarge, db.r5.16xlarge, and db.r5.24xlarge. For more information about instance types for Aurora MySQL clusters, see Choosing the DB instance class .	November 19, 2019
Aurora MySQL 2.* support for backtracking (p. 1824)	Aurora MySQL 2.* versions now offer a quick way to recover from user errors, such as dropping the wrong table or deleting the wrong row. Backtrack allows you to move your database to a prior point in time without needing to restore from a backup, and it completes within seconds, even for large databases. For details, see Backtracking an Aurora DB cluster .	November 19, 2019
Aurora MySQL version 2.04.7 (p. 1164)	Aurora MySQL version 2.04.7 is available.	November 14, 2019
Aurora MySQL version 2.05.0 (p. 1158)	Aurora MySQL version 2.05.0 is available.	November 11, 2019
Aurora MySQL version 1.20.0 (p. 1213)	Aurora MySQL version 1.20.0 is available.	November 11, 2019

Billing tag support for Aurora (p. 1824)	You can now use tags to keep track of cost allocation for resources such as Aurora clusters, DB instances within Aurora clusters, I/O, backups, snapshots, and so on. You can see costs associated with each tag using AWS Cost Explorer. For more information about using tags with Aurora, see Tagging Amazon RDS resources . For general information about tags and ways to use them for cost analysis, see Using cost allocation tags and User-defined cost allocation tags .	October 23, 2019
Data API for Aurora PostgreSQL (p. 1824)	Aurora PostgreSQL now supports using the Data API with Amazon Aurora Serverless DB clusters. For more information, see Using the Data API for Aurora Serverless .	September 23, 2019
Aurora MySQL version 2.04.6 (p. 1166)	Aurora MySQL version 2.04.6 is available.	September 19, 2019
Aurora MySQL version 1.19.5 (p. 1214)	Aurora MySQL version 1.19.5 is available.	September 19, 2019
Aurora PostgreSQL supports uploading database logs to CloudWatch logs (p. 1824)	You can configure your Aurora PostgreSQL DB cluster to publish log data to a log group in Amazon CloudWatch Logs. With CloudWatch Logs, you can perform real-time analysis of the log data, and use CloudWatch to create alarms and view metrics. You can use CloudWatch Logs to store your log records in highly durable storage. For more information, see Publishing Aurora PostgreSQL logs to Amazon CloudWatch Logs .	August 9, 2019
Multi-master clusters for Aurora MySQL (p. 1824)	You can set up Aurora MySQL multi-master clusters. In these clusters, each DB instance has read/write capability. For more information, see Working with Aurora multi-master clusters .	August 8, 2019

Aurora PostgreSQL supports Aurora Serverless (p. 1824)	You can now use Amazon Aurora Serverless with Aurora PostgreSQL. An Aurora Serverless DB cluster automatically starts up, shuts down, and scales up or down its compute capacity based on your application's needs. For more information, see Using Amazon Aurora Serverless .	July 9, 2019
Aurora MySQL version 2.04.5 (p. 1168)	Aurora MySQL version 2.04.5 is available.	July 8, 2019
Aurora PostgreSQL releases 2.3.3 and 1.5.2 (p. 1824)	Amazon Aurora PostgreSQL-Compatible Edition release 2.3.3 is available and compatible with PostgreSQL 10.7. Amazon Aurora PostgreSQL-Compatible Edition release 1.5.2 is available and compatible with PostgreSQL 9.6.12. For more information, see Database engine versions for Amazon Aurora PostgreSQL .	July 3, 2019
Cross-account cloning for Aurora MySQL (p. 1824)	You can now clone the cluster volume for an Aurora MySQL DB cluster between AWS accounts. You authorize the sharing through AWS Resource Access Manager (AWS RAM). The cloned cluster volume uses a copy-on-write mechanism, which only requires additional storage for new or changed data. For more information about cloning for Aurora, see Cloning databases in an Aurora DB cluster .	July 2, 2019
Aurora PostgreSQL releases 2.3.1 and 1.5.1 (p. 1824)	Amazon Aurora PostgreSQL-Compatible Edition release 2.3.1 is available and compatible with PostgreSQL 10.7. Amazon Aurora PostgreSQL-Compatible Edition release 1.5.1 is available and compatible with PostgreSQL 9.6.12. For more information, see Database engine versions for Amazon Aurora PostgreSQL .	July 2, 2019
Aurora PostgreSQL supports db.t3 DB instance classes (p. 1824)	You can now create Aurora PostgreSQL DB clusters that use the db.t3 DB instance classes. For more information, see DB instance class .	June 20, 2019

