

---

# Amazon Aurora

## User Guide for Aurora



---

**Amazon Aurora: User Guide for Aurora**

Copyright © 2022 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 .....	8
Automatic minor version upgrades .....	8
How long Amazon Aurora major versions remain available .....	8
How often Amazon Aurora minor versions are released .....	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 .....	24
Aurora parallel queries .....	27
Amazon RDS Proxy .....	28
Aurora Serverless v2 .....	31
Aurora Serverless v1 .....	32
Data API for Aurora Serverless v1 .....	33
Aurora connection management .....	35
Types of Aurora endpoints .....	36
Viewing endpoints .....	37
Using the cluster endpoint .....	37
Using the reader endpoint .....	38
Using custom endpoints .....	38
Creating a custom endpoint .....	40
Viewing custom endpoints .....	42
Editing a custom endpoint .....	47
Deleting a custom endpoint .....	49
End-to-end AWS CLI example for custom endpoints .....	50
Using the instance endpoints .....	54
Endpoints and high availability .....	54
DB instance classes .....	56
DB instance class types .....	56
Supported DB engines .....	57
Determining DB instance class support in AWS Regions .....	61
Hardware specifications .....	64
Aurora storage and reliability .....	67
Overview of Aurora storage .....	67
Cluster volume contents .....	67
How storage resizes .....	67
Data billing .....	68

---

Reliability .....	68
Aurora security .....	70
Using SSL with Aurora DB clusters .....	71
High availability for Amazon Aurora .....	71
High availability for Aurora data .....	71
High availability for Aurora DB instances .....	71
High availability across AWS Regions with Aurora global databases .....	72
Fault tolerance .....	72
Replication with Aurora .....	73
Aurora Replicas .....	73
Aurora MySQL .....	74
Aurora PostgreSQL .....	75
DB instance billing for Aurora .....	75
On-Demand DB instances .....	77
Reserved DB instances .....	78
Setting up your environment .....	87
Get an AWS account and your root user credentials .....	87
Create an IAM user .....	87
Sign in as an IAM user .....	88
Create IAM user access keys .....	89
Determine requirements .....	89
Provide access to the DB cluster .....	90
Getting started .....	93
Creating an Aurora MySQL DB cluster and connecting to it .....	93
Create an Aurora MySQL DB cluster .....	93
Connect to an instance in a DB cluster .....	97
Delete the sample DB cluster, DB subnet group, and VPC .....	100
Creating an Aurora PostgreSQL DB cluster and connecting to it .....	100
Create an Aurora PostgreSQL DB cluster .....	101
Connect to an instance in an Aurora PostgreSQL DB cluster .....	104
Delete the sample DB cluster, DB subnet group, and VPC .....	106
Tutorial: Create a web server and an Amazon Aurora DB cluster .....	107
Create a DB cluster .....	108
Create a web server .....	113
Tutorials and sample code .....	123
Tutorials in this guide .....	123
Tutorials in other AWS guides .....	123
Tutorials and sample code in GitHub .....	124
Configuring your Aurora DB cluster .....	126
Creating a DB cluster .....	127
Prerequisites .....	127
Creating a DB cluster .....	131
Available settings .....	137
Settings that don't apply to Aurora for DB clusters .....	147
Settings that don't apply to Aurora DB instances .....	147
Creating resources with AWS CloudFormation .....	150
Aurora and AWS CloudFormation templates .....	150
Learn more about AWS CloudFormation .....	150
Using Aurora global databases .....	151
Overview of Aurora global databases .....	151
Advantages of Amazon Aurora global databases .....	152
Limitations of Aurora global databases .....	152
Getting started with Aurora global databases .....	154
Managing an Aurora global database .....	175
Connecting to an Aurora global database .....	180
Using write forwarding in an Aurora global database .....	181
Using failover in an Aurora global database .....	192

---

Monitoring an Aurora global database .....	202
Using Aurora global databases with other AWS services .....	205
Upgrading an Amazon Aurora global database .....	206
Connecting to a DB cluster .....	207
Connecting to Aurora MySQL .....	207
Connecting to Aurora PostgreSQL .....	212
Troubleshooting connections .....	214
Working with parameter groups .....	215
Working with DB cluster parameter groups .....	217
Working with DB parameter groups .....	228
Comparing parameter groups .....	238
Specifying DB parameters .....	238
Migrating data to a DB cluster .....	242
Aurora MySQL .....	242
Aurora PostgreSQL .....	242
Managing an Aurora DB cluster .....	243
Stopping and starting a cluster .....	244
Overview of stopping and starting a cluster .....	244
Limitations .....	244
Stopping a DB cluster .....	245
While a DB cluster is stopped .....	246
Starting a DB cluster .....	246
Modifying an Aurora DB cluster .....	248
Modifying the DB cluster by using the console, CLI, and API .....	248
Modify a DB instance in a DB cluster .....	249
Available settings .....	251
Settings that don't apply to Aurora DB clusters .....	268
Settings that don't apply to Aurora DB instances .....	269
Adding Aurora Replicas .....	270
Managing performance and scaling .....	274
Storage scaling .....	274
Instance scaling .....	278
Read scaling .....	278
Managing connections .....	278
Managing query execution plans .....	279
Cloning a volume for an Aurora DB cluster .....	280
Overview of Aurora cloning .....	280
Limitations of Aurora cloning .....	281
How Aurora cloning works .....	281
Creating an Aurora clone .....	284
Cross-account cloning .....	293
Integrating with AWS services .....	304
Aurora MySQL .....	304
Aurora PostgreSQL .....	304
Using Auto Scaling with Aurora replicas .....	305
Using machine learning with Aurora .....	320
Maintaining an Aurora DB cluster .....	321
Viewing pending maintenance .....	321
Applying updates .....	323
The maintenance window .....	324
Adjusting the maintenance window for a DB cluster .....	326
Automatic minor version upgrades for Aurora DB clusters .....	327
Choosing the frequency of Aurora MySQL maintenance updates .....	327
Rebooting an Aurora DB cluster or instance .....	329
Rebooting a DB instance within an Aurora cluster .....	329
Rebooting an Aurora cluster (Aurora PostgreSQL and Aurora MySQL before version 2.10) .....	330
Rebooting an Aurora MySQL cluster (version 2.10 and higher) .....	330

---

Checking uptime for Aurora clusters and instances .....	331
Examples of Aurora reboot operations .....	333
Deleting Aurora clusters and instances .....	345
Deleting an Aurora DB cluster .....	345
Deletion protection for Aurora clusters .....	349
Deleting a stopped Aurora cluster .....	350
Deleting Aurora MySQL clusters that are read replicas .....	350
The final snapshot when deleting a cluster .....	350
Deleting a DB instance from an Aurora DB cluster .....	350
Tagging RDS resources .....	352
Overview .....	352
Using tags for access control with IAM .....	353
Using tags to produce detailed billing reports .....	353
Adding, listing, and removing tags .....	354
Using the AWS Tag Editor .....	356
Copying tags to DB cluster snapshots .....	356
Tutorial: Use tags to specify which Aurora DB clusters to stop .....	357
Working with ARNs .....	360
Constructing an ARN .....	360
Getting an existing ARN .....	364
Aurora updates .....	367
Identifying your Amazon Aurora version .....	367
Backing up and restoring an Aurora DB cluster .....	368
Overview of backing up and restoring .....	369
Backups .....	369
Backup window .....	369
Restoring data .....	371
Backtrack .....	371
Backup storage .....	372
Creating a DB cluster snapshot .....	373
Determining whether the snapshot is available .....	374
Restoring from a DB cluster snapshot .....	375
Parameter groups .....	375
Security groups .....	375
Aurora considerations .....	375
Restoring from a snapshot .....	376
Copying a DB cluster snapshot .....	378
Limitations .....	378
Snapshot retention .....	378
Copying shared snapshots .....	379
Handling encryption .....	379
Incremental snapshot copying .....	379
Cross-Region copying .....	379
Parameter groups .....	380
Copying a DB cluster snapshot .....	380
Sharing a DB cluster snapshot .....	388
Sharing public snapshots .....	388
Sharing encrypted snapshots .....	389
Sharing a snapshot .....	391
Exporting DB cluster snapshot data to Amazon S3 .....	396
Limitations .....	397
Overview of exporting snapshot data .....	397
Setting up access to an S3 bucket .....	398
Using a cross-account KMS key .....	401
Exporting a snapshot to an S3 bucket .....	402
Monitoring snapshot exports .....	404
Canceling a snapshot export .....	406

---

Failure messages .....	406
Troubleshooting PostgreSQL permissions errors .....	407
File naming convention .....	408
Data conversion .....	408
Point-in-time recovery .....	415
Deleting a DB cluster snapshot .....	417
Deleting a DB cluster snapshot .....	417
Tutorial: Restore a DB cluster from a snapshot .....	419
Restoring a DB cluster using the console .....	419
Restoring a DB cluster using the AWS CLI .....	422
Monitoring metrics in an Aurora DB cluster .....	427
Overview of monitoring .....	428
Monitoring plan .....	428
Performance baseline .....	428
Performance guidelines .....	428
Monitoring tools .....	429
Viewing cluster status and recommendations .....	432
Viewing a DB cluster .....	433
Viewing DB cluster status .....	439
Viewing DB instance status in an Aurora cluster .....	441
Viewing Amazon Aurora recommendations .....	444
Viewing metrics in the Amazon RDS console .....	449
Monitoring Aurora with CloudWatch .....	452
Overview of Amazon Aurora and Amazon CloudWatch .....	453
Viewing CloudWatch metrics .....	454
Creating CloudWatch alarms .....	459
Monitoring DB load with Performance Insights .....	461
Overview of Performance Insights .....	461
Turning Performance Insights on and off .....	466
Turning on the Performance Schema for Aurora MySQL .....	469
Performance Insights policies .....	473
Analyzing metrics with the Performance Insights dashboard .....	476
Retrieving metrics with the Performance Insights API .....	496
Logging Performance Insights calls using AWS CloudTrail .....	510
Analyzing performance with DevOps Guru for RDS .....	512
Benefits of DevOps Guru for RDS .....	512
How DevOps Guru for RDS works .....	513
Setting up DevOps Guru for RDS .....	513
Monitoring the OS with Enhanced Monitoring .....	518
Overview of Enhanced Monitoring .....	518
Setting up and enabling Enhanced Monitoring .....	519
Viewing OS metrics in the RDS console .....	523
Viewing OS metrics using CloudWatch Logs .....	524
Aurora metrics reference .....	525
CloudWatch metrics for Aurora .....	525
CloudWatch dimensions for Aurora .....	541
Availability of Aurora metrics in the Amazon RDS console .....	542
CloudWatch metrics for Performance Insights .....	544
Counter metrics for Performance Insights .....	546
SQL statistics for Performance Insights .....	554
OS metrics in Enhanced Monitoring .....	558
Monitoring events, logs, and database activity streams .....	563
Viewing logs, events, and streams in the Amazon RDS console .....	563
Monitoring Aurora events .....	568
Overview of events for Aurora .....	568
Viewing Amazon RDS events .....	569
Working with Amazon RDS event notification .....	572

Creating a rule that triggers on an Amazon Aurora event .....	587
Amazon RDS event categories and event messages .....	590
Monitoring Aurora logs .....	597
Viewing and listing database log files .....	597
Downloading a database log file .....	598
Watching a database log file .....	599
Publishing to CloudWatch Logs .....	601
Reading log file contents using REST .....	602
MySQL database log files .....	604
PostgreSQL database log files .....	610
Monitoring Aurora API calls in CloudTrail .....	615
CloudTrail integration with Amazon Aurora .....	615
Amazon Aurora log file entries .....	615
Monitoring Aurora with Database Activity Streams .....	619
Overview .....	619
Aurora MySQL network prerequisites .....	622
Starting a database activity stream .....	623
Getting the activity stream status .....	624
Stopping a database activity stream .....	625
Monitoring activity streams .....	626
Managing access to activity streams .....	648
Working with Aurora MySQL .....	651
Overview of Aurora MySQL .....	651
Amazon Aurora MySQL performance enhancements .....	651
Aurora MySQL and spatial data .....	652
Aurora MySQL version 3 compatible with MySQL 8.0 .....	653
Aurora MySQL version 2 compatible with MySQL 5.7 .....	680
Security with Aurora MySQL .....	681
Master user privileges with Aurora MySQL .....	682
Using SSL/TLS with Aurora MySQL DB clusters .....	683
Updating applications for new SSL/TLS certificates .....	687
Determining whether any applications are connecting to your Aurora MySQL DB cluster using SSL .....	688
Determining whether a client requires certificate verification to connect .....	688
Updating your application trust store .....	689
Example Java code for establishing SSL connections .....	690
Migrating data to Aurora MySQL .....	690
Migrating from an external MySQL database to Aurora MySQL .....	692
Migrating from a MySQL DB instance to Aurora MySQL .....	706
Managing Aurora MySQL .....	721
Managing performance and scaling for Amazon Aurora MySQL .....	721
Backtracking a DB cluster .....	725
Testing Amazon Aurora using fault injection queries .....	738
Altering tables in Amazon Aurora using fast DDL .....	741
Displaying volume status for an Aurora DB cluster .....	746
Tuning Aurora MySQL with wait events and thread states .....	746
Essential concepts for Aurora MySQL tuning .....	747
Tuning Aurora MySQL with wait events .....	749
Tuning Aurora MySQL with thread states .....	785
Parallel query for Aurora MySQL .....	790
Overview of parallel query .....	791
Planning for a parallel query cluster .....	794
Creating a parallel query cluster .....	795
Turning parallel query on and off .....	799
Upgrading a parallel query cluster .....	802
Performance tuning .....	803
Creating schema objects .....	803

---

Verifying parallel query usage .....	803
Monitoring .....	806
Parallel query and SQL constructs .....	810
Advanced Auditing with Aurora MySQL .....	823
Enabling Advanced Auditing .....	823
Viewing audit logs .....	825
Audit log details .....	826
Single-master replication with Aurora MySQL .....	826
Aurora replicas .....	827
Options .....	828
Performance .....	828
Zero-downtime restart (ZDR) .....	829
Monitoring .....	830
Replicating Amazon Aurora MySQL DB clusters across AWS Regions .....	831
Replication between Aurora and MySQL or between Aurora and another Aurora DB cluster (binary log replication) .....	841
Using GTID-based replication .....	863
Working with multi-master clusters .....	867
Overview of multi-master clusters .....	867
Creating a multi-master cluster .....	873
Managing multi-master clusters .....	875
Application considerations .....	878
Performance considerations .....	887
Approaches to multi-master clusters .....	889
Integrating Aurora MySQL with AWS services .....	890
Authorizing Aurora MySQL to access AWS services .....	891
Loading data from text files in Amazon S3 .....	903
Saving data into text files in Amazon S3 .....	911
Invoking a Lambda function from Aurora MySQL .....	917
Publishing Aurora MySQL logs to CloudWatch Logs .....	924
Using machine learning with Aurora MySQL .....	927
Aurora MySQL lab mode .....	939
Aurora lab mode features .....	940
Best practices with Amazon Aurora MySQL .....	940
Determining which DB instance you are connected to .....	941
Best practices for Aurora MySQL performance and scaling .....	941
Best practices for Aurora MySQL high availability .....	947
Recommendations for MySQL features .....	948
Aurora MySQL reference .....	949
Configuration parameters .....	949
MySQL parameters that don't apply to Aurora MySQL .....	969
MySQL status variables that don't apply to Aurora MySQL .....	970
Aurora MySQL wait events .....	971
Aurora MySQL thread states .....	975
Aurora MySQL isolation levels .....	978
Aurora MySQL hints .....	982
Stored procedures .....	984
Aurora MySQL updates .....	990
Version Numbers and Special Versions .....	991
Preparing for Aurora MySQL version 1 end of life .....	994
Upgrading Amazon Aurora MySQL DB clusters .....	996
Database engine updates for Amazon Aurora MySQL version 3 .....	1016
Database engine updates for Amazon Aurora MySQL version 2 .....	1016
Database engine updates for Amazon Aurora MySQL version 1 .....	1017
Database engine updates for Aurora MySQL Serverless clusters .....	1017
MySQL bugs fixed by Aurora MySQL database engine updates .....	1017
Security vulnerabilities fixed in Amazon Aurora MySQL .....	1017

---

Working with Aurora PostgreSQL .....	1018
Security with Aurora PostgreSQL .....	1018
Understanding PostgreSQL roles and permissions .....	1019
Securing Aurora PostgreSQL data with SSL/TLS .....	1029
Updating applications for new SSL/TLS certificates .....	1032
Determining whether applications are connecting to Aurora PostgreSQL DB clusters using SSL .....	1033
Determining whether a client requires certificate verification in order to connect .....	1033
Updating your application trust store .....	1034
Using SSL/TLS connections for different types of applications .....	1034
Using Kerberos authentication .....	1035
Region and version availability .....	1035
Overview of Kerberos authentication .....	1036
Setting up .....	1037
Managing a DB cluster in a Domain .....	1046
Connecting with Kerberos authentication .....	1047
Migrating data to Aurora PostgreSQL .....	1048
Migrating an RDS for PostgreSQL DB instance using a snapshot .....	1049
Migrating an RDS for PostgreSQL DB instance using an Aurora read replica .....	1054
Using Babelfish for Aurora PostgreSQL .....	1063
Understanding Babelfish architecture and configuration .....	1064
Creating a Babelfish for Aurora PostgreSQL DB cluster .....	1086
Migrating a SQL Server database to Babelfish .....	1093
Connecting to a Babelfish DB cluster .....	1097
Working with Babelfish .....	1106
Troubleshooting Babelfish .....	1123
Turning off Babelfish .....	1125
Babelfish versions .....	1126
Babelfish reference .....	1128
Managing Aurora PostgreSQL .....	1143
Scaling Aurora PostgreSQL DB instances .....	1143
Maximum connections .....	1144
Temporary storage limits .....	1146
Testing Amazon Aurora PostgreSQL by using fault injection queries .....	1147
Displaying volume status for an Aurora DB cluster .....	1151
Specifying the RAM disk for the stats_temp_directory .....	1152
Tuning with wait events for Aurora PostgreSQL .....	1152
Essential concepts for Aurora PostgreSQL tuning .....	1153
Aurora PostgreSQL wait events .....	1156
Client:ClientRead .....	1158
Client:ClientWrite .....	1160
CPU .....	1161
IO:BuffFileRead and IO:BuffFileWrite .....	1166
IO:DataFileRead .....	1171
IO:XactSync .....	1177
ipc:damrecordtxack .....	1179
Lock:advisory .....	1180
Lock:extend .....	1182
Lock:Relation .....	1184
Lock:transactionid .....	1187
Lock:tuple .....	1189
lwlock:buffer_content (BufferContent) .....	1192
LWLock:buffer_mapping .....	1193
LWLock:BufferIO .....	1195
LWLock:lock_manager .....	1196
Timeout:PgSleep .....	1199
Best practices with Aurora PostgreSQL .....	1200
Troubleshooting storage issues .....	1200