Support for importing data from Amazon S3 for Aurora PostgreSQL (p. 1824)	You can now import data from an Amazon S3 file into a table in an Aurora PostgreSQL DB cluster. For more information, see Importing Amazon S3 data into an Aurora PostgreSQL DB cluster .	June 19, 2019
Aurora PostgreSQL now provides fast failover recovery with cluster cache management (p. 1824)	Aurora PostgreSQL now provides cluster cache management to ensure fast recovery of the primary DB instance in the event of a failover. For more information, see Fast recovery after failover with cluster cache management .	June 11, 2019
Aurora MySQL version 1.19.2 (p. 1215)	Aurora MySQL version 1.19.2 is available.	June 5, 2019
Data API for Aurora Serverless generally available (p. 1824)	You can access Aurora Serverless clusters with web services-based applications using the Data API. For more information, see Using the Data API for Aurora Serverless .	May 30, 2019
Aurora PostgreSQL supports database monitoring with database activity streams (p. 1824)	Aurora PostgreSQL now includes database activity streams, which provide a near-real-time data stream of the database activity in your relational database. For more information, see Using database activity streams .	May 30, 2019
Aurora PostgreSQL release 2.3 (p. 1824)	Release 2.3 of Amazon Aurora PostgreSQL-Compatible Edition is available and compatible with PostgreSQL 10.7. For more information, see Version 2.3 .	May 30, 2019
Aurora MySQL version 2.04.4 (p. 1169)	Aurora MySQL version 2.04.4 is available.	May 29, 2019
Amazon Aurora recommendations (p. 1824)	Amazon Aurora now provides automated recommendations for Aurora resources. For more information, see Using Amazon Aurora recommendations .	May 22, 2019
Aurora PostgreSQL releases 1.2.2, 1.3.2, 2.0.1, 2.1.1, 2.2.1 (p. 1824)	The following patch releases for Amazon Aurora PostgreSQL-Compatible Edition are now available and include releases 1.2.2, 1.3.2, 2.0.1, 2.1.1, and 2.2.1. For more information, see Database engine versions for Amazon Aurora PostgreSQL .	May 21, 2019

Performance Insights support for Aurora global database (p. 1824)	You can now use Performance Insights with Aurora Global Database. For information about Performance Insights for Aurora, see Using Amazon RDS performance insights . For information about Aurora global databases, see Working with Aurora global database .	May 13, 2019
Aurora PostgreSQL release 1.4 (p. 1824)	Release 1.4 of Amazon Aurora PostgreSQL-Compatible Edition is available and compatible with PostgreSQL 9.6.11. For more information, see Version 1.4 .	May 9, 2019
Aurora MySQL version 2.04.3 (p. 1170)	Aurora MySQL version 2.04.3 is available.	May 9, 2019
Aurora MySQL version 1.19.1 (p. 1216)	Aurora MySQL version 1.19.1 is available.	May 9, 2019
Performance Insights is available for Aurora MySQL 5.7 (p. 1824)	Amazon RDS Performance Insights is now available for Aurora MySQL 2.x versions, which are compatible with MySQL 5.7. For more information, see Using Amazon RDS performance insights .	May 3, 2019
Aurora MySQL version 2.04.2 (p. 1172)	Aurora MySQL version 2.04.2 is available.	May 2, 2019
Aurora global databases available in more AWS Regions (p. 1824)	You can now create Aurora global databases in most AWS Regions where Aurora is available. For information about Aurora global databases, see Working with Amazon Aurora global databases .	April 30, 2019
Minimum capacity of 1 for Aurora Serverless (p. 1824)	The minimum capacity setting you can use for an Aurora Serverless cluster is 1. Formerly, the minimum was 2. For information about specifying Aurora Serverless capacity values, see Setting the capacity of an Aurora Serverless DB cluster .	April 29, 2019
Aurora Serverless timeout action (p. 1824)	You can now specify the action to take when an Aurora Serverless capacity change times out. For more information, see Timeout action for capacity changes .	April 29, 2019