---

Fast failover .....	1201
Fast recovery after failover .....	1208
Managing connection churn .....	1212
Tuning memory parameters for Aurora PostgreSQL .....	1218
Replication with Aurora PostgreSQL .....	1224
Aurora Replicas .....	1224
Monitoring replication .....	1224
Using logical replication .....	1225
Integrating Aurora PostgreSQL with AWS services .....	1230
Importing data from Amazon S3 into Aurora PostgreSQL .....	1230
Exporting PostgreSQL data to Amazon S3 .....	1243
Invoking a Lambda function from Aurora PostgreSQL .....	1254
Publishing Aurora PostgreSQL logs to CloudWatch Logs .....	1265
Using machine learning with Aurora PostgreSQL .....	1272
Managing query execution plans for Aurora PostgreSQL .....	1290
Turning on query plan management .....	1291
Upgrading query plan management .....	1292
Query plan management basics .....	1292
Best practices for Aurora PostgreSQL query plan management .....	1295
Examining Aurora PostgreSQL query plans in the dba_plans view .....	1297
Capturing Aurora PostgreSQL execution plans .....	1299
Using Aurora PostgreSQL managed plans .....	1301
Maintaining Aurora PostgreSQL execution plans .....	1304
Parameter reference for Aurora PostgreSQL query plan management .....	1308
Function reference for Aurora PostgreSQL query plan management .....	1311
Working with extensions and foreign data wrappers .....	1318
Managing large objects more efficiently with the lo module .....	1318
Managing spatial data with PostGIS .....	1320
Scheduling maintenance with the pg_cron extension .....	1327
Managing partitions with the pg_partman extension .....	1333
Supported foreign data wrappers .....	1337
Aurora PostgreSQL reference .....	1341
Aurora PostgreSQL collations for EBCDIC and other mainframe migrations .....	1341
Aurora PostgreSQL functions reference .....	1343
Aurora PostgreSQL parameters .....	1369
Aurora PostgreSQL wait events .....	1395
Aurora PostgreSQL updates .....	1413
Identifying versions of Amazon Aurora PostgreSQL .....	1413
Aurora PostgreSQL releases .....	1414
Extension versions for Aurora PostgreSQL .....	1415
Upgrading the PostgreSQL DB engine .....	1415
Using a long-term support (LTS) release .....	1429
Using RDS Proxy .....	1430
Region and version availability .....	1430
Quotas and limitations .....	1431
Planning where to use RDS Proxy .....	1432
RDS Proxy concepts and terminology .....	1433
Overview of RDS Proxy concepts .....	1433
Connection pooling .....	1434
Security .....	1434
Failover .....	1436
Transactions .....	1436
Getting started with RDS Proxy .....	1437
Setting up network prerequisites .....	1437
Setting up database credentials in Secrets Manager .....	1438
Setting up IAM policies .....	1440
Creating an RDS Proxy .....	1442

Viewing an RDS Proxy .....	1445
Connecting through RDS Proxy .....	1446
Managing an RDS Proxy .....	1448
Modifying an RDS Proxy .....	1449
Adding a database user .....	1453
Changing database passwords .....	1453
Configuring connection settings .....	1453
Avoiding pinning .....	1455
Deleting an RDS Proxy .....	1457
Working with RDS Proxy endpoints .....	1458
Overview of proxy endpoints .....	1458
Using reader endpoints with Aurora clusters .....	1459
Accessing Aurora and RDS databases across VPCs .....	1462
Creating a proxy endpoint .....	1463
Viewing proxy endpoints .....	1464
Modifying a proxy endpoint .....	1465
Deleting a proxy endpoint .....	1466
Limitations for proxy endpoints .....	1467
Monitoring RDS Proxy with CloudWatch .....	1467
Working with RDS Proxy events .....	1472
RDS Proxy events .....	1472
RDS Proxy examples .....	1473
Troubleshooting RDS Proxy .....	1475
Verifying connectivity for a proxy .....	1476
Common issues and solutions .....	1477
Using RDS Proxy with AWS CloudFormation .....	1481
Using Aurora Serverless v2 .....	1482
Aurora Serverless v2 use cases .....	1482
Converting provisioned workloads .....	1483
Advantages of Aurora Serverless v2 .....	1483
How Aurora Serverless v2 works .....	1484
Overview .....	1485
Cluster configurations .....	1486
Capacity .....	1486
Scaling .....	1487
High availability .....	1488
Storage .....	1489
Configuration parameters .....	1489
Requirements for Aurora Serverless v2 .....	1490
Aurora Serverless v2 is available in certain AWS Regions .....	1490
Aurora Serverless v2 requires minimum engine versions .....	1490
Clusters that use Aurora Serverless v2 must have a capacity range specified .....	1491
Some provisioned features aren't supported in Aurora Serverless v2 .....	1491
Some Aurora Serverless v2 aspects are different from Aurora Serverless v1 .....	1492
Getting started with Aurora Serverless v2 .....	1492
Using Aurora Serverless v2 with an existing cluster .....	1492
Switching from a provisioned cluster .....	1495
Moving from Aurora Serverless v1 .....	1499
Upgrading from Aurora Serverless v2 (preview) .....	1506
Migrating from an on-premises database to Aurora Serverless v2 .....	1506
Creating a cluster for Aurora Serverless v2 .....	1506
Managing Aurora Serverless v2 .....	1509
Setting the Aurora Serverless v2 capacity range for a cluster .....	1510
Checking the Aurora Serverless v2 capacity range .....	1513
Creating an Aurora Serverless v2 writer .....	1515
Adding an Aurora Serverless v2 reader .....	1517
Converting from provisioned to Aurora Serverless v2 .....	1518

Converting from Aurora Serverless v2 to provisioned .....	1520
Choosing the promotion tier for an Aurora Serverless v2 reader .....	1521
Using TLS/SSL with Aurora Serverless v2 .....	1521
Viewing Aurora Serverless v2 writers and readers .....	1523
Logging for Aurora Serverless v2 .....	1523
Performance and scaling for Aurora Serverless v2 .....	1526
Choosing the capacity range .....	1527
Working with parameter groups for Aurora Serverless v2 .....	1535
Avoiding out-of-memory errors .....	1538
Important CloudWatch metrics .....	1539
Monitoring Aurora Serverless v2 performance with Performance Insights .....	1542
Using Aurora Serverless v1 .....	1543
Advantages of Aurora Serverless v1 .....	1543
Use cases for Aurora Serverless v1 .....	1544
Limitations of Aurora Serverless v1 .....	1544
Configuration requirements for Aurora Serverless v1 .....	1545
Using TLS/SSL with Aurora Serverless v1 .....	1546
Supported cipher suites for connections to Aurora Serverless v1 DB clusters .....	1548
How Aurora Serverless v1 works .....	1548
Aurora Serverless v1 architecture .....	1549
Autoscaling .....	1549
Timeout action .....	1550
Pause and resume .....	1551
Determining max_connections .....	1552
Parameter groups .....	1554
Logging .....	1556
Maintenance .....	1558
Failover .....	1559
Snapshots .....	1559
Creating an Aurora Serverless v1 DB cluster .....	1559
Restoring an Aurora Serverless v1 DB cluster .....	1566
Modifying an Aurora Serverless v1 DB cluster .....	1570
Scaling Aurora Serverless v1 DB cluster capacity manually .....	1574
Viewing Aurora Serverless v1 DB clusters .....	1576
Monitoring Aurora Serverless v1 DB clusters with CloudWatch .....	1577
Deleting an Aurora Serverless v1 DB cluster .....	1578
Aurora Serverless v1 and Aurora database engine versions .....	1580
Aurora MySQL Serverless .....	1580
Aurora PostgreSQL Serverless .....	1580
Using the Data API .....	1581
Data API availability .....	1581
Authorizing access .....	1582
Tag-based authorization .....	1582
Storing credentials in a secret .....	1584
Enabling the Data API .....	1584
Creating an Amazon VPC endpoint .....	1586
Calling the Data API .....	1588
Calling the Data API with the AWS CLI .....	1590
Calling the Data API from a Python application .....	1596
Calling the Data API from a Java application .....	1599
Using the Java client library .....	1602
Downloading the Java client library for Data API .....	1602
Java client library examples .....	1602
Processing query results in JSON format .....	1603
Retrieving query results in JSON format .....	1604
Data Type Mapping .....	1604
Troubleshooting .....	1605

Examples .....	1605
Troubleshooting Data API issues .....	1608
Transaction <transaction_ID> is not found .....	1609
Packet for query is too large .....	1609
Database response exceeded size limit .....	1609
HttpEndpoint is not enabled for cluster <cluster_ID> .....	1609
Logging Data API calls with AWS CloudTrail .....	1610
Working with Data API information in CloudTrail .....	1610
Understanding Data API log file entries .....	1611
Excluding Data API events from a CloudTrail trail .....	1611
Using the query editor .....	1613
Availability of the query editor .....	1613
Authorizing access .....	1613
Running queries .....	1614
DBQMS API reference .....	1617
CreateFavoriteQuery .....	1618
CreateQueryHistory .....	1618
CreateTab .....	1618
DeleteFavoriteQueries .....	1618
DeleteQueryHistory .....	1618
DeleteTab .....	1618
DescribeFavoriteQueries .....	1618
DescribeQueryHistory .....	1618
DescribeTabs .....	1618
GetQueryString .....	1619
UpdateFavoriteQuery .....	1619
UpdateQueryHistory .....	1619
UpdateTab .....	1619
Best practices with Aurora .....	1620
Basic operational guidelines for Amazon Aurora .....	1620
DB instance RAM recommendations .....	1620
Monitoring Amazon Aurora .....	1621
Working with DB parameter groups and DB cluster parameter groups .....	1621
Amazon Aurora best practices presentation video .....	1621
Performing an Aurora proof of concept .....	1622
Overview of an Aurora proof of concept .....	1622
1. Identify your objectives .....	1622
2. Understand your workload characteristics .....	1623
3. Practice with the console or CLI .....	1624
Practice with the console .....	1624
Practice with the AWS CLI .....	1624
4. Create your Aurora cluster .....	1625
5. Set up your schema .....	1626
6. Import your data .....	1626
7. Port your SQL code .....	1627
8. Specify configuration settings .....	1627
9. Connect to Aurora .....	1628
10. Run your workload .....	1629
11. Measure performance .....	1629
12. Exercise Aurora high availability .....	1631
13. What to do next .....	1632
Security .....	1634
Database authentication .....	1635
Password authentication .....	1636
IAM database authentication .....	1636
Kerberos authentication .....	1636
Data protection .....	1637

---

Data encryption .....	1637
Internetwork traffic privacy .....	1652
Identity and access management .....	1653
Audience .....	1653
Authenticating with identities .....	1653
Managing access using policies .....	1655
How Amazon Aurora works with IAM .....	1657
Identity-based policy examples .....	1662
AWS managed policies .....	1673
Policy updates .....	1679
Cross-service confused deputy prevention .....	1681
IAM database authentication .....	1683
Troubleshooting .....	1710
Logging and monitoring .....	1711
Compliance validation .....	1714
Resilience .....	1715
Backup and restore .....	1715
Replication .....	1715
Failover .....	1715
Infrastructure security .....	1717
Security groups .....	1717
Public accessibility .....	1717
VPC endpoints (AWS PrivateLink) .....	1718
Considerations .....	1718
Availability .....	1718
Creating an interface VPC endpoint .....	1719
Creating a VPC endpoint policy .....	1719
Security best practices .....	1720
Controlling access with security groups .....	1721
Overview of VPC security groups .....	1721
Security group scenario .....	1721
Creating a VPC security group .....	1723
Associating with a DB cluster .....	1723
Master user account privileges .....	1723
Service-linked roles .....	1724
Service-linked role permissions for Amazon Aurora .....	1724
Using Amazon Aurora with Amazon VPC .....	1729
Working with a DB cluster in a VPC .....	1729
Scenarios for accessing a DB cluster in a VPC .....	1739
Tutorial: Create a VPC for use with a DB cluster (IPv4 only) .....	1744
Tutorial: Create a VPC for use with a DB cluster (dual-stack mode) .....	1749
Quotas and constraints .....	1756
Quotas in Amazon Aurora .....	1756
Naming constraints in Amazon Aurora .....	1759
Amazon Aurora size limits .....	1759
Troubleshooting .....	1761
Can't connect to DB instance .....	1761
Testing the DB instance connection .....	1762
Troubleshooting connection authentication .....	1763
Security issues .....	1763
Error message "failed to retrieve account attributes, certain console functions may be impaired." .....	1763
Resetting the DB instance owner password .....	1763
DB instance outage or reboot .....	1764
Parameter changes not taking effect .....	1764
Aurora freeable memory issues .....	1764
Aurora MySQL out of memory issues .....	1765

Aurora MySQL replication issues .....	1766
Diagnosing and resolving lag between read replicas .....	1766
Diagnosing and resolving a MySQL read replication failure .....	1767
Replication stopped error .....	1768
Amazon RDS API reference .....	1769
Using the Query API .....	1769
Query parameters .....	1769
Query request authentication .....	1769
Troubleshooting applications .....	1770
Retrieving errors .....	1770
Troubleshooting tips .....	1770
Document history .....	1771
AWS glossary .....	1801

# 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 \(p. 87\)](#), 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 \(p. 35\)](#)
- [Aurora DB instance classes \(p. 56\)](#)
- [Amazon Aurora storage and reliability \(p. 67\)](#)

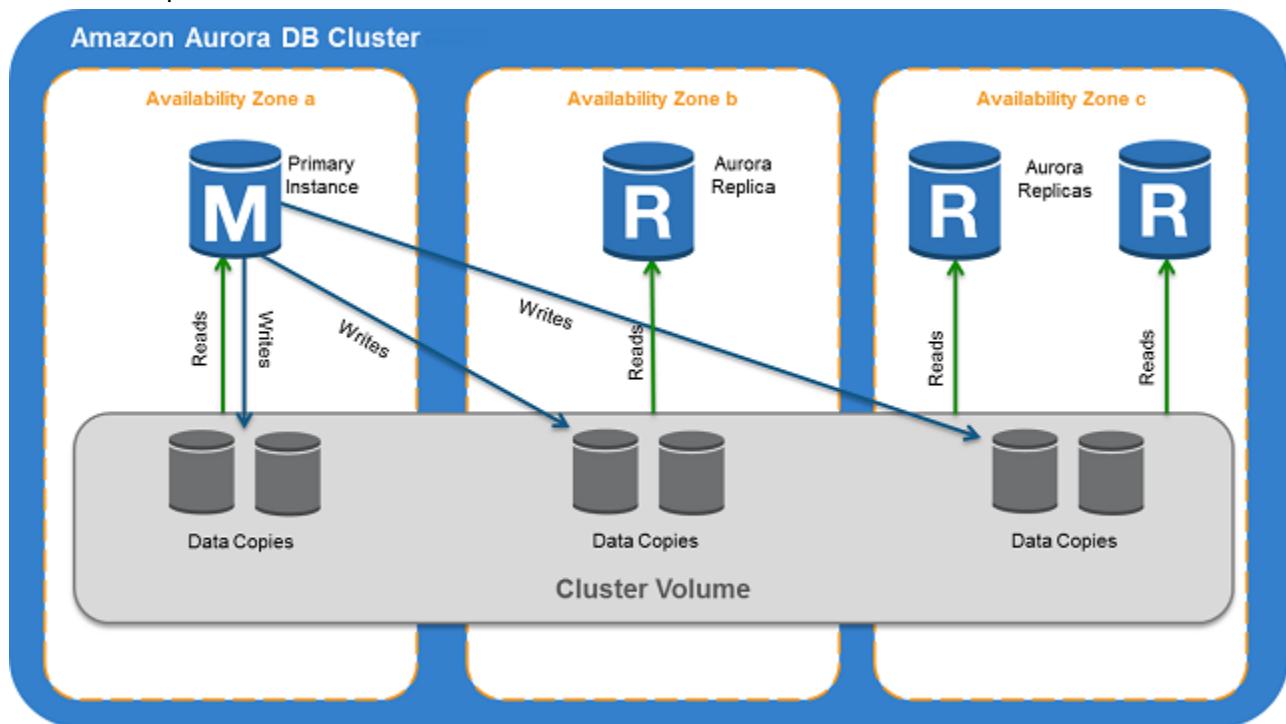
- [Amazon Aurora security \(p. 70\)](#)
- [High availability for Amazon Aurora \(p. 71\)](#)
- [Replication with Amazon Aurora \(p. 73\)](#)
- [DB instance billing for Aurora \(p. 75\)](#)

# 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 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 \(p. 8\)](#)
- [Automatic minor version upgrades \(p. 8\)](#)
- [How long Amazon Aurora major versions remain available \(p. 8\)](#)
- [How often Amazon Aurora minor versions are released \(p. 9\)](#)
- [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 \(p. 651\)](#). For a detailed list of available versions, see [Database engine updates for Amazon Aurora MySQL \(p. 990\)](#).
- Amazon Aurora PostgreSQL-Compatible Edition. For usage information, see [Working with Amazon Aurora PostgreSQL \(p. 1018\)](#). For a detailed list of available versions, see [Amazon Aurora PostgreSQL updates \(p. 1413\)](#).

## 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 \(p. 992\)](#) and [Identifying versions of Amazon Aurora PostgreSQL \(p. 1413\)](#).

## 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. Applies to PostgreSQL 10.17 and older versions only. For version 10.18 and higher versions, the Aurora version is the same as the <i>major.minor</i> version of the PostgreSQL community version, with a third digit in <i>patch</i> location.
PostgreSQL 11	Aurora PostgreSQL 3. Applies to PostgreSQL 11.12 and older versions only. For version 11.13 and higher versions, the Aurora version is the same as the <i>major.minor</i> version of the PostgreSQL community version, with a third digit in the <i>patch</i> location.
PostgreSQL 12	Aurora PostgreSQL 4. Applies to PostgreSQL 12.7 and older versions only. For version 12.8 and higher versions, the Aurora version is the same

Community major version	Aurora major version
	as the <i>major.minor</i> version of the PostgreSQL community version, with a third digit in the <i>patch</i> location.
PostgreSQL 13	Aurora PostgreSQL 4. Applies to PostgreSQL 13.2 and older versions only. For version 13.3 and higher versions, the Aurora version is the same as the <i>major.minor</i> version of the PostgreSQL community version, with a third digit in the <i>patch</i> location.
PostgreSQL 14	Aurora PostgreSQL 14.3. The Aurora version is the same as the <i>major.minor</i> version of the PostgreSQL community version, with a third digit in the <i>patch</i> location when patches to Aurora are released.