Per-second billing (p. 1824)	Amazon RDS is now billed in 1-second increments in all AWS Regions except AWS GovCloud (US) for on-demand instances. For more information, see DB instance billing for Aurora .	April 25, 2019
Sharing Aurora Serverless snapshots across AWS Regions (p. 1824)	With Aurora Serverless, snapshots are always encrypted. If you encrypt the snapshot with your own AWS KMS key, you can now copy or share the snapshot across AWS Regions. For information about snapshots of Aurora Serverless DB clusters, see Aurora Serverless and snapshots .	April 17, 2019
Restore MySQL 5.7 backups from Amazon S3 (p. 1824)	You can now create a backup of your MySQL version 5.7 database, store it on Amazon S3, and then restore the backup file onto a new Aurora MySQL DB cluster. For more information, see Migrating data from an external MySQL database to an Aurora MySQL DB cluster .	April 17, 2019
Sharing Aurora Serverless snapshots across regions (p. 1824)	With Aurora Serverless, snapshots are always encrypted. If you encrypt the snapshot with your own AWS KMS key, you can now copy or share the snapshot across regions. For information about snapshots of Aurora Serverless DB clusters, see Aurora Serverless and snapshots .	April 16, 2019
Aurora proof-of-concept tutorial (p. 1824)	You can learn how to perform a proof of concept to try your application and workload with Aurora. For the full tutorial, see Performing an Aurora proof of concept .	April 16, 2019
Aurora Serverless supports restoring from an Amazon S3 backup (p. 1824)	You can now import backups from Amazon S3 to an Aurora Serverless cluster. For details about that procedure, see Migrating data from MySQL by using an Amazon S3 bucket .	April 16, 2019

New modifiable parameters for Aurora Serverless (p. 1824)	You can now modify the following DB parameters for an Aurora Serverless cluster: <code>innodb_file_format</code> , <code>innodb_file_per_table</code> , <code>innodb_large_prefix</code> , <code>innodb_lock_wait_timeout</code> , <code>innodb_monitor_disable</code> , <code>innodb_monitor_enable</code> , <code>innodb_monitor_reset</code> , <code>innodb_monitor_reset_all</code> , <code>innodb_print_all_deadlocks</code> , <code>log_warnings</code> , <code>net_read_timeout</code> , <code>net_retry_count</code> , <code>net_write_timeout</code> , <code>sql_mode</code> , and <code>tx_isolation</code> . For more information about configuration parameters for Aurora Serverless clusters, see Aurora Serverless and parameter groups .	April 4, 2019
Aurora PostgreSQL supports db.r5 DB instance classes (p. 1824)	You can now create Aurora PostgreSQL DB clusters that use the db.r5 DB instance classes. For more information, see DB instance class .	April 4, 2019
Aurora PostgreSQL logical replication (p. 1824)	You can now use PostgreSQL logical replication to replicate parts of a database for an Aurora PostgreSQL DB cluster. For more information, see Using PostgreSQL logical replication .	March 28, 2019
GTID support for Aurora MySQL 2.04 (p. 1824)	You can now use replication with the global transaction ID (GTID) feature of MySQL 5.7. This feature simplifies performing binary log (binlog) replication between Aurora MySQL and an external MySQL 5.7-compatible database. The replication can use the Aurora MySQL cluster as the source or the destination. This feature is available for Aurora MySQL 2.04 and higher. For more information about GTID-based replication and Aurora MySQL, see Using GTID-based replication for Aurora MySQL .	March 25, 2019
Aurora MySQL version 2.04.1 (p. 1173)	Aurora MySQL version 2.04.1 is available.	March 25, 2019