## 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 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 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 \(p. 321\)](#).

## 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 [Release Notes for Aurora MySQL](#). For Aurora PostgreSQL release notes, see [Release Notes for Aurora PostgreSQL](#).

## 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 \(p. 127\)](#). To learn how to change the version of an existing Aurora cluster, see [Modifying an Amazon Aurora DB cluster \(p. 248\)](#).

## 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 \(p. 997\)](#) and [How to perform minor version upgrades and apply patches \(p. 1425\)](#).

## How long Amazon Aurora major versions remain available

Amazon Aurora major versions remain 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 earliest date that an upgrade to a newer version might be required. If Amazon extends support for an Aurora version for longer than originally stated, we plan to update this table to reflect the later date.

Database community version	Aurora version	Expected date for upgrading to a newer version
MySQL 5.6	1	February 28, 2023, 00:00:00 UTC
MySQL 5.7	2	October 31, 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 <a href="#">Amazon Aurora major versions (p. 6)</a> .	January 31, 2023
PostgreSQL 11		January 31, 2024
PostgreSQL 12		February 28, 2025
PostgreSQL 13		January 31, 2026
PostgreSQL 14	The Aurora version is the same as the <i>major.minor</i> version of the PostgreSQL community version, with a third digit in the <i>patch</i> location.	January 31, 2027

Before we ask that you upgrade to a newer major version and to help you plan, 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 with 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.

## 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 critical 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 any DB instance in your Aurora cluster.

For the version numbers of all Aurora LTS versions, see [Aurora MySQL long-term support \(LTS\) releases \(p. 993\)](#) and [Aurora PostgreSQL long-term support \(LTS\) releases \(p. 1429\)](#).

## 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 deprecation, 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 \(p. 996\)](#) and [Upgrading the PostgreSQL DB engine for Aurora PostgreSQL \(p. 1415\)](#).

## 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 Amazon Aurora DB cluster \(p. 280\)](#) and [Creating a DB cluster snapshot \(p. 373\)](#).

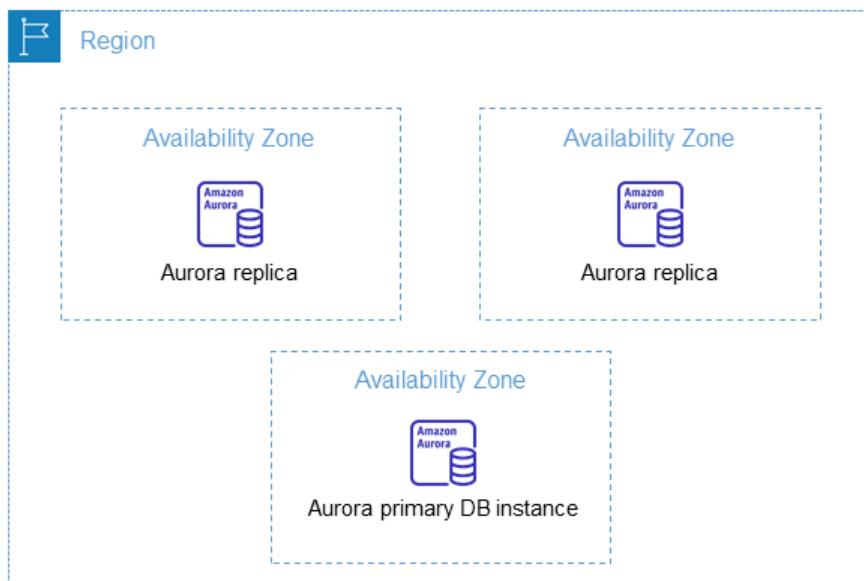
# Regions and Availability Zones

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	

<b>Region Name</b>	<b>Region</b>	<b>Endpoint</b>	<b>Protocol</b>	
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.

<b>Region Name</b>	<b>Region</b>	<b>Endpoint</b>	<b>Protocol</b>	
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.

To learn how to specify the AZ when you create a cluster or add instances to it, see [Configure the network for the DB cluster \(p. 127\)](#).

## 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 \(p. 217\)](#).

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 \(p. 24\)](#)
- [Aurora parallel queries \(p. 27\)](#)
- [Amazon RDS Proxy \(p. 28\)](#)
- [Aurora Serverless v2 \(p. 31\)](#)
- [Aurora Serverless v1 \(p. 32\)](#)
- [Data API for Aurora Serverless v1 \(p. 33\)](#)

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 \(p. 725\)](#).

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 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. For more information, see [Using Amazon Aurora global databases \(p. 151\)](#).

### Topics

- [Aurora global databases with Aurora MySQL \(p. 21\)](#)
- [Aurora global databases with Aurora PostgreSQL \(p. 22\)](#)

## Aurora global databases with Aurora MySQL

Following are the supported engines and Region availability for Aurora global databases with Aurora MySQL.

Region	Aurora MySQL 5.6	Aurora MySQL 5.7	Aurora MySQL 8.0
US East (Ohio)	Version 5.6.10a; version 1.22 and higher	Version 2.07 and higher	Version 3.01.0 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
US West (N. California)	Version 5.6.10a; version 1.22 and higher	Version 2.07 and higher	Version 3.01.0 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
Africa (Cape Town)	-	-	-
Asia Pacific (Hong Kong)	-	-	-
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
Asia Pacific (Osaka)	Version 1.22.3 and higher	Version 2.07.3 and higher	Version 3.01.0 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

Region	Aurora MySQL 5.6	Aurora MySQL 5.7	Aurora MySQL 8.0
Asia Pacific (Singapore)	Version 5.6.10a; Version 1.22 and higher	Version 2.07 and higher	Version 3.01.0 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
Asia Pacific (Tokyo)	Version 5.6.10a; Version 1.22 and higher	Version 2.07 and higher	Version 3.01.0 and higher
Canada (Central)	Version 5.6.10a; Version 1.22 and higher	Version 2.07 and higher	Version 3.01.0 and higher
China (Beijing)	Version 1.22.2 and higher	Version 2.07.2 and higher	Version 3.01.0 and higher
China (Ningxia)	Version 1.22.2 and higher	Version 2.07.2 and higher	Version 3.01.0 and higher
Europe (Frankfurt)	Version 5.6.10a; Version 1.22 and higher	Version 2.07 and higher	Version 3.01.0 and higher
Europe (Ireland)	Version 5.6.10a; Version 1.22 and higher	Version 2.07 and higher	Version 3.01.0 and higher
Europe (London)	Version 5.6.10a; Version 1.22 and higher	Version 2.07 and higher	Version 3.01.0 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
Europe (Stockholm)	Version 1.22.2 and higher	Version 2.07.0 and higher	Version 3.01.0 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
AWS GovCloud (US-East)	Version 5.6.10a; version 1.22 and higher	Version 2.07 and higher	Version 3.01.0 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

## Aurora global databases with Aurora PostgreSQL

Following are the supported engines and Region availability for Aurora global databases with Aurora PostgreSQL.

Region	Aurora PostgreSQL 10	Aurora PostgreSQL 11	Aurora PostgreSQL 12	Aurora PostgreSQL 13	Aurora PostgreSQL 14
US East (Ohio)	Version 10.14 and higher	Version 11.9 and higher	Version 12.4 and higher	Version 13.3 and higher	Version 14.3 and higher
US East (N. Virginia)	Version 10.14 and higher	Version 11.9 and higher	Version 12.4 and higher	Version 13.3 and higher	Version 14.3 and higher

Region	Aurora PostgreSQL 10	Aurora PostgreSQL 11	Aurora PostgreSQL 12	Aurora PostgreSQL 13	Aurora PostgreSQL 14
US West (N. California)	Version 10.14 and higher	Version 11.9 and higher	Version 12.4 and higher	Version 13.3 and higher	Version 14.3 and higher
US West (Oregon)	Version 10.14 and higher	Version 11.9 and higher	Version 12.4 and higher	Version 13.3 and higher	Version 14.3 and higher
Africa (Cape Town)	-	-	-	-	-
Asia Pacific (Hong Kong)	-	-	-	-	-
Asia Pacific (Jakarta)	-	-	-	-	-
Asia Pacific (Mumbai)	Version 10.14 and higher	Version 11.9 and higher	Version 12.4 and higher	Version 13.3 and higher	Version 14.3 and higher
Asia Pacific (Osaka)	Version 10.14 and higher	Version 11.9 and higher	Version 12.4 and higher	Version 13.3 and higher	Version 14.3 and higher
Asia Pacific (Seoul)	Version 10.14 and higher	Version 11.9 and higher	Version 12.4 and higher	Version 13.3 and higher	Version 14.3 and higher
Asia Pacific (Singapore)	Version 10.14 and higher	Version 11.9 and higher	Version 12.4 and higher	Version 13.3 and higher	Version 14.3 and higher
Asia Pacific (Sydney)	Version 10.14 and higher	Version 11.9 and higher	Version 12.4 and higher	Version 13.3 and higher	Version 14.3 and higher
Asia Pacific (Tokyo)	Version 10.14 and higher	Version 11.9 and higher	Version 12.4 and higher	Version 13.3 and higher	Version 14.3 and higher
Canada (Central)	Version 10.14 and higher	Version 11.9 and higher	Version 12.4 and higher	Version 13.3 and higher	Version 14.3 and higher
China (Beijing)	Version 10.14 and higher	Version 11.9 and higher	Version 12.4 and higher	Version 13.3 and higher	Version 14.3 and higher
China (Ningxia)	Version 10.14 and higher	Version 11.9 and higher	Version 12.4 and higher	Version 13.3 and higher	Version 14.3 and higher
Europe (Frankfurt)	Version 10.14 and higher	Version 11.9 and higher	Version 12.4 and higher	Version 13.3 and higher	Version 14.3 and higher
Europe (Ireland)	Version 10.14 and higher	Version 11.9 and higher	Version 12.4 and higher	Version 13.3 and higher	Version 14.3 and higher
Europe (London)	Version 10.14 and higher	Version 11.9 and higher	Version 12.4 and higher	Version 13.3 and higher	Version 14.3 and higher
Europe (Milan)	-	-	-	-	-
Europe (Paris)	Version 10.14 and higher	Version 11.9 and higher	Version 12.4 and higher	Version 13.3 and higher	Version 14.3 and higher
Europe (Stockholm)	Version 10.14 and higher	Version 11.9 and higher	Version 12.4 and higher	Version 13.3 and higher	Version 14.3 and higher

Region	Aurora PostgreSQL 10	Aurora PostgreSQL 11	Aurora PostgreSQL 12	Aurora PostgreSQL 13	Aurora PostgreSQL 14
Middle East (Bahrain)	-	-	-	-	-
South America (São Paulo)	Version 10.14 and higher	Version 11.9 and higher	Version 12.4 and higher	Version 13.3 and higher	Version 14.3 and higher
AWS GovCloud (US-East)	Version 10.14 and higher	Version 11.9 and higher	Version 12.4 and higher	Version 13.3 and higher	Version 14.3 and higher
AWS GovCloud (US-West)	Version 10.14 and higher	Version 11.9 and higher	Version 12.4 and higher	Version 13.3 and higher	Version 14.3 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 \(p. 320\)](#).

### Topics

- [Aurora machine learning with Aurora MySQL \(p. 24\)](#)
- [Aurora machine learning with Aurora PostgreSQL \(p. 25\)](#)

## Aurora machine learning with Aurora MySQL

Aurora machine learning is supported for SageMaker and Amazon Comprehend in the AWS Regions where they are available. For a list of AWS Regions where Amazon SageMaker is available, see [Amazon SageMaker endpoints and quotas](#) in the *Amazon Web Services General Reference*. For a list of AWS Regions where Amazon Comprehend is available, see [Amazon Comprehend endpoints and quotas](#) in the *Amazon Web Services General Reference*. Following are the supported engines and Region availability for Aurora machine learning with Aurora MySQL.

Region	Aurora MySQL 5.6	Aurora MySQL 5.7	Aurora MySQL 8.0
US East (Ohio)	-	Version 2.07 and higher	Version 3.01.0 and higher
US East (N. Virginia)	-	Version 2.07 and higher	Version 3.01.0 and higher
US West (N. California)	-	Version 2.07 and higher	Version 3.01.0 and higher
US West (Oregon)	-	Version 2.07 and higher	Version 3.01.0 and higher
Africa (Cape Town)	-	-	-
Asia Pacific (Hong Kong)	-	Version 2.07 and higher	Version 3.01.0 and higher
Asia Pacific (Jakarta)	-	Version 2.07 and higher	Version 3.01.0 and higher
Asia Pacific (Mumbai)	-	Version 2.07 and higher	Version 3.01.0 and higher
Asia Pacific (Osaka)	-	Version 2.07.3 and higher	Version 3.01.0 and higher

Region	Aurora MySQL 5.6	Aurora MySQL 5.7	Aurora MySQL 8.0
Asia Pacific (Seoul)	-	Version 2.07 and higher	Version 3.01.0 and higher
Asia Pacific (Singapore)	-	Version 2.07 and higher	Version 3.01.0 and higher
Asia Pacific (Sydney)	-	Version 2.07 and higher	Version 3.01.0 and higher
Asia Pacific (Tokyo)	-	Version 2.07 and higher	Version 3.01.0 and higher
Canada (Central)	-	Version 2.07 and higher	Version 3.01.0 and higher
China (Beijing)	-	Version 2.07 and higher	Version 3.01.0 and higher
China (Ningxia)	-	Version 2.07 and higher	Version 3.01.0 and higher
Europe (Frankfurt)	-	Version 2.07 and higher	Version 3.01.0 and higher
Europe (Ireland)	-	Version 2.07 and higher	Version 3.01.0 and higher
Europe (London)	-	Version 2.07 and higher	Version 3.01.0 and higher
Europe (Milan)	-	-	-
Europe (Paris)	-	Version 2.07 and higher	Version 3.01.0 and higher
Europe (Stockholm)	-	Version 2.07 and higher	Version 3.01.0 and higher
Middle East (Bahrain)	-	Version 2.07 and higher	Version 3.01.0 and higher
South America (São Paulo)	-	Version 2.07 and higher	Version 3.01.0 and higher
AWS GovCloud (US-East)	-	Version 2.07 and higher	Version 3.01.0 and higher
AWS GovCloud (US-West)	-	Version 2.07 and higher	Version 3.01.0 and higher

## Aurora machine learning with Aurora PostgreSQL

Aurora machine learning is supported for SageMaker and Amazon Comprehend in the AWS Regions where they are available. For a list of AWS Regions where Amazon SageMaker is available, see [Amazon SageMaker endpoints and quotas](#) in the *Amazon Web Services General Reference*. For a list of AWS Regions where Amazon Comprehend is available, see [Amazon Comprehend endpoints and quotas](#) in the *Amazon Web Services General Reference*. Following are the supported engines and Region availability for Aurora machine learning with Aurora PostgreSQL.

Region	Aurora PostgreSQL 10	Aurora PostgreSQL 11	Aurora PostgreSQL 12	Aurora PostgreSQL 13	Aurora PostgreSQL 14
US East (Ohio)	Version 10.14	Version 11.9 and higher	Version 12.4 and higher	Version 13.3 and higher	Version 14.3 and higher
US East (N. Virginia)	Version 10.14	Version 11.9 and higher	Version 12.4 and higher	Version 13.3 and higher	Version 14.3 and higher
US West (N. California)	Version 10.14	Version 11.9 and higher	Version 12.4 and higher	Version 13.3 and higher	Version 14.3 and higher
US West (Oregon)	Version 10.14	Version 11.9 and higher	Version 12.4 and higher	Version 13.3 and higher	Version 14.3 and higher

Region	Aurora PostgreSQL 10	Aurora PostgreSQL 11	Aurora PostgreSQL 12	Aurora PostgreSQL 13	Aurora PostgreSQL 14
Africa (Cape Town)	-	-	-	-	-
Asia Pacific (Hong Kong)	Version 10.14	Version 11.9 and higher	Version 12.4 and higher	Version 13.3 and higher	Version 14.3 and higher
Asia Pacific (Jakarta)	Version 10.14	Version 11.9 and higher	Version 12.4 and higher	Version 13.3 and higher	Version 14.3 and higher
Asia Pacific (Mumbai)	Version 10.14	Version 11.9 and higher	Version 12.4 and higher	Version 13.3 and higher	Version 14.3 and higher
Asia Pacific (Osaka)	Version 10.14	Version 11.9 and higher	Version 12.4 and higher	Version 13.3 and higher	Version 14.3 and higher
Asia Pacific (Seoul)	Version 10.14	Version 11.9 and higher	Version 12.4 and higher	Version 13.3 and higher	Version 14.3 and higher
Asia Pacific (Singapore)	Version 10.14	Version 11.9 and higher	Version 12.4 and higher	Version 13.3 and higher	Version 14.3 and higher
Asia Pacific (Sydney)	Version 10.14	Version 11.9 and higher	Version 12.4 and higher	Version 13.3 and higher	Version 14.3 and higher
Asia Pacific (Tokyo)	Version 10.14	Version 11.9 and higher	Version 12.4 and higher	Version 13.3 and higher	Version 14.3 and higher
Canada (Central)	Version 10.14	Version 11.9 and higher	Version 12.4 and higher	Version 13.3 and higher	Version 14.3 and higher
China (Beijing)	Version 10.14	Version 11.9 and higher	Version 12.4 and higher	Version 13.3 and higher	Version 14.3 and higher
China (Ningxia)	Version 10.14	Version 11.9 and higher	Version 12.4 and higher	Version 13.3 and higher	Version 14.3 and higher
Europe (Frankfurt)	Version 10.14	Version 11.9 and higher	Version 12.4 and higher	Version 13.3 and higher	Version 14.3 and higher
Europe (Ireland)	Version 10.14	Version 11.9 and higher	Version 12.4 and higher	Version 13.3 and higher	Version 14.3 and higher
Europe (London)	Version 10.14	Version 11.9 and higher	Version 12.4 and higher	Version 13.3 and higher	Version 14.3 and higher
Europe (Milan)	-	-	-	-	-
Europe (Paris)	Version 10.14	Version 11.9 and higher	Version 12.4 and higher	Version 13.3 and higher	Version 14.3 and higher
Europe (Stockholm)	Version 10.14	Version 11.9 and higher	Version 12.4 and higher	Version 13.3 and higher	Version 14.3 and higher
Middle East (Bahrain)	Version 10.14	Version 11.9 and higher	Version 12.4 and higher	Version 13.3 and higher	Version 14.3 and higher
South America (São Paulo)	Version 10.14	Version 11.9 and higher	Version 12.4 and higher	Version 13.3 and higher	Version 14.3 and higher

Region	Aurora PostgreSQL 10	Aurora PostgreSQL 11	Aurora PostgreSQL 12	Aurora PostgreSQL 13	Aurora PostgreSQL 14
AWS GovCloud (US-East)	Version 10.14	Version 11.9 and higher	Version 12.4 and higher	Version 13.3 and higher	Version 14.3 and higher
AWS GovCloud (US-West)	Version 10.14	Version 11.9 and higher	Version 12.4 and higher	Version 13.3 and higher	Version 14.3 and higher

## 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 \(p. 790\)](#). 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 \(p. 794\)](#).

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
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

Region	Aurora MySQL 5.6	Aurora MySQL 5.7	Aurora MySQL 8.0
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 about RDS Proxy, see [Using Amazon RDS Proxy \(p. 1430\)](#).

### Topics

- [Amazon RDS Proxy with Aurora MySQL \(p. 28\)](#)
- [Amazon RDS Proxy with Aurora PostgreSQL \(p. 29\)](#)

## Amazon RDS Proxy with Aurora MySQL

Following are the supported engines and Region availability for RDS Proxy with Aurora MySQL.

Region	Aurora MySQL 5.6	Aurora MySQL 5.7	Aurora MySQL 8.0
US East (Ohio)	Version 5.6.10a; version 1.22 and higher	Version 2.07 and higher	Version 3.01.0 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
US West (N. California)	Version 5.6.10a; version 1.22 and higher	Version 2.07 and higher	Version 3.01.0 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
Africa (Cape Town)	Version 5.6.10a; version 1.22 and higher	Version 2.07 and higher	Version 3.01.0 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
Asia Pacific (Jakarta)	-	-	-

Region	Aurora MySQL 5.6	Aurora MySQL 5.7	Aurora MySQL 8.0
Asia Pacific (Mumbai)	Version 5.6.10a; version 1.22 and higher	Version 2.07 and higher	Version 3.01.0 and higher
Asia Pacific (Osaka)	Version 1.22 and higher	Version 2.07 and higher	Version 3.01.0 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
Asia Pacific (Singapore)	Version 5.6.10a; version 1.22 and higher	Version 2.07 and higher	Version 3.01.0 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
Asia Pacific (Tokyo)	Version 5.6.10a; version 1.22 and higher	Version 2.07 and higher	Version 3.01.0 and higher
Canada (Central)	Version 5.6.10a; version 1.22 and higher	Version 2.07 and higher	Version 3.01.0 and higher
China (Beijing)	-	-	-
China (Ningxia)	-	-	-
Europe (Frankfurt)	Version 5.6.10a; version 1.22 and higher	Version 2.07 and higher	Version 3.01.0 and higher
Europe (Ireland)	Version 5.6.10a; version 1.22 and higher	Version 2.07 and higher	Version 3.01.0 and higher
Europe (London)	Version 5.6.10a; version 1.22 and higher	Version 2.07 and higher	Version 3.01.0 and higher
Europe (Milan)	Version 5.6.10a; version 1.22 and higher	Version 2.07 and higher	Version 3.01.0 and higher
Europe (Paris)	Version 5.6.10a; version 1.22 and higher	Version 2.07 and higher	Version 3.01.0 and higher
Europe (Stockholm)	Version 5.6.10a; version 1.22 and higher	Version 2.07 and higher	Version 3.01.0 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
South America (São Paulo)	Version 5.6.10a; version 1.22 and higher	Version 2.07 and higher	Version 3.01.0 and higher
AWS GovCloud (US-East)	-	-	-
AWS GovCloud (US-West)	-	-	-

## Amazon RDS Proxy with Aurora PostgreSQL

Following are the supported engines and Region availability for RDS Proxy with Aurora PostgreSQL.

<b>Region</b>	<b>Aurora PostgreSQL 10</b>	<b>Aurora PostgreSQL 11</b>	<b>Aurora PostgreSQL 12</b>	<b>Aurora PostgreSQL 13</b>
US East (Ohio)	Version 10.14 and higher	Version 11.9 and higher	Version 12.4 and higher	Version 13.3 and higher
US East (N. Virginia)	Version 10.14 and higher	Version 11.9 and higher	Version 12.4 and higher	Version 13.3 and higher
US West (N. California)	Version 10.14 and higher	Version 11.9 and higher	Version 12.4 and higher	Version 13.3 and higher
US West (Oregon)	Version 10.14 and higher	Version 11.9 and higher	Version 12.4 and higher	Version 13.3 and higher
Africa (Cape Town)	Version 10.14 and higher	Version 11.9 and higher	Version 12.4 and higher	Version 13.3 and higher
Asia Pacific (Hong Kong)	Version 10.14 and higher	Version 11.9 and higher	Version 12.4 and higher	Version 13.3 and higher
Asia Pacific (Jakarta)	-	-	-	-
Asia Pacific (Mumbai)	Version 10.14 and higher	Version 11.9 and higher	Version 12.4 and higher	Version 13.3 and higher
Asia Pacific (Osaka)	Version 10.14 and higher	Version 11.9 and higher	Version 12.4 and higher	Version 13.3 and higher
Asia Pacific (Seoul)	Version 10.14 and higher	Version 11.9 and higher	Version 12.4 and higher	Version 13.3 and higher
Asia Pacific (Singapore)	Version 10.14 and higher	Version 11.9 and higher	Version 12.4 and higher	Version 13.3 and higher
Asia Pacific (Sydney)	Version 10.14 and higher	Version 11.9 and higher	Version 12.4 and higher	Version 13.3 and higher
Asia Pacific (Tokyo)	Version 10.14 and higher	Version 11.9 and higher	Version 12.4 and higher	Version 13.3 and higher
Canada (Central)	Version 10.14 and higher	Version 11.9 and higher	Version 12.4 and higher	Version 13.3 and higher
China (Beijing)	-	-	-	-
China (Ningxia)	-	-	-	-
Europe (Frankfurt)	Version 10.14 and higher	Version 11.9 and higher	Version 12.4 and higher	Version 13.3 and higher
Europe (Ireland)	Version 10.14 and higher	Version 11.9 and higher	Version 12.4 and higher	Version 13.3 and higher
Europe (London)	Version 10.14 and higher	Version 11.9 and higher	Version 12.4 and higher	Version 13.3 and higher
Europe (Milan)	Version 10.14 and higher	Version 11.9 and higher	Version 12.4 and higher	Version 13.3 and higher

Region	Aurora PostgreSQL 10	Aurora PostgreSQL 11	Aurora PostgreSQL 12	Aurora PostgreSQL 13
Europe (Paris)	Version 10.14 and higher	Version 11.9 and higher	Version 12.4 and higher	Version 13.3 and higher
Europe (Stockholm)	Version 10.14 and higher	Version 11.9 and higher	Version 12.4 and higher	Version 13.3 and higher
Middle East (Bahrain)	Version 10.14 and higher	Version 11.9 and higher	Version 12.4 and higher	Version 13.3 and higher
South America (São Paulo)	Version 10.14 and higher	Version 11.9 and higher	Version 12.4 and higher	Version 13.3 and higher
AWS GovCloud (US-East)	-	-	-	-
AWS GovCloud (US-West)	-	-	-	-

## Aurora Serverless v2

Aurora Serverless v2 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. The scaling is faster and more granular than with Aurora Serverless v1. With Aurora Serverless v2, each cluster can contain a writer DB instance and multiple reader DB instances. You can combine Aurora Serverless v2 and traditional provisioned DB instances within the same cluster. For more information, see [Using Aurora Serverless v2 \(p. 1482\)](#).

Region	Aurora MySQL version 3	Aurora PostgreSQL 13	Aurora PostgreSQL 14
US East (Ohio)	Version 3.02.0	Version 13.6 and higher	Version 14.3 and higher
US East (N. Virginia)	Version 3.02.0	Version 13.6 and higher	Version 14.3 and higher
US West (N. California)	Version 3.02.0	Version 13.6 and higher	Version 14.3 and higher
US West (Oregon)	Version 3.02.0	Version 13.6 and higher	Version 14.3 and higher
Africa (Cape Town)	-	-	-
Asia Pacific (Hong Kong)	Version 3.02.0	Version 13.6 and higher	Version 14.3 and higher
Asia Pacific (Mumbai)	Version 3.02.0	Version 13.6 and higher	Version 14.3 and higher
Asia Pacific (Mumbai)	Version 3.02.0	Version 13.6 and higher	Version 14.3 and higher
Asia Pacific (Osaka)	-	-	-
Asia Pacific (Seoul)	Version 3.02.0	Version 13.6 and higher	Version 14.3 and higher
Asia Pacific (Singapore)	Version 3.02.0	Version 13.6 and higher	Version 14.3 and higher
Asia Pacific (Sydney)	Version 3.02.0	Version 13.6 and higher	Version 14.3 and higher
Asia Pacific (Tokyo)	Version 3.02.0	Version 13.6 and higher	Version 14.3 and higher

Region	Aurora MySQL version 3	Aurora PostgreSQL 13	Aurora PostgreSQL 14
Canada (Central)	Version 3.02.0	Version 13.6 and higher	Version 14.3 and higher
China (Beijing)	-	-	-
China (Ningxia)	-	-	-
Europe (Frankfurt)	Version 3.02.0	Version 13.6 and higher	Version 14.3 and higher
Europe (Ireland)	Version 3.02.0	Version 13.6 and higher	Version 14.3 and higher
Europe (London)	Version 3.02.0	Version 13.6 and higher	Version 14.3 and higher
Europe (Milan)	-	-	-
Europe (Paris)	Version 3.02.0	Version 13.6 and higher	Version 14.3 and higher
Europe (Stockholm)	Version 3.02.0	Version 13.6 and higher	Version 14.3 and higher
Middle East (Bahrain)	-	-	-
South America (São Paulo)	Version 3.02.0	Version 13.6 and higher	Version 14.3 and higher
AWS GovCloud (US-East)	-	-	-
AWS GovCloud (US-West)	-	-	-

## Aurora Serverless v1

Aurora Serverless 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, using a single DB instance in each cluster. For more information, see [Using Amazon Aurora Serverless v1 \(p. 1543\)](#).

Region	Aurora MySQL 5.6	Aurora MySQL 5.7	Aurora MySQL 8.0	Aurora PostgreSQL 10	Aurora PostgreSQL 11
US East (Ohio)	Version 1.22.3	Version 2.08.3	-	Version 10.18	Version 11.13
US East (N. Virginia)	Version 1.22.3	Version 2.08.3	-	Version 10.18	Version 11.13
US West (N. California)	Version 1.22.3	Version 2.08.3	-	Version 10.18	Version 11.13
US West (Oregon)	Version 1.22.3	Version 2.08.3	-	Version 10.18	Version 11.13
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
Asia Pacific (Mumbai)	Version 1.22.3	Version 2.08.3	-	Version 10.18	Version 11.13
Asia Pacific (Osaka)	-	-	-	-	-
Asia Pacific (Seoul)	Version 1.22.3	Version 2.08.3	-	Version 10.18	Version 11.13
Asia Pacific (Singapore)	Version 1.22.3	Version 2.08.3	-	Version 10.18	Version 11.13
Asia Pacific (Sydney)	Version 1.22.3	Version 2.08.3	-	Version 10.18	Version 11.13
Asia Pacific (Tokyo)	Version 1.22.3	Version 2.08.3	-	Version 10.18	Version 11.13
Canada (Central)	Version 1.22.3	Version 2.08.3	-	Version 10.18	Version 11.13
China (Beijing)	-	-	-	-	-
China (Ningxia)	-	-	-	-	-
Europe (Frankfurt)	Version 1.22.3	Version 2.08.3	-	Version 10.18	Version 11.13
Europe (Ireland)	Version 1.22.3	Version 2.08.3	-	Version 10.18	Version 11.13
Europe (London)	Version 1.22.3	Version 2.08.3	-	Version 10.18	Version 11.13
Europe (Milan)	-	-	-	-	-
Europe (Paris)	Version 1.22.3	Version 2.08.3	-	Version 10.18	Version 11.13
Europe (Stockholm)	-	-	-	-	-
Middle East (Bahrain)	-	-	-	-	-
South America (São Paulo)	-	-	-	-	-
AWS GovCloud (US-East)	-	-	-	-	-
AWS GovCloud (US-West)	-	-	-	-	-

## Data API for Aurora Serverless v1

The Data API for Aurora Serverless provides a web-services interface to an Aurora Serverless v1 cluster. Instead of 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 v1 \(p. 1581\)](#).

<b>Region</b>	<b>Aurora MySQL 5.6</b>	<b>Aurora MySQL 5.7</b>	<b>Aurora MySQL 8.0</b>	<b>Aurora PostgreSQL 10</b>	<b>Aurora PostgreSQL 11</b>
US East (Ohio)	Version 1.22.3	Version 2.08.3	-	Version 10.18	Version 11.13
US East (N. Virginia)	Version 1.22.3	Version 2.08.3	-	Version 10.18	Version 11.13
US West (N. California)	Version 1.22.3	Version 2.08.3	-	Version 10.18	Version 11.13
US West (Oregon)	Version 1.22.3	Version 2.08.3	-	Version 10.18	Version 11.13
Africa (Cape Town)	-	-	-	-	-
Asia Pacific (Hong Kong)	-	-	-	-	-
Asia Pacific (Jakarta)	-	-	-	-	-
Asia Pacific (Mumbai)	Version 1.22.3	Version 2.08.3	-	Version 10.18	Version 11.13
Asia Pacific (Osaka)	-	-	-	-	-
Asia Pacific (Seoul)	Version 1.22.3	Version 2.08.3	-	Version 10.18	Version 11.13
Asia Pacific (Singapore)	Version 1.22.3	Version 2.08.3	-	Version 10.18	Version 11.13
Asia Pacific (Sydney)	Version 1.22.3	Version 2.08.3	-	Version 10.18	Version 11.13
Asia Pacific (Tokyo)	Version 1.22.3	Version 2.08.3	-	Version 10.18	Version 11.13
Canada (Central)	Version 1.22.3	Version 2.08.3	-	Version 10.18	Version 11.13
China (Beijing)	-	-	-	-	-
China (Ningxia)	-	-	-	-	-
Europe (Frankfurt)	Version 1.22.3	Version 2.08.3	-	Version 10.18	Version 11.13
Europe (Ireland)	Version 1.22.3	Version 2.08.3	-	Version 10.18	Version 11.13
Europe (London)	Version 1.22.3	Version 2.08.3	-	Version 10.18	Version 11.13
Europe (Milan)	-	-	-	-	-
Europe (Paris)	Version 1.22.3	Version 2.08.3	-	Version 10.18	Version 11.13
Europe (Stockholm)	-	-	-	-	-

Region	Aurora MySQL 5.6	Aurora MySQL 5.7	Aurora MySQL 8.0	Aurora PostgreSQL 10	Aurora PostgreSQL 11
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 \(p. 36\)](#)
- [Viewing the endpoints for an Aurora cluster \(p. 37\)](#)
- [Using the cluster endpoint \(p. 37\)](#)
- [Using the reader endpoint \(p. 38\)](#)
- [Using custom endpoints \(p. 38\)](#)
- [Creating a custom endpoint \(p. 40\)](#)
- [Viewing custom endpoints \(p. 42\)](#)
- [Editing a custom endpoint \(p. 47\)](#)
- [Deleting a custom endpoint \(p. 49\)](#)
- [End-to-end AWS CLI example for custom endpoints \(p. 50\)](#)
- [Using the instance endpoints \(p. 54\)](#)
- [How Aurora endpoints work with high availability \(p. 54\)](#)

## 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 \(p. 1201\)](#). 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,CustomEndpoints: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 \(p. 1459\)](#).

## 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 \(p. 39\)](#)
- [Membership rules for custom endpoints \(p. 39\)](#)

- [Managing custom endpoints \(p. 40\)](#)

## 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 the check box clear, the custom endpoint uses a static list containing only the DB instances specified on the page. 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 not selected on the page. 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](#)

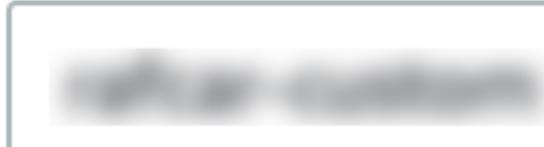
## Creating a custom endpoint

### 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

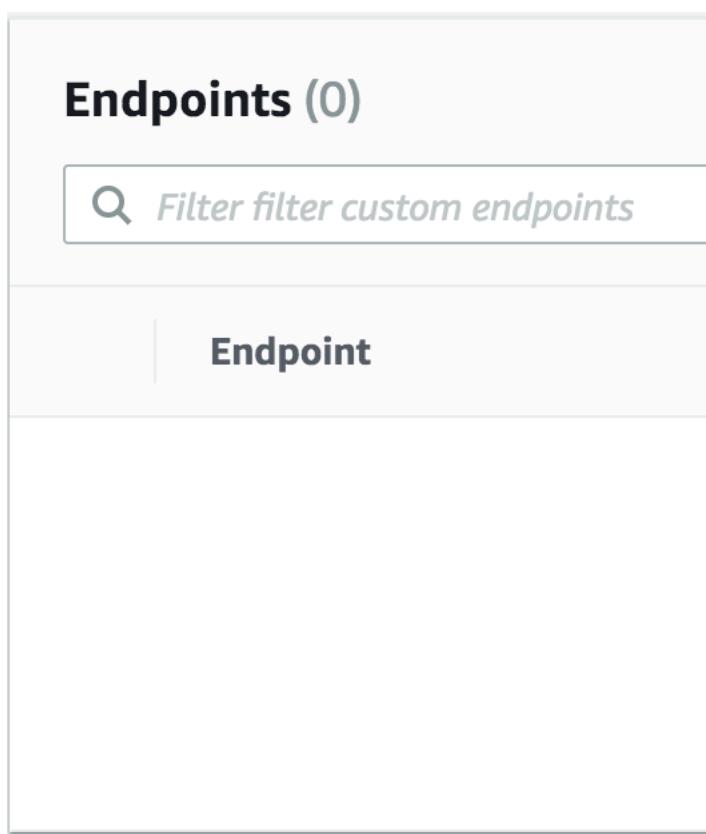
To create a custom endpoint with the RDS API, run the [CreateDBClusterEndpoint](#) operation.