Aurora MySQL version 2.04 (p. 1175)	Aurora MySQL version 2.04 is available.	March 25, 2019
Uploading Aurora Serverless logs to Amazon CloudWatch (p. 1824)	You can now have Aurora upload database logs to CloudWatch for an Aurora Serverless cluster. For more information, see Viewing Aurora Serverless DB clusters . As part of this enhancement, you can now define values for instance-level parameters in a DB cluster parameter group, and those values apply to all DB instances in the cluster unless you override them in the DB parameter group. For more information, see Working with DB parameter groups and DB cluster parameter groups .	February 25, 2019
Aurora MySQL supports db.t3 DB instance classes (p. 1824)	You can now create Aurora MySQL DB clusters that use the db.t3 DB instance classes. For more information, see DB instance class .	February 25, 2019
Aurora MySQL supports db.r5 DB instance classes (p. 1824)	You can now create Aurora MySQL DB clusters that use the db.r5 DB instance classes. For more information, see DB instance class .	February 25, 2019
Performance Insights counters for Aurora MySQL (p. 1824)	You can now add performance counters to your Performance Insights charts for Aurora MySQL DB instances. For more information, see Performance Insights dashboard components .	February 19, 2019
Aurora PostgreSQL release 2.2.0 (p. 1824)	Release 2.2.0 of Aurora PostgreSQL is available and compatible with PostgreSQL 10.6. For more information, see Version 2.2.0 .	February 13, 2019
Aurora MySQL version 2.03.4 (p. 1176)	Aurora MySQL version 2.03.4 is available.	February 7, 2019
Aurora MySQL version 1.19.0 (p. 1217)	Aurora MySQL version 1.19.0 is available.	February 7, 2019

Amazon RDS Performance Insights supports viewing more SQL text for Aurora MySQL (p. 1824)	Amazon RDS Performance Insights now supports viewing more SQL text in the Performance Insights dashboard for Aurora MySQL DB instances. For more information, see Viewing more SQL text in the Performance Insights dashboard .	February 6, 2019
Amazon RDS Performance Insights supports viewing more SQL text for Aurora PostgreSQL (p. 1824)	Amazon RDS Performance Insights now supports viewing more SQL text in the Performance Insights dashboard for Aurora PostgreSQL DB instances. For more information, see Viewing more SQL text in the Performance Insights dashboard .	January 24, 2019
Aurora MySQL version 2.03.3 (p. 1177)	Aurora MySQL version 2.03.3 is available.	January 18, 2019
Aurora MySQL version 1.17.8 (p. 1219)	Aurora MySQL version 1.17.8 is available.	January 17, 2019
Aurora MySQL version 2.03.2 (p. 1179)	Aurora MySQL version 2.03.2 is available.	January 9, 2019
Aurora backup billing (p. 1824)	You can use the Amazon CloudWatch metrics <code>TotalBackupStorageBilled</code> , <code>SnapshotStorageUsed</code> , and <code>BackupRetentionPeriodStorageUsed</code> to monitor the space usage of your Aurora backups. For more information about how to use CloudWatch metrics, see Overview of monitoring . For more information about how to manage storage for backup data, see Understanding Aurora backup storage usage .	January 3, 2019
Performance Insights counters (p. 1824)	You can now add performance counters to your Performance Insights charts. For more information, see Performance Insights dashboard components .	December 6, 2018
Aurora global database (p. 1824)	You can now create Aurora global databases. An Aurora global database spans multiple AWS Regions, enabling low latency global reads and disaster recovery from region-wide outages. For more information, see Working with Amazon Aurora global database .	November 28, 2018