# 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



45

Filter endpoint members

To see the additional detail of whether new DB instances added to the cluster are automatically added to the endpoint also, open the [Edit](#) page 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 \(p. 39\)](#).

You can continue connecting to and using a custom endpoint while the changes from an edit action are in progress.

## 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
```

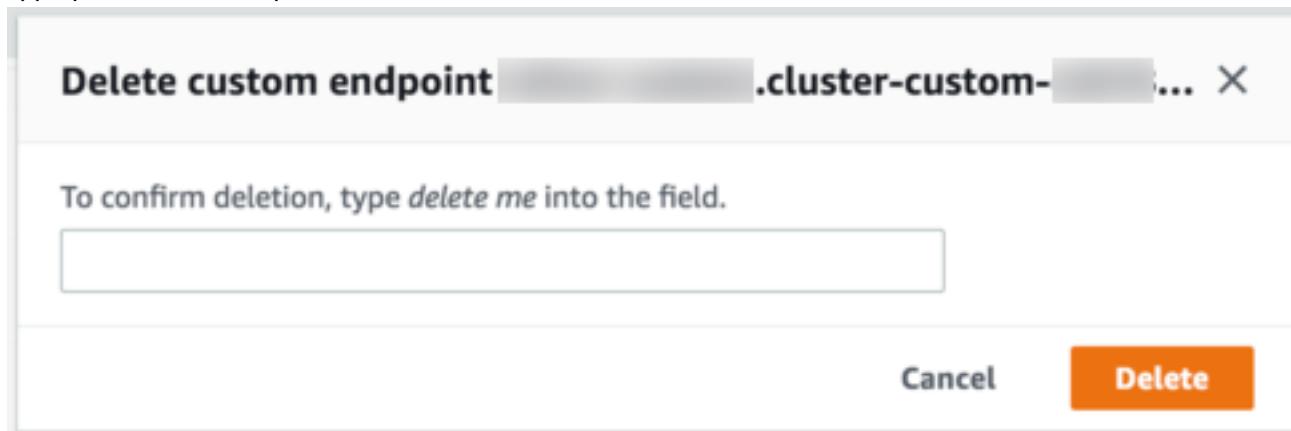
## RDS API

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. Use the [describe-db-clusters](#) AWS CLI command or [DescribeDBClusters](#) RDS API operation to find the DB cluster and reader endpoints, DB instances, whether DB instances are readers, and their promotion tiers. 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 \(p. 72\)](#).

# Aurora DB instance classes

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

A DB instance class consists of both the DB instance type and the size. For example, db.m6g is a general-purpose DB instance type powered by AWS Graviton2 processors, while db.m6g.2xlarge is a DB instance class within the db.m6g instance type.

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

## Topics

- [DB instance class types \(p. 56\)](#)
- [Supported DB engines for DB instance classes \(p. 57\)](#)
- [Determining DB instance class support in AWS Regions \(p. 61\)](#)
- [Hardware specifications for DB instance classes for Aurora \(p. 64\)](#)

## DB instance class types

Amazon Aurora supports three types of instance classes: Aurora Serverless v2, memory optimized, and burstable performance. For more information about Amazon EC2 instance types, see [Instance types](#) in the Amazon EC2 documentation.

The following is the Aurora Serverless v2 type available:

- **db.serverless** – A special DB instance class used by Aurora Serverless v2. Aurora adjusts the compute, memory, and network resources dynamically as the workload changes. For usage details, see [Using Aurora Serverless v2 \(p. 1482\)](#).

The following are the memory optimized DB instance types 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.

You can modify a DB instance to use one of the DB instance classes powered by AWS Graviton2 processors by completing the same steps as any other DB instance modification.

- **db.r6i** – Instance classes that are ideal for running memory-intensive workloads.
- **db.r5** – 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 types available:

- **db.t4g** – Burstable instance classes powered by Arm-based AWS Graviton2 processors. These deliver better price performance than previous 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** – 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 \(p. 64\)](#).

## Supported DB engines for DB 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
<b>db.serverless – Aurora Serverless v2 instance class with automatic capacity scaling</b>		
db.serverless	See <a href="#">Requirements for Aurora Serverless v2 (p. 1490)</a>	See <a href="#">Requirements for Aurora Serverless v2 (p. 1490)</a>
<b>db.x2g – memory-optimized instance classes powered by AWS Graviton2 processors</b>		
db.x2g.16xlarge	2.09.2 and higher, 2.10.0 and higher, 3.01.0 and higher	14.3, 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	14.3, 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	14.3, 13.3, 12.4 and higher, 11.9 and higher
db.x2g.4xlarge	2.09.2 and higher, 2.10.0 and higher, 3.01.0 and higher	14.3, 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	14.3, 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	14.3, 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	14.3, 13.3, 12.4 and higher, 11.9 and higher
<b>db.r6g – memory-optimized instance classes powered by AWS Graviton2 processors</b>		

<b>Instance class</b>	<b>Aurora MySQL</b>	<b>Aurora PostgreSQL</b>
db.r6g.16xlarge	2.09.2 and higher, 2.10.0 and higher, 3.01.0 and higher	14.3, 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	14.3, 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	14.3, 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	14.3, 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	14.3, 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	14.3, 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	14.3, 13.3, 12.4 and higher, 11.9 and higher
<b>db.r6i – memory-optimized instance classes</b>		
db.r6i.32xlarge	No	14.3 and higher, 13.5 and higher, 12.9 and higher
db.r6i.24xlarge	No	14.3 and higher, 13.5 and higher, 12.9 and higher
db.r6i.16xlarge	No	14.3 and higher, 13.5 and higher, 12.9 and higher
db.r6i.12xlarge	No	14.3 and higher, 13.5 and higher, 12.9 and higher
db.r6i.8xlarge	No	14.3 and higher, 13.5 and higher, 12.9 and higher
db.r6i.4xlarge	No	14.3 and higher, 13.5 and higher, 12.9 and higher
db.r6i.2xlarge	No	14.3 and higher, 13.5 and higher, 12.9 and higher

Instance class	Aurora MySQL	Aurora PostgreSQL
db.r6i.xlarge	No	14.3 and higher, 13.5 and higher, 12.9 and higher
db.r6i.large	No	14.3 and higher, 13.5 and higher, 12.9 and higher
<b>db.r5 – memory-optimized instance classes</b>		
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
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
<b>db.r4 – memory-optimized instance classes</b>		
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

Instance class	Aurora MySQL	Aurora PostgreSQL
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
<b>db.r3 – memory-optimized instance classes</b>		
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
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
<b>db.t4g – burstable-performance instance classes powered by AWS Graviton2 processors</b>		
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	14.3, 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	14.3, 13.3, 12.4 and higher, 11.9 and higher
db.t4g.small	No	No
<b>db.t3 – burstable-performance instance classes</b>		
db.t3.2xlarge	No	No
db.t3.xlarge	No	No
db.t3.large	2.10 and higher, 3.01.0 and higher	14.3, 13.3, 11.6 and higher, 10.11 and higher

Instance class	Aurora MySQL	Aurora PostgreSQL
db.t3.medium	1.14.4 and higher, 3.01.0 and higher	14.3, 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
<b>db.t2 – burstable-performance instance classes</b>		
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 \(p. 61\)](#)
- [Using the AWS CLI to determine DB instance class support in AWS Regions \(p. 62\)](#)
  - [Listing the DB instance classes that are supported by a specific DB engine version in an AWS Region \(p. 62\)](#)
  - [Listing the DB engine versions that support a specific DB instance class in an AWS Region \(p. 63\)](#)

## 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	<a href="#">Database engine updates for Amazon Aurora MySQL version 1</a> in the <a href="#">Release Notes for Aurora MySQL</a>
MySQL 5.7-compatible and 8.0-compatible Aurora	aurora-mysql	<a href="#">Database engine updates for Amazon Aurora MySQL version 2</a> and <a href="#">Database engine updates for Amazon Aurora MySQL version 3</a> in the <a href="#">Release Notes for Aurora MySQL</a>
Aurora PostgreSQL	aurora-postgresql	<a href="#">Release Notes for Aurora PostgreSQL</a>

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 \(p. 62\)](#)
- [Listing the DB engine versions that support a specific DB instance class in an AWS Region \(p. 63\)](#)

### 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 13.6 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  
13.6 ^  
--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  
13.6 ^  
--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[].\n{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[].\n{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 \(p. 57\)](#).

Instance class	vCPU	ECU	Memory (GiB)	Max. bandwidth (mbps) of local storage	Network performance
<b>db.x2g – memory-optimized instance classes</b>					
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
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
<b>db.r6g – memory-optimized instance classes powered by AWS Graviton2 processors</b>					
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
<b>db.r6i – memory-optimized instance classes</b>					
db.r6i.32xlarge	128	—	1,024	40,000	50 Gbps
db.r6i.24xlarge	96	—	768	30,000	37.5 Gbps
db.r6i.16xlarge	64	—	512	20,000	25 Gbps
db.r6i.12xlarge	48	—	384	15,000	18.75 Gbps
db.r6i.8xlarge	32	—	256	10,000	12.5 Gbps
db.r6i.4xlarge*	16	—	128	Up to 10,000	Up to 12.5 Gbps
db.r6i.2xlarge*	8	—	64	Up to 10,000	Up to 12.5 Gbps
db.r6i.xlarge*	4	—	32	Up to 10,000	Up to 12.5 Gbps
db.r6i.large*	2	—	16	Up to 10,000	Up to 12.5 Gbps
<b>db.r5 – memory-optimized instance classes</b>					
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

Instance class	vCPU	ECU	Memory (GiB)	Max. bandwidth (mbps) of local storage	Network performance
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
<b>db.r4 – memory-optimized instance classes</b>					
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
<b>db.r3 – memory-optimized instance classes</b>					
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
db.r3.xlarge	4	13	30.5	500	Moderate
db.r3.large	2	6.5	15.25	—	Moderate
<b>db.t4g – burstable-performance instance classes</b>					
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
<b>db.t3 – burstable-performance instance classes</b>					
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
<b>db.t2 – burstable-performance instance classes</b>					
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 \(p. 67\)](#)
- [What the cluster volume contains \(p. 67\)](#)
- [How Aurora storage automatically resizes \(p. 67\)](#)
- [How Aurora data storage is billed \(p. 68\)](#)
- [Amazon Aurora reliability \(p. 68\)](#)

## 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.

Aurora uses separate local storage for nonpersistent, temporary files. This includes files that are used for such purposes as sorting large data sets during query processing, and building indexes. For more information, see [Temporary storage limits for Aurora MySQL \(p. 724\)](#) and [Temporary storage limits for Aurora PostgreSQL \(p. 1146\)](#).

## 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 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 \(p. 1759\)](#). 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 \(p. 746\)](#) or [Displaying volume status for an Aurora PostgreSQL DB cluster \(p. 1151\)](#). For ways to balance storage costs against other priorities, [Storage scaling \(p. 274\)](#) 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 \(p. 274\)](#). 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 \(p. 274\)](#). 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 \(p. 68\)](#)
- [Survivable page cache \(p. 69\)](#)
- [Crash recovery \(p. 69\)](#)

## 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 page cache

In Aurora, each DB instance's page cache is managed in a separate process from the database, which allows the page cache to survive independently of the database. (The page cache is also called the *InnoDB buffer pool* on Aurora MySQL and the *buffer cache* on Aurora PostgreSQL.)

In the unlikely event of a database failure, the page cache remains in memory, which keeps current data pages "warm" in the page cache when the database restarts. This provides a performance gain by bypassing the need for the initial queries to execute read I/O operations to "warm up" the page cache.

For Aurora MySQL, page cache behavior when rebooting and failing over is the following:

- Versions earlier than 2.10 – When the writer DB instance reboots, the page cache on the writer instance survives, but reader DB instances lose their page caches.
- Version 2.10 and higher – You can reboot the writer instance without rebooting the reader instances.
  - If the reader instances don't reboot when the writer instance reboots, they don't lose their page caches.
  - If the reader instances reboot when the writer instance reboots, they do lose their page caches.
- When a reader instance reboots, the page caches on the writer and reader instances both survive.
- When the DB cluster fails over, the effect is similar to when a writer instance reboots. On the new writer instance (previously the reader instance) the page cache survives, but on the reader instance (previously the writer instance), the page cache doesn't survive.

For Aurora PostgreSQL, you can use cluster cache management to preserve the page cache of a designated reader instance that becomes the writer instance after failover. For more information, see [Fast recovery after failover with cluster cache management for Aurora PostgreSQL \(p. 1208\)](#).

## 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 \(p. 72\)](#).

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 \(p. 73\)](#). 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 for Amazon Aurora \(p. 1653\)](#).

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 VPC VPCs and Amazon Aurora \(p. 1729\)](#).
- 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 \(p. 681\)](#) or [Security with Amazon Aurora PostgreSQL \(p. 1018\)](#).

- You can use IAM database authentication.

With IAM database authentication, you authenticate to your Aurora 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 \(p. 1683\)](#).

- You can use Kerberos authentication for Aurora PostgreSQL.

You can use Kerberos to authenticate users when they connect to your Aurora PostgreSQL DB cluster. In this case, your DB cluster 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. 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. For more information, see [Using Kerberos authentication with Aurora PostgreSQL \(p. 1035\)](#)

For information about configuring security, see [Security in Amazon Aurora \(p. 1634\)](#).

## 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 \(p. 681\)](#), [Security with Amazon Aurora PostgreSQL \(p. 1018\)](#), or [Using TLS/SSL with Aurora Serverless v1 \(p. 1546\)](#).

# 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 \(p. 71\)](#)
- [High availability for Aurora DB instances \(p. 71\)](#)
- [High availability across AWS Regions with Aurora global databases \(p. 72\)](#)
- [Fault tolerance for an Aurora DB cluster \(p. 72\)](#)

## 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 \(p. 243\)](#).

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 \(p. 35\)](#).

**Tip**

Within each AWS Region, Availability Zones (AZs) 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 \(p. 151\)](#).

## Fault tolerance for an Aurora DB cluster

An Aurora DB cluster is fault tolerant by design. The cluster volume spans multiple Availability Zones (AZs) 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 your provisioned or Aurora Serverless v2 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 or Aurora Serverless v2 cluster only contains a single DB instance, or if the primary instance and all reader instances are in the same AZ, make sure to manually create one or more new DB instances in another AZ.
- If your cluster uses Aurora Serverless v1, 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 \(binary log replication\) \(p. 841\)](#).

## 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 \(p. 73\)](#)
- [Replication with Aurora MySQL \(p. 74\)](#)
- [Replication with Aurora PostgreSQL \(p. 75\)](#)

## 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 \(p. 67\)](#). 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 \(p. 329\)](#). 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 \(p. 72\)](#).

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 \(p. 270\)](#).

## 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 \(p. 72\)](#)
  - 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 \(p. 826\)](#).

## 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 \(p. 151\)](#).
- 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 \(p. 1224\)](#).

## DB instance billing for Aurora

Amazon RDS provisioned instances in an Amazon 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 \(p. 56\)](#).
- 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 \(p. 67\)](#).
- 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 \(p. 368\)](#).

- 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.
- **Aurora Serverless v2** – Aurora Serverless v2 provides on-demand capacity where the billing unit is Aurora capacity unit (ACU) hours instead of DB instance hours. Aurora Serverless v2 capacity increases and decreases, within a range that you specify, depending on the load on your database. You can configure a cluster where all the capacity is Aurora Serverless v2, or a combination of Aurora Serverless v2 and on-demand or reserved provisioned instances. For information about how Aurora Serverless v2 ACUs work, see [How Aurora Serverless v2 works \(p. 1484\)](#).

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

#### Topics

- [On-Demand DB instances for Aurora \(p. 77\)](#)
- [Reserved DB instances for Aurora \(p. 78\)](#)

## 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 \(p. 441\)](#).

## 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 \(p. 79\)](#).

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.r5.large. For more information about instance classes, see [Aurora DB instance classes \(p. 56\)](#).

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.r5.large can apply to a db.r5.xlarge, but not to a db.r6g.large, because db.r5 and db.r6g 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).

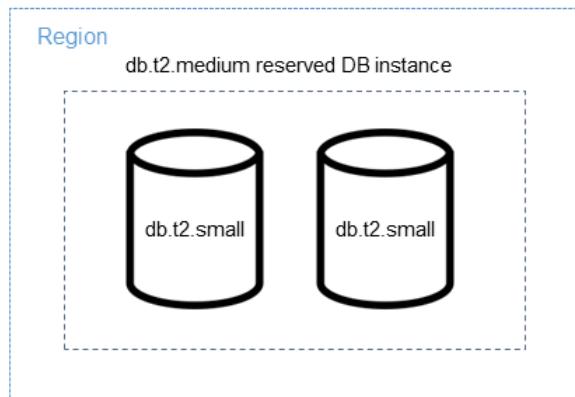
Size-flexible reserved DB instances are available for the following Aurora database engines:

- Aurora MySQL
- Aurora PostgreSQL

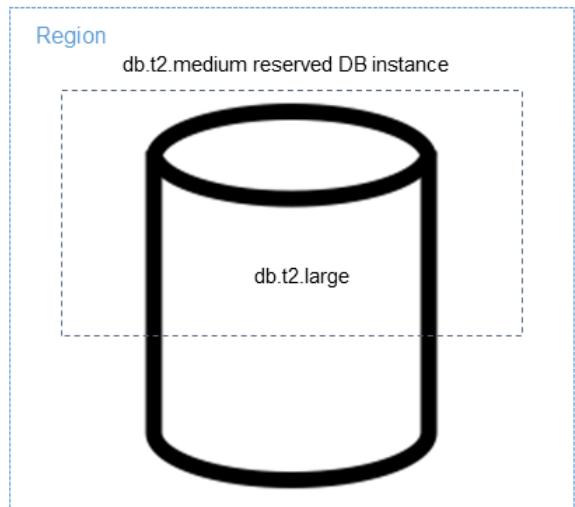
You can compare usage for different reserved DB instance sizes by using normalized units. For example, one unit of usage on two db.r3.large DB instances is equivalent to eight normalized units of usage on one db.r3.small. The following table shows the number of normalized units for each DB instance size.

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 (set to 'No'), 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), a note about additional taxes, and a note about hourly rates for Reserved Instances. At the bottom are 'Cancel' and 'Continue' buttons.

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 a highlighted orange button 'Order'.

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.r3.large	y	1y	1820.00
USD	0.368 USD	mysql	Partial Upfront		
OFFERING	649fd0c8-cf6d-47a0-bfa6-060f8e75e95f	db.r3.small	n	1y	227.50
USD	0.046 USD	mysql	Partial Upfront		
OFFERING	123456cd-ab1c-47a0-bfa6-12345667232f	db.r3.small	n	1y	162.00
USD	0.00 USD	mysql	All Upfront		
Recurring Charges:	Amount	Currency	Frequency		
Recurring Charges:	0.123	USD	Hourly		
OFFERING	123456cd-ab1c-37a0-bfa6-12345667232d	db.r3.large	y	1y	700.00
USD	0.00 USD	mysql	All Upfront		
Recurring Charges:	Amount	Currency	Frequency		
Recurring Charges:	1.25	USD	Hourly		
OFFERING	123456cd-ab1c-17d0-bfa6-12345667234e	db.r3.xlarge	n	1y	4242.00
USD	2.42 USD	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.r3.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.r3.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

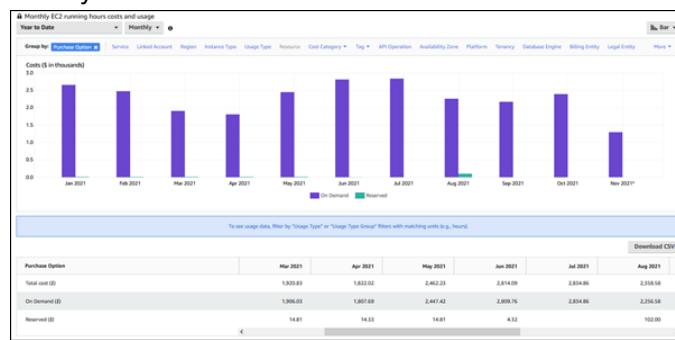
The reserved DB instance in this example was purchased All Upfront, so there are no hourly charges.

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.

## Topics

- [Get an AWS account and your root user credentials \(p. 87\)](#)
- [Create an IAM user \(p. 87\)](#)
- [Sign in as an IAM user \(p. 88\)](#)
- [Create IAM user access keys \(p. 89\)](#)
- [Determine requirements \(p. 89\)](#)
- [Provide access to the DB cluster in the VPC by creating a security group \(p. 90\)](#)

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 \(p. 93\)](#).

## Get an AWS account and your root user credentials

To access AWS, you must sign up for an AWS account.

### To sign up for an 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.

AWS sends you a confirmation email after the sign-up process is complete. At any time, you can view your current account activity and manage your account by going to <https://aws.amazon.com/> and choosing **My Account**.

## Create an IAM user

If your account already includes an IAM user with full AWS administrative permissions, you can skip this section.

When you first create an AWS account, you begin with one 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.

### Important

We strongly recommend that you do not use the root user for your everyday tasks. Safeguard your root user credentials and use them to perform the tasks that only the root user can perform. For the complete list of tasks that require you to sign in as the root user, see [Tasks that require root user credentials](#) in the *AWS General Reference*.

## 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 users**.
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](#).

## Sign in as an IAM user

Sign in to the [IAM console](#) by choosing **IAM user** and entering your AWS account ID or account alias. On the next page, enter your IAM user name and your password.

### Note

For your convenience, the AWS sign-in page uses a browser cookie to remember your IAM user name and account information. If you previously signed in as a different user, choose the sign-in link beneath the button to return to the main sign-in page. From there, you can enter your AWS account ID or account alias to be redirected to the IAM user sign-in page for your account.

# Create IAM user access keys

Access keys consist of an access key ID and secret access key, which are used to sign programmatic requests that you make to AWS. If you don't have access keys, you can create them from the AWS Management Console. As a best practice, do not use the AWS account root user access keys for any task where it's not required. Instead, [create a new administrator IAM user](#) with access keys for yourself.

The only time that you can view or download the secret access key is when you create the keys. You cannot recover them later. However, you can create new access keys at any time. You must also have permissions to perform the required IAM actions. For more information, see [Permissions required to access IAM resources](#) in the *IAM User Guide*.

## To create access keys for an IAM user

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, choose **Users**.
3. Choose the name of the user whose access keys you want to create, and then choose the **Security credentials** tab.
4. In the **Access keys** section, choose **Create access key**.
5. To view the new access key pair, choose **Show**. You will not have access to the secret access key again after this dialog box closes. Your credentials will look something like this:
  - Access key ID: AKIAIOSFODNN7EXAMPLE
  - Secret access key: wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
6. To download the key pair, choose **Download .csv file**. Store the keys in a secure location. You will not have access to the secret access key again after this dialog box closes.

Keep the keys confidential in order to protect your AWS account and never email them. Do not share them outside your organization, even if an inquiry appears to come from AWS or Amazon.com. No one who legitimately represents Amazon will ever ask you for your secret key.

7. After you download the .csv file, choose **Close**. When you create an access key, the key pair is active by default, and you can use the pair right away.

## Related topics

- [What is IAM?](#) in the *IAM User Guide*
- [AWS security credentials](#) in *AWS General 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 \(p. 56\)](#).

- **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 3: Create a VPC security group \(p. 1739\)](#).
    - 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 3: Create a VPC security group \(p. 1739\)](#).
    - 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 VPC VPCs and Amazon Aurora \(p. 1729\)](#).
    - 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 cluster in a VPC \(p. 1729\)](#).
- **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 \(p. 71\)](#).
- **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 for Amazon Aurora \(p. 1653\)](#).
- **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 instance, you can use the VPC security group you set up for the Amazon EC2 instance.

You can configure connectivity between an Amazon EC2 instance and a DB cluster when you create the DB cluster. For more information, see [Configure automatic network connectivity with an EC2 instance \(p. 127\)](#).

**Tip**

You can set up network connectivity between an Amazon EC2 instance and a DB cluster automatically when you create the DB cluster. For more information, see [Configure automatic network connectivity with an EC2 instance \(p. 127\)](#).

For more information about creating a VPC for use with Aurora, see [Tutorial: Create a VPC for use with a DB cluster \(IPv4 only\) \(p. 1744\)](#). For information about common scenarios for accessing a DB instance, see [Scenarios for accessing a DB cluster in a VPC \(p. 1739\)](#).

### 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 \(p. 127\)](#). For information about getting started by creating a DB cluster that uses a specific DB engine, see [Getting started with Amazon Aurora \(p. 93\)](#).

# 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 \(p. 87\)](#).

**Topics**

- [Creating a DB cluster and connecting to a database on an Aurora MySQL DB cluster \(p. 93\)](#)
- [Creating a DB cluster and connecting to a database on an Aurora PostgreSQL DB cluster \(p. 100\)](#)
- [Tutorial: Create a web server and an Amazon Aurora DB cluster \(p. 107\)](#)

## 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 \(p. 87\)](#).

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 \(p. 93\)](#)
- [Connect to an instance in a DB cluster \(p. 97\)](#)
- [Delete the sample DB cluster, DB subnet group, and VPC \(p. 100\)](#)

## 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 [Tutorial: Create](#)

a VPC for use with a DB cluster (IPv4 only) (p. 1744). 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, size, and DB cluster 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**

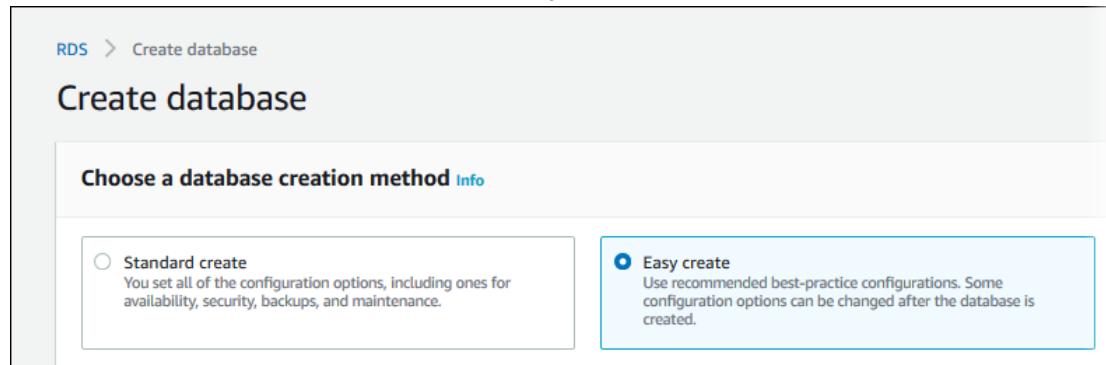
For information about creating DB clusters with **Standard create**, see [Creating an Amazon Aurora DB cluster \(p. 127\)](#).

### 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.

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.

9. For **Master username**, enter a name for the user, or leave the default name (**admin**). This user name and its password give you control over the Aurora MySQL DB cluster.

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

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

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

<b>▼ View default settings for Easy create</b>		
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
Publicly 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 \(p. 87\)](#).
- 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**.

#### 11. 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.

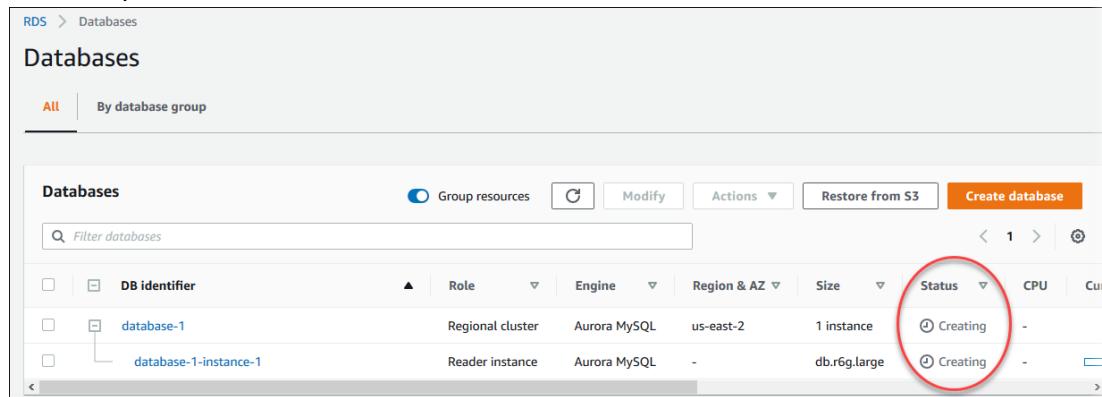
#### **Important**

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 \(p. 248\)](#).

#### 12. 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.



A screenshot of the AWS RDS Databases page. The top navigation bar shows 'RDS > Databases'. Below it, a 'Databases' section has 'All' selected. A table lists two entries:

DB identifier	Role	Engine	Region & AZ	Size	Status	CPU	Cur
database-1	Regional cluster	Aurora MySQL	us-east-2	1 instance	Creating	-	
database-1-instance-1	Reader instance	Aurora MySQL	-	db.r6g.large	Creating	-	

## 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 cluster.

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 cluster is publicly accessible, you can install the SQL client outside of the VPC. If your DB cluster 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 \(p. 87\)](#).

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 \(p. 248\)](#).

If your DB cluster is publicly accessible, make sure its associated security group has inbound rules for the IP addresses that you want to access it. If your DB cluster is private, make sure its associated security group has inbound rules for the security group of each resource that you want to access it, such as the security group of an Amazon EC2 instance.

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 \(p. 207\)](#). If you can't connect to your DB cluster, see [Can't connect to Amazon RDS DB instance \(p. 1761\)](#).

## 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 \(p. 87\)](#).

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 \(p. 101\)](#)
- [Connect to an instance in an Aurora PostgreSQL DB cluster \(p. 104\)](#)
- [Delete the sample DB cluster, DB subnet group, and VPC \(p. 106\)](#)

## 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 [Tutorial: Create a VPC for use with a DB cluster \(IPv4 only\) \(p. 1744\)](#). 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, size, and DB cluster 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.

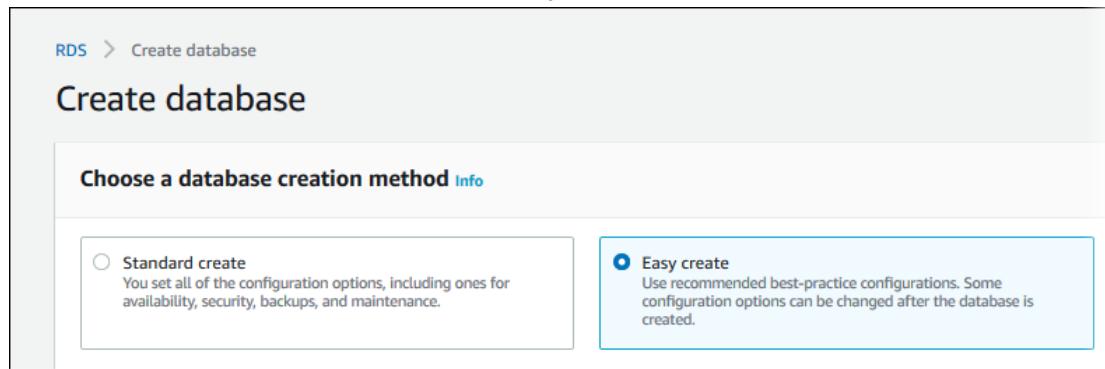
In this example, you use **Easy create** to create an Aurora PostgreSQL DB cluster.

**Note**

For information about creating DB clusters with **Standard create**, see [Creating an Amazon Aurora DB cluster \(p. 127\)](#).

### 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 \(p. 87\)](#).
- 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 \(p. 248\)](#).

## 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						
		Role	Engine	Region & AZ	Size	Status
DB identifier						CPU
database-1		Regional cluster	Aurora PostgreSQL	us-west-1	1 instance	Creating
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 \(p. 87\)](#).

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 \(p. 248\)](#).

If your DB cluster is publicly accessible, make sure its associated security group has inbound rules for the IP addresses that you want to access it. If your DB cluster is private, make sure its associated security group has inbound rules for the security group of each resource that you want to access it, such as the security group of an Amazon EC2 instance.

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.

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

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

The following format is used to connect to an Aurora PostgreSQL DB cluster.

```
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 an Aurora PostgreSQL DB cluster 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 \(p. 212\)](#). If you can't connect to your DB cluster, see [Can't connect to Amazon RDS DB instance \(p. 1761\)](#).

## 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.

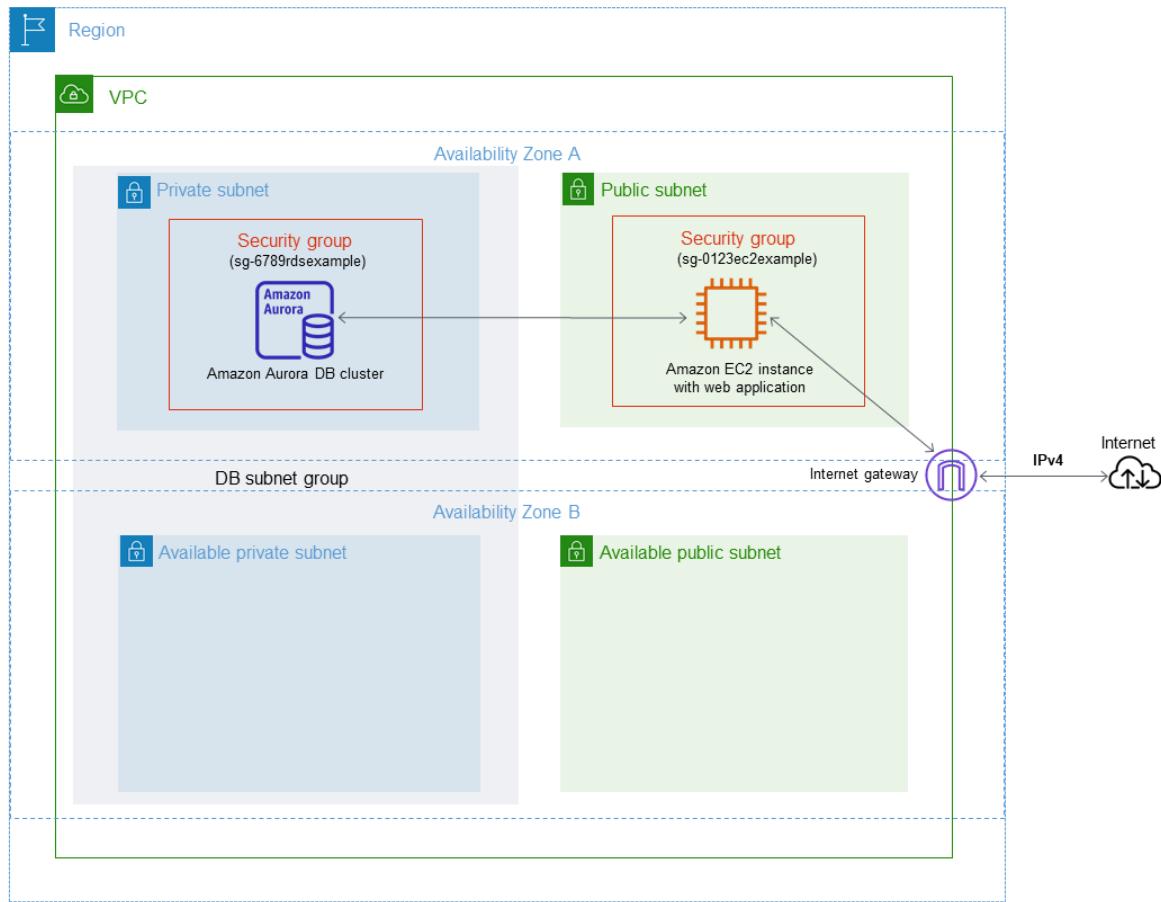
In the tutorial that follows, you specify the VPC, subnets, and security groups when you create the DB cluster. You also specify them when you create the EC2 instance to host your web server. The VPC, subnets, and security groups are required for the DB cluster and the web server to communicate. After 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 a VPC for use with a DB cluster \(IPv4 only\) \(p. 1744\)](#).