Query plan management in Aurora PostgreSQL (p. 1824)	Aurora PostgreSQL now provides query plan management that you can use to manage PostgreSQL query execution plans. For more information, see Managing query execution plans for Aurora PostgreSQL .	November 20, 2018
Query editor for Aurora Serverless (beta) (p. 1824)	You can run SQL statements in the Amazon RDS console on Aurora Serverless clusters. For more information, see Using the query editor for Aurora Serverless .	November 20, 2018
Data API for Aurora Serverless (beta) (p. 1824)	You can access Aurora Serverless clusters with web services-based applications using the Data API. For more information, see Using the Data API for Aurora Serverless .	November 20, 2018
Aurora PostgreSQL version 2.1 (p. 1824)	Aurora PostgreSQL version 2.1 is available and compatible with PostgreSQL 10.5. For more information, see Version 2.1 .	November 20, 2018
TLS support for Aurora Serverless (p. 1824)	Aurora Serverless clusters support TLS/SSL encryption. For more information, see TLS/SSL for Aurora Serverless .	November 19, 2018
Custom endpoints (p. 1824)	You can now create endpoints that are associated with an arbitrary set of DB instances. This feature helps with load balancing and high availability for Aurora clusters where some DB instances have different capacity or configuration than others. You can use custom endpoints instead of connecting to a specific DB instance through its instance endpoint. For more information, see Amazon Aurora connection management .	November 12, 2018
IAM authentication support in Aurora PostgreSQL (p. 1824)	Aurora PostgreSQL now supports IAM authentication. For more information see IAM database authentication .	November 8, 2018
Aurora MySQL version 2.03.1 (p. 1181)	Aurora MySQL version 2.03.1 is available.	October 24, 2018

Custom parameter groups for restore and point in time recovery (p. 1824)	You can now specify a custom parameter group when you restore a snapshot or perform a point in time recovery operation. For more information, see Restoring from a DB cluster snapshot and Restoring a DB cluster to a specified time .	October 15, 2018
Aurora MySQL version 2.03 (p. 1182)	Aurora MySQL version 2.03 is available.	October 11, 2018
Aurora MySQL version 2.02.5 (p. 1184)	Aurora MySQL version 2.02.5 is available.	October 8, 2018
Aurora MySQL version 1.17.7 (p. 1220)	Aurora MySQL version 1.17.7 is available.	October 8, 2018
Deletion protection for Aurora DB clusters (p. 1824)	When you enable deletion protection for a DB cluster, the database cannot be deleted by any user. For more information, see Deleting a DB cluster .	September 26, 2018
Aurora PostgreSQL version 2.0 (p. 1824)	Aurora PostgreSQL version 2.0 is available and compatible with PostgreSQL 10.4. For more information, see Version 2.0 .	September 25, 2018
Stop/Start feature Aurora (p. 1824)	You can now stop or start an entire Aurora cluster with a single operation. For more information, see Stopping and starting an Aurora cluster .	September 24, 2018
Aurora MySQL version 2.02.4 (p. 1185)	Aurora MySQL version 2.02.4 is available.	September 21, 2018
Parallel query feature for Aurora MySQL (p. 1824)	Aurora MySQL now offers an option to parallelize I/O work for queries across the Aurora storage infrastructure. This feature speeds up data-intensive analytic queries, which are often the most time-consuming operations in a workload. For more information, see Working with parallel query for Aurora MySQL .	September 20, 2018
Aurora MySQL version 1.18.0 (p. 1218)	Aurora MySQL version 1.18.0 is available.	September 20, 2018
Aurora PostgreSQL version 1.3 (p. 1824)	Aurora PostgreSQL version 1.3 is now available and is compatible with PostgreSQL 9.6.9. For more information, see Version 1.3 .	September 11, 2018

[Aurora MySQL version 1.17.6 \(p. 1221\)](#)

Aurora MySQL version 1.17.6 is available.

September 6, 2018

[New guide \(p. 1824\)](#)

This is the first release of the *Amazon Aurora User Guide*.

August 31, 2018

AWS glossary

For the latest AWS terminology, see the [AWS glossary](#) in the *AWS General Reference*.