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 \(p. 1745\)](#)
  - b. [Create a VPC security group for a public web server \(p. 1745\)](#)
  - c. [Create a VPC security group for a private DB cluster \(p. 1746\)](#)
  - d. [Create a DB subnet group \(p. 1747\)](#)
2. [Create an Amazon Aurora DB cluster \(p. 108\)](#)
  3. [Create an EC2 instance and install a web server \(p. 113\)](#)

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 a VPC for use with a DB cluster \(IPv4 only\) \(p. 1744\)](#). Complete the steps in [Create a VPC with private and public subnets \(p. 1745\)](#), [Create a VPC security group for a public web server \(p. 1745\)](#), and [Create a VPC security group for a private DB cluster \(p. 1746\)](#).

### 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.

## Engine options

Engine type [Info](#)

Amazon Aurora



MySQL



MariaDB



PostgreSQL



Oracle



Microsoft SQL Server



Edition

Amazon Aurora MySQL-Compatible Edition

Amazon Aurora PostgreSQL-Compatible Edition

6. In the **Templates** section, choose **Dev/Test**.
7. In the **Availability and durability** section, use the default values.
8. 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](#)

\*\*\*\*\*

9. In the **Instance configuration** section, set these values:

- **Burstable classes (includes t classes)**
- **db.t3.micro**

## Instance configuration

The DB instance configuration options below are limited to those supported by the engine that you selected above.

### DB instance class [Info](#)

- Standard classes (includes m classes)
- Memory optimized classes (includes r and x classes)
- Burstable classes (includes t classes)**

db.t3.micro

2 vCPUs 1 GiB RAM Network: 2,085 Mbps

- Include previous generation classes**

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 \(p. 1745\)](#).

**Note**

The VPC must have subnets in different Availability Zones.

- **DB subnet group** – Choose a DB subnet group for the VPC, such as the `tutorial-db-subnet-group` created in [Create a DB subnet group \(p. 1747\)](#).
- **Public access** – Choose **No**.
- **VPC security group (firewall)** – Select **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 cluster \(p. 1746\)](#).

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

- **Availability Zone** – Choose **us-west-2a**.

To avoid cross-AZ traffic, make sure the DB cluster and the EC2 instance are in the same Availability Zone.

- 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 cluster.

tutorial-vpc (vpc-04badc20a546242e6)

Only VPCs with a corresponding DB subnet group are listed.

**After a database is created, you can't change its VPC.**

**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](#)**

us-west-2a

**Additional configuration**

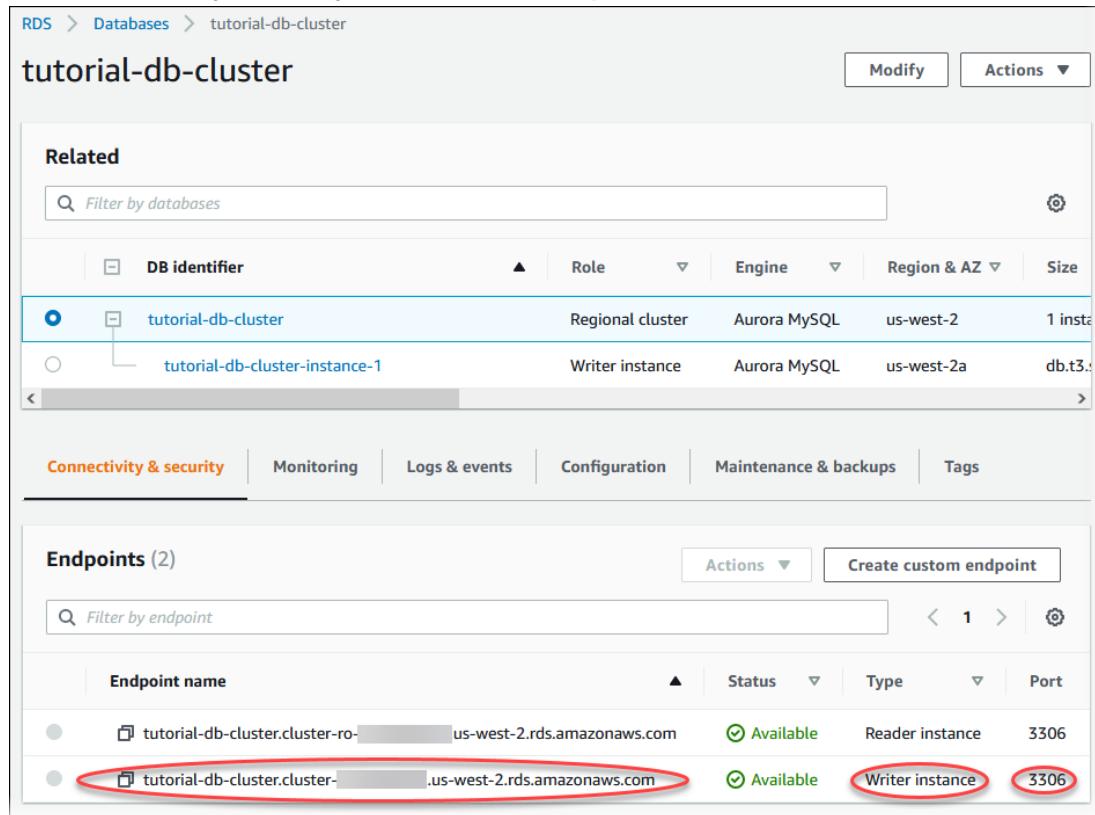
**Database port [Info](#)**  
TCP/IP port that the database will use for application connections.

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
- Related:** Shows the DB identifier for the cluster and its instances.
- Endpoints (2):** Shows two endpoints for the cluster:
  - Reader instance: tutorial-db-cluster.cluster-ro-... (Status: Available, Port: 3306)
  - Writer instance: tutorial-db-cluster.cluster-... (Status: Available, Type: Writer instance, Port: 3306)

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 \(p. 113\)](#).

## 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 \(p. 108\)](#).

### 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.

## Resources

You are using the following Amazon EC2 resources in the US West (Oregon) Region:

Instances (running)	3	Dedicated Hosts	0
Instances	3	Key pairs	5
Placement groups	0	Security groups	10
Volumes	3		

**i** Easily size, configure, and deploy Microsoft SQL Server Always On availability groups on AWS using [Learn more](#)

### Launch instance

To get started, launch an Amazon EC2 instance, which is a virtual server in the cloud.

**Launch instance ▾** [Migrate a server ↗](#)

Note: Your instances will launch in the US West (Oregon) Region

**Service health**

Region  
US West (Oregon)

**Zones**

3. Make sure you have opted into the new launch experience.
4. Under **Name and tags**, for **Name**, enter **tutorial-web-server**.
5. Under **Application and OS Images (Amazon Machine Image)**, choose **Amazon Linux**, and then choose the **Amazon Linux 2 AMI**. Keep the defaults for the other choices.

▼ Application and OS Images (Amazon Machine Image) [Info](#)

An AMI is a template that contains the software configuration (operating system, application server, and applications) required to launch your instance. Search or Browse for AMIs if you don't see what you are looking for below

Search our full catalog including 1000s of application and OS images

Recents    Quick Start

Amazon Linux    Ubuntu    Windows    Red Hat    SUSE Linux

aws    ubuntu®    Microsoft    Red Hat    SUSE

Amazon Machine Image (AMI)

Amazon Linux 2 AMI (HVM) - Kernel 5.10, SSD Volume Type    Free tier eligible

ami-098e42ae54c764c35 (64-bit (x86)) / ami-08e93a9522bbe6df6 (64-bit (Arm))  
Virtualization: hvm    ENA enabled: true    Root device type: ebs

Description  
Amazon Linux 2 Kernel 5.10 AMI 2.0.20220606.1 x86\_64 HVM gp2

Architecture    AMI ID

64-bit (x86)    ami-098e42ae54c764c35

Browse more AMIs  
Including AMIs from AWS, Marketplace and the Community

6. Under **Instance type**, choose **t2.micro**.
  7. Under **Key pair (login)**, choose a **Key pair name** to use an existing key pair. To create a new key pair for the Amazon EC2 instance, choose **Create new key pair** and then use the **Create key pair** window to create it.
- For more information about creating a new key pair, see [Create a key pair](#) in the *Amazon EC2 User Guide for Linux Instances*.
8. For **Network settings**, choose **Edit**.
  9. Under **Network settings**, set these values and keep the other values as their defaults:
    - **VPC (required)** – 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 \(p. 1745\)](#).
    - **Subnet** – Choose an existing public subnet, such as `subnet-identifier tutorial-subnet-public1-us-west-2a` created in [Create a VPC security group for a public web server \(p. 1745\)](#).
    - **Auto-assign public IP** – Choose **Enable**.
    - **Firewall (security groups)** – Choose **Select an existing security group**.
    - **Common security groups** – Choose choose an existing security group, such as the `tutorial-securitygroup` created in [Create a VPC security group for a public web server \(p. 1745\)](#). Make sure that the security group that you choose includes inbound rules for Secure Shell (SSH) and HTTP access.

- **Advanced network configuration** – Leave the default values.

▼ **Network settings**

**VPC - required** [Info](#)

vpc-04badc20a546242e6 (tutorial-vpc)  
10.0.0.0/16

**Subnet Info**

subnet-08f73f5494bd3dbc0 tutorial-subnet-public1-us-west-2a  
VPC: vpc-04badc20a546242e6 Owner: [REDACTED]  
Availability Zone: us-west-2a IP addresses available: 4091

**Create new subnet** [\[+\] Create new subnet](#)

**Auto-assign public IP** [Info](#)

Enable

**Firewall (security groups)** [Info](#)

A security group is a set of firewall rules that control the traffic for your instance. Add rules to allow specific traffic to reach your instance.

Create security group  Select existing security group

**Common security groups** [Info](#)

Select security groups

tutorial-securitygroup sg-097660d74243a426b X  
VPC: vpc-04badc20a546242e6

**Compare security group rules** [\[+\] Compare security group rules](#)

Security groups that you add or remove here will be added to or removed from all your network interfaces.

► **Advanced network configuration**

10. Leave the default values for the remaining sections.
11. Review a summary of your instance configuration in the **Summary** panel, and when you're ready, choose **Launch instance**.
12. On the **Launch Status** page, shown following, note the identifier for your new EC2 instance, for example: i-0288d65fd4470b6a9.

EC2 > Instances > Launch an instance

**Success**  
Successfully initiated launch of instance (i-03a6ad47e97ba9dc5)

▶ Launch log

**Next Steps**

**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 instance**  
Your instance is launching and it might be a few minutes until it is in the running state, when it will be ready for you to use  
Click View Instances to monitor your instance's status. Once your instance is in the 'running' state, you can connect to it from the Instances screen. Find out how to connect to your instance

[View more resources to get you started](#)

[View all instances](#)

13. Choose **View all instances** to find your instance.
14. Wait until **Instance State** for your instance is **running** before continuing.

## Install an Apache web server with PHP and MariaDB

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 php8.0 mariadb10.5
```

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 a VPC for use with a DB cluster \(IPv4 only\) \(p. 1744\)](#). 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 cluster

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 client on the Amazon EC2 instance, connect to the DB cluster, and query the table.

For information about connecting to a DB cluster, see [Connecting to an Amazon Aurora DB cluster \(p. 207\)](#).

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 \(p. 345\)](#). 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 \(p. 123\)](#)
- [Tutorials in other AWS guides \(p. 123\)](#)
- [Tutorials and sample code in GitHub \(p. 124\)](#)

## Tutorials in this guide

The following tutorials in this guide show you how to perform common tasks with Amazon Aurora:

- [Tutorial: Create a VPC for use with a DB cluster \(IPv4 only\) \(p. 1744\)](#)  
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 VPC for use with a DB cluster \(dual-stack mode\) \(p. 1749\)](#)  
Learn how to include a DB cluster in an Amazon virtual private cloud (VPC) that shares data with an Amazon EC2 instance in the same VPC. In this tutorial, you create the VPC for this scenario that works with a database running in dual-stack mode.
- [Tutorial: Create a web server and an Amazon Aurora DB cluster \(p. 107\)](#)  
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: Restore an Amazon Aurora DB cluster from a DB cluster snapshot \(p. 419\)](#)  
Learn how to restore a DB cluster from a DB cluster snapshot.
- [Tutorial: Use tags to specify which Aurora DB clusters to stop \(p. 357\)](#)  
Learn how to use tags to specify which Aurora DB clusters to stop.
- [Tutorial: Log DB instance state changes using Amazon EventBridge \(p. 587\)](#)  
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 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 \(p. 127\)](#)
- [Creating Amazon Aurora resources with AWS CloudFormation \(p. 150\)](#)
- [Using Amazon Aurora global databases \(p. 151\)](#)
- [Connecting to an Amazon Aurora DB cluster \(p. 207\)](#)
- [Working with parameter groups \(p. 215\)](#)
- [Migrating data to an Amazon Aurora DB cluster \(p. 242\)](#)

# 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 \(p. 127\)](#).

For simple instructions on connecting to your Aurora DB cluster, see [Connecting to an Amazon Aurora DB cluster \(p. 207\)](#).

## Topics

- [DB cluster prerequisites \(p. 127\)](#)
- [Creating a DB cluster \(p. 131\)](#)
- [Settings for Aurora DB clusters \(p. 137\)](#)
- [Settings that don't apply to Amazon Aurora for DB clusters \(p. 147\)](#)
- [Settings that don't apply to Amazon Aurora DB instances \(p. 147\)](#)

## 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 \(p. 87\)](#).

The following are prerequisites to complete before creating a DB cluster.

## Topics

- [Configure the network for the DB cluster \(p. 127\)](#)
- [Additional prerequisites \(p. 130\)](#)

## Configure the network for the DB cluster

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 plan to set up connectivity between your new DB cluster and an EC2 instance in the same VPC, you can do so during DB cluster creation. If you plan to connect to your DB cluster from resources other than EC2 instances in the same VPC, you can configure the network connections manually.

## Topics

- [Configure automatic network connectivity with an EC2 instance \(p. 127\)](#)
- [Configure the network manually \(p. 130\)](#)

## Configure automatic network connectivity with an EC2 instance

When you create an Aurora DB cluster, you can use the AWS Management Console to set up connectivity between an Amazon EC2 instance and the new DB cluster. When you do so, RDS configures your VPC and

network settings automatically. The DB cluster is created in the same VPC as the EC2 instance so that the EC2 instance can access the DB cluster.

The following are requirements for connecting an EC2 instance with the DB cluster:

- The EC2 instance must exist in the AWS Region before you create the DB cluster.

If no EC2 instances exist in the AWS Region, the console provides a link to create one.

- Currently, the DB cluster can't be an Aurora Serverless DB cluster or part of an Aurora global database.
- The user who is creating the DB instance must have permissions to perform the following operations:
  - `ec2:AssociateRouteTable`
  - `ec2:AuthorizeSecurityGroupEgress`
  - `ec2>CreateRouteTable`
  - `ec2>CreateSubnet`
  - `ec2>CreateSecurityGroup`
  - `ec2:DescribeInstances`
  - `ec2:DescribeNetworkInterfaces`
  - `ec2:DescribeRouteTables`
  - `ec2:DescribeSecurityGroups`
  - `ec2:DescribeSubnets`
  - `ec2:ModifyNetworkInterfaceAttribute`
  - `ec2:RevokeSecurityGroupEgress`

Using this option creates a private DB cluster. The DB cluster uses a DB subnet group with only private subnets to restrict access to resources within the VPC.

To connect an EC2 instance to the DB cluster, choose **Connect to an EC2 compute resource** in the **Connectivity** section on the **Create database** page.

**Connectivity** Info C

**Compute resource**  
Choose whether to set up a connection to a compute resource for this database. Setting up a connection will automatically change connectivity settings so that the compute resource can connect to this database.

**Don't connect to an EC2 compute resource**  
Don't set up a connection to a compute resource for this database. You can manually set up a connection to a compute resource later.

**Connect to an EC2 compute resource**  
Set up a connection to an EC2 compute resource for this database.

**EC2 Instance** Info  
Choose the EC2 instance to add as the compute resource for this database. A VPC security group is added to this EC2 instance. A VPC security group is also added to the database with an inbound rule that allows the EC2 instance to access the database.

**Choose EC2 instances** ▼

When you choose **Connect to an EC2 compute resource**, RDS sets the following options automatically. You can't change these settings unless you choose not to set up connectivity with an EC2 instance by choosing **Don't connect to an EC2 compute resource**.

Console option	Automatic setting
<b>Network type</b>	RDS sets network type to <b>IPv4</b> . Currently, dual-stack mode isn't supported when you set up a connection between an EC2 instance and the DB cluster.
<b>Virtual Private Cloud (VPC)</b>	RDS sets the VPC to the one associated with the EC2 instance.
<b>DB subnet group</b>	<p>A DB subnet group with a private subnet in each Availability Zone in the AWS Region is required. If a DB subnet group that meets this requirement exists, RDS uses the existing DB subnet group.</p> <p>If a DB subnet group that meets this requirement doesn't exist, RDS uses an available private subnet in each Availability Zone to create a DB subnet group using the private subnets. If a private subnet isn't available in an Availability Zone, RDS creates a private subnet in the Availability Zone and then creates the DB subnet group.</p> <p>When a private subnet is available, RDS uses the route table associated with it and adds any subnets it creates to this route table. When no private subnet is available, RDS creates a route table with no internet gateway access and adds the subnets it creates to the route table.</p>
<b>Public access</b>	<p>RDS chooses <b>No</b> so that the DB cluster isn't publicly accessible.</p> <p>For security, it is a best practice to keep the database private and make sure it isn't accessible from the internet.</p>
<b>VPC security group (firewall)</b>	<p>RDS creates a new security group that is associated with the DB cluster. The security group is named <code>rds-ec2-n</code>, where <code>n</code> is a number. This security group includes an inbound rule with the EC2 VPC security group (firewall) as the source. This security group that is associated with the DB cluster allows the EC2 instance to access the DB cluster.</p> <p>RDS also creates a new security group that is associated with the EC2 instance. The security group is named <code>ec2-rds-n</code>, where <code>n</code> is a number. This security group includes an outbound rule with the VPC security group of the DB cluster as the source. This security group allows the DB cluster to send traffic to the EC2 instance.</p> <p>You can add another new security group by choosing <b>Create new</b> and typing the name of the new security group.</p> <p>You can add existing security groups by choosing <b>Choose existing</b> and selecting security groups to add.</p>
<b>Availability Zone</b>	<p>When you don't create an Aurora Replica in <b>Availability &amp; durability</b> during DB cluster creation (Single-AZ deployment), RDS chooses the Availability Zone of the EC2 instance.</p> <p>When you create an Aurora Replica during DB cluster creation (Multi-AZ deployment), RDS chooses the Availability Zone of the EC2 instance for one DB instance in the DB cluster. RDS randomly chooses a different Availability Zone for the other DB instance in the DB cluster. Either the primary DB instance or the Aurora Replica is created in the same Availability Zone as the EC2 instance. There is the possibility of cross Availability Zone costs if the primary DB instance and EC2 instance are in different Availability Zones.</p>

For more information about these settings, see [Settings for Aurora DB clusters \(p. 137\)](#).

If you make any changes to these settings after the DB cluster is created, the changes might affect the connection between the EC2 instance and the DB cluster.

## Configure the network manually

If you plan to connect to your DB cluster from resources other than EC2 instances in the same VPC, you can configure the network connections manually. If you use the AWS Management Console to create your 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 Availability Zones automatically for you. To choose a specific Availability Zone, you need to change the **Availability & durability** Multi-AZ deployment setting to **Don't create an Aurora Replica**. Doing so exposes an **Availability Zone** setting that lets you choose from among the Availability Zones in your VPC. However, we strongly recommend that you keep the default setting and let Amazon RDS create a Multi-AZ deployment and choose Availability Zones 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.

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 a 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 [Working with a DB cluster in a VPC \(p. 1729\)](#) and [Tutorial: Create a VPC for use with a DB cluster \(IPv4 only\) \(p. 1744\)](#).
- Specify a VPC security group that authorizes connections to your DB cluster. For more information, see [Provide access to the DB cluster in the VPC by creating a security group \(p. 90\)](#) and [Controlling access with security groups \(p. 1721\)](#).
- Specify an RDS DB subnet group that defines at least two subnets in the VPC that can be used by the DB cluster. For more information, see [Working with DB subnet groups \(p. 1730\)](#).

For information on VPCs, see [Amazon VPC VPCs and Amazon Aurora \(p. 1729\)](#). For a tutorial that configures the network for a private DB cluster, see [Tutorial: Create a VPC for use with a DB cluster \(IPv4 only\) \(p. 1744\)](#).

## Additional prerequisites

Before you create your DB cluster, consider the following 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 for Amazon Aurora \(p. 1653\)](#).

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 parameter groups \(p. 215\)](#).
- 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

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.

### 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 \(p. 93\)](#).

### 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**.

## Engine options

Engine type [Info](#)

Amazon Aurora 

MySQL 

MariaDB 

PostgreSQL 

Oracle 

Microsoft SQL Server 

Edition

Amazon Aurora MySQL-Compatible Edition

Amazon Aurora PostgreSQL-Compatible Edition

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 Aurora Serverless v2 \(p. 1482\)](#) and [Using Amazon Aurora Serverless v1 \(p. 1543\)](#).
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. (Optional) Set up a connection to a compute resource for this DB cluster.

You can configure connectivity between an Amazon EC2 instance and the new DB cluster during DB cluster creation. For more information, see [Configure automatic network connectivity with an EC2 instance \(p. 127\)](#).

13. For the remaining sections, specify your DB cluster settings. For information about each setting, see [Settings for Aurora DB clusters \(p. 137\)](#).
14. 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 \(p. 248\)](#).

15. 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' section of the Amazon RDS console. At the top, there are tabs for 'All' and 'By database group'. Below this is a search bar labeled 'Filter databases'. A table lists 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 \(p. 433\)](#).

The screenshot shows the AWS RDS console with the path: RDS > Databases > database-1. The main title is "database-1". On the right, there are "Modify" and "Actions" buttons. Below the title, there's a "Related" section with a search bar and a table for "DB identifier". The table shows "database-1" as a Regional cluster (Aurora MySQL, us-east-2, 1 instance) and "database-1-instance-1" as a Writer instance (Aurora MySQL, us-east-2b, db.r6g.large). Below the table are tabs: Connectivity & security (selected), Monitoring, Logs & events, Configuration, Maintenance & backups, and Tags.

**Endpoints (2)**

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

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.

## 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 \(p. 127\)](#).

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 \(p. 127\)](#).

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 v1 DB cluster. For information about these settings, see [Creating an Aurora Serverless v1 DB cluster \(p. 1559\)](#). Also, some settings aren't available for Aurora Serverless v1 because of Aurora Serverless v1 limitations. For more information, see [Limitations of Aurora Serverless v1 \(p. 1544\)](#).

Console setting	Setting description	CLI option and RDS API parameter
<b>Auto minor version upgrade</b>	<p>Choose <b>Enable auto minor version upgrade</b> 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 <b>Auto minor version upgrade</b> 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 <a href="#">Amazon Aurora PostgreSQL updates (p. 1413)</a>.</p> <p>For more information about engine updates for Aurora MySQL, see <a href="#">Database engine updates for Amazon Aurora MySQL (p. 990)</a>.</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>
<b>AWS KMS key</b>	Only available if <b>Encryption</b> is set to <b>Enable encryption</b> . Choose the AWS KMS key to use for encrypting this DB cluster. For more information, see <a href="#">Encrypting Amazon Aurora resources (p. 1638)</a> .	<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>
<b>Backtrack</b>	Applies only to Aurora MySQL. Choose <b>Enable Backtrack</b> to enable backtracking or <b>Disable Backtrack</b> to disable backtracking. Using 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 <a href="#">Backtracking an Aurora DB cluster (p. 725)</a> .	<p>Using the AWS CLI, run <code>create-db-cluster</code> and set the <code>--backtrack-window</code> option.</p> <p>Using the RDS API, call <code>CreateDBCluster</code> and set the <code>BacktrackWindow</code> parameter.</p>
<b>Copy tags to snapshots</b>	Choose this option to copy any DB instance tags to a DB snapshot when you create a snapshot.	<p>Using the AWS CLI, run <code>create-db-cluster</code> and set the <code>--copy-tags-to-snapshot   --no-copy-tags-to-snapshot</code> option.</p>

Console setting	Setting description	CLI option and RDS API parameter
		Using the RDS API, call <a href="#">CreateDBCluster</a> and set the <code>CopyTagsToSnapshot</code> parameter.
<b>Database authentication</b>	<p>The database authentication you want to use.</p> <p>For MySQL:</p> <ul style="list-style-type: none"> <li>Choose <b>Password authentication</b> to authenticate database users with database passwords only.</li> <li>Choose <b>Password and IAM database authentication</b> to authenticate database users with database passwords and user credentials through IAM users and roles. For more information, see <a href="#">IAM database authentication (p. 1683)</a>.</li> </ul> <p>For PostgreSQL:</p> <ul style="list-style-type: none"> <li>Choose <b>IAM database authentication</b> to authenticate database users with database passwords and user credentials through IAM users and roles. For more information, see <a href="#">IAM database authentication (p. 1683)</a>.</li> <li>Choose <b>Kerberos authentication</b> to authenticate database passwords and user credentials using Kerberos authentication. For more information, see <a href="#">Using Kerberos authentication with Aurora PostgreSQL (p. 1035)</a>.</li> </ul>	<p>To use IAM database authentication with the AWS CLI, run <code>create-db-cluster</code> and set the <code>--enable-iam-database-authentication</code>   <code>--no-enable-iam-database-authentication</code> option.</p> <p>To use IAM database authentication with the RDS API, call <a href="#">CreateDBCluster</a> and set the <code>EnableIAMDatabaseAuthentication</code> parameter.</p> <p>To use Kerberos authentication with the AWS CLI, run <code>create-db-cluster</code> and set the <code>--domain</code> and <code>--domain-iam-role-name</code> options.</p> <p>To use Kerberos authentication with the RDS API, call <a href="#">CreateDBCluster</a> and set the <code>Domain</code> and <code>DomainIAMRoleName</code> parameters.</p>
<b>Database port</b>	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>Using the AWS CLI, run <code>create-db-cluster</code> and set the <code>--port</code> option.</p> <p>Using the RDS API, call <a href="#">CreateDBCluster</a> and set the <code>Port</code> parameter.</p>

Console setting	Setting description	CLI option and RDS API parameter
<b>DB cluster identifier</b>	<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 <a href="#">Amazon Aurora connection management (p. 35)</a>.</p> <p>The DB cluster identifier has the following constraints:</p> <ul style="list-style-type: none"> <li>• It must contain from 1 to 63 alphanumeric characters or hyphens.</li> <li>• Its first character must be a letter.</li> <li>• It cannot end with a hyphen or contain two consecutive hyphens.</li> <li>• It must be unique for all DB clusters per AWS account, per AWS Region.</li> </ul>	<p>Using the AWS CLI, run <a href="#">create-db-cluster</a> and set the --db-cluster-identifier option.</p> <p>Using the RDS API, call <a href="#">CreateDBCluster</a> and set the DBClusterIdentifier parameter.</p>
<b>DB cluster parameter group</b>	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 <a href="#">Working with parameter groups (p. 215)</a> .	<p>Using the AWS CLI, run <a href="#">create-db-cluster</a> and set the --db-cluster-parameter-group-name option.</p> <p>Using the RDS API, call <a href="#">CreateDBCluster</a> and set the DBClusterParameterGroupName parameter.</p>
<b>DB instance class</b>	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 <a href="#">Aurora DB instance classes (p. 56)</a> .	<p>Set this value for every DB instance in your Aurora cluster.</p> <p>Using the AWS CLI, run <a href="#">create-db-instance</a> and set the --db-instance-class option.</p> <p>Using the RDS API, call <a href="#">CreateDBInstance</a> and set the DBInstanceClass parameter.</p>
<b>DB parameter group</b>	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 <a href="#">Working with parameter groups (p. 215)</a> .	<p>Set this value for every DB instance in your Aurora cluster.</p> <p>Using the AWS CLI, run <a href="#">create-db-instance</a> and set the --db-parameter-group-name option.</p> <p>Using the RDS API, call <a href="#">CreateDBInstance</a> and set the DBParameterGroupName parameter.</p>

Console setting	Setting description	CLI option and RDS API parameter
<b>DB subnet group</b>	Choose the DB subnet group to use for the DB cluster. For more information, see <a href="#">DB cluster prerequisites (p. 127)</a> .	Using the AWS CLI, run <code>create-db-cluster</code> and set the <code>--db-subnet-group-name</code> option.  Using the RDS API, call <code>CreateDBCluster</code> and set the <code>DBSubnetGroupName</code> parameter.
<b>Enable deletion protection</b>	Choose <b>Enable deletion protection</b> to prevent your DB cluster from being deleted. If you create a production DB cluster with the console, deletion protection is enabled by default.	Using the AWS CLI, run <code>create-db-cluster</code> and set the <code>--deletion-protection   --no-deletion-protection</code> option.  Using the RDS API, call <code>CreateDBCluster</code> and set the <code>DeletionProtection</code> parameter.
<b>Enable encryption</b>	Choose <b>Enable encryption</b> to enable encryption at rest for this DB cluster. For more information, see <a href="#">Encrypting Amazon Aurora resources (p. 1638)</a> .	Using the AWS CLI, run <code>create-db-cluster</code> and set the <code>--storage-encrypted   --no-storage-encrypted</code> option.  Using the RDS API, call <code>CreateDBCluster</code> and set the <code>StorageEncrypted</code> parameter.
<b>Enable Enhanced Monitoring</b>	Choose <b>Enable enhanced monitoring</b> to enable gathering metrics in real time for the operating system that your DB cluster runs on. For more information, see <a href="#">Monitoring OS metrics with Enhanced Monitoring (p. 518)</a> .	Set these values for every DB instance in your Aurora cluster.  Using the AWS CLI, run <code>create-db-instance</code> and set the <code>--monitoring-interval</code> and <code>--monitoring-role-arn</code> options.  Using the RDS API, call <code>CreateDBInstance</code> and set the <code>MonitoringInterval</code> and <code>MonitoringRoleArn</code> parameters.
<b>Engine type</b>	Choose the database engine to be used for this DB cluster.	Using the AWS CLI, run <code>create-db-cluster</code> and set the <code>--engine</code> option.  Using the RDS API, call <code>CreateDBCluster</code> and set the <code>Engine</code> parameter.
<b>Engine version</b>	Applies only to the provisioned capacity type. Choose the version number of your DB engine.	Using the AWS CLI, run <code>create-db-cluster</code> and set the <code>--engine-version</code> option.  Using the RDS API, call <code>CreateDBCluster</code> and set the <code>EngineVersion</code> parameter.

Console setting	Setting description	CLI option and RDS API parameter
<b>Failover priority</b>	<p>Choose a failover priority for the instance. If you don't choose a value, the default is <b>tier-1</b>. This priority determines the order in which Aurora Replicas are promoted when recovering from a primary instance failure. For more information, see <a href="#">Fault tolerance for an Aurora DB cluster (p. 72)</a>.</p>	<p>Set this value for every DB instance in your Aurora cluster.</p> <p>Using the AWS CLI, run <a href="#">create-db-instance</a> and set the --promotion-tier option.</p> <p>Using the RDS API, call <a href="#">CreateDBInstance</a> and set the PromotionTier parameter.</p>
<b>Initial database name</b>	<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"> <li>• It must contain 1–64 alphanumeric characters.</li> <li>• It can't be a word reserved by the database engine.</li> </ul> <p>For Aurora PostgreSQL, the default database name has these constraints:</p> <ul style="list-style-type: none"> <li>• It must contain 1–63 alphanumeric characters.</li> <li>• It must begin with a letter or an underscore. Subsequent characters can be letters, underscores, or digits (0–9).</li> <li>• It can't be a word reserved by the database engine.</li> </ul> <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 <a href="#">Connecting to an Amazon Aurora DB cluster (p. 207)</a>.</p>	<p>Using the AWS CLI, run <a href="#">create-db-cluster</a> and set the --database-name option.</p> <p>Using the RDS API, call <a href="#">CreateDBCluster</a> and set the DatabaseName parameter.</p>

Console setting	Setting description	CLI option and RDS API parameter
<b>Log exports</b>	In the <b>Log exports</b> 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 <a href="#">Publishing Amazon Aurora MySQL logs to Amazon CloudWatch Logs (p. 924)</a> . For more information about publishing Aurora PostgreSQL logs to CloudWatch Logs, see <a href="#">Publishing Aurora PostgreSQL logs to Amazon CloudWatch Logs (p. 1265)</a> .	Using the AWS CLI, run <a href="#">create-db-cluster</a> and set the <code>--enable-cloudwatch-logs-exports</code> option.  Using the RDS API, call <a href="#">CreateDBCluster</a> and set the <code>EnableCloudwatchLogsExports</code> parameter.
<b>Maintenance window</b>	Choose <b>Select window</b> and specify the weekly time range during which system maintenance can occur. Or choose <b>No preference</b> for Amazon RDS to assign a period randomly.	Using the AWS CLI, run <a href="#">create-db-cluster</a> and set the <code>--preferred-maintenance-window</code> option.  Using the RDS API, call <a href="#">CreateDBCluster</a> and set the <code>PreferredMaintenanceWindow</code> parameter.
<b>Master password</b>	Enter a password to log on to your DB cluster: <ul style="list-style-type: none"><li>• For Aurora MySQL, the password must contain 8–41 printable ASCII characters.</li><li>• For Aurora PostgreSQL, it must contain 8–128 printable ASCII characters.</li><li>• It can't contain /, ", @, or a space.</li></ul>	Using the AWS CLI, run <a href="#">create-db-cluster</a> and set the <code>--master-user-password</code> option.  Using the RDS API, call <a href="#">CreateDBCluster</a> and set the <code>MasterUserPassword</code> parameter.
<b>Master username</b>	Enter a name to use as the master user name to log on to your DB cluster: <ul style="list-style-type: none"><li>• For Aurora MySQL, the name must contain 1–16 alphanumeric characters.</li><li>• For Aurora PostgreSQL, it must contain 1–63 alphanumeric characters.</li><li>• The first character must be a letter.</li><li>• The name can't be a word reserved by the database engine.</li></ul> You can't change the master user name after the DB cluster is created.	Using the AWS CLI, run <a href="#">create-db-cluster</a> and set the <code>--master-username</code> option.  Using the RDS API, call <a href="#">CreateDBCluster</a> and set the <code>MasterUsername</code> parameter.

Console setting	Setting description	CLI option and RDS API parameter
<b>Multi-AZ deployment</b>	<p>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 <b>Create Replica in Different Zone</b>, 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 <a href="#">Regions and Availability Zones (p. 11)</a>.</p>	<p>Using the AWS CLI, run <a href="#">create-db-cluster</a> and set the <code>--availability-zones</code> option.</p> <p>Using the RDS API, call <a href="#">CreateDBCluster</a> and set the <code>AvailabilityZones</code> parameter.</p>
<b>Network type</b>	<p>The IP addressing protocols supported by the DB cluster.</p> <p><b>IPv4</b> to specify that resources can communicate with the DB cluster only over the IPv4 addressing protocol.</p> <p><b>Dual-stack mode</b> to specify that resources can communicate with the DB cluster over IPv4, IPv6, or both. Use dual-stack mode if you have any resources that must communicate with your DB cluster over the IPv6 addressing protocol. To use dual-stack mode, make sure at least two subnets spanning two Availability Zones that support both the IPv4 and IPv6 network protocol. Also, make sure you associate an IPv6 CIDR block with subnets in the DB subnet group you specify.</p> <p>For more information, see <a href="#">Amazon Aurora IP addressing (p. 1731)</a>.</p>	<p>Using the AWS CLI, run <a href="#">create-db-cluster</a> and set the <code>-network-type</code> option.</p> <p>Using the RDS API, call <a href="#">CreateDBCluster</a> and set the <code>NetworkType</code> parameter.</p>

Console setting	Setting description	CLI option and RDS API parameter
<b>Public access</b>	<p>Choose <b>Publicly accessible</b> to give the DB cluster a public IP address, or choose <b>Not publicly accessible</b>. 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 <a href="#">Hiding a DB cluster in a VPC from the internet (p. 1735)</a>.</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 <a href="#">Can't connect to Amazon RDS DB instance (p. 1761)</a>.</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 <a href="#">Internetwork traffic privacy (p. 1652)</a>.</p>	<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>--publicly-accessible</code>   <code>--no-publicly-accessible</code> option.</p> <p>Using the RDS API, call <code>CreateDBInstance</code> and set the <code>PubliclyAccessible</code> parameter.</p>
<b>RDS Proxy</b>	<p>Choose <b>Create an RDS Proxy</b> to create a proxy for your DB cluster. Amazon RDS automatically creates an IAM role and a Secrets Manager secret for the proxy.</p> <p>For more information, see <a href="#">Using Amazon RDS Proxy (p. 1430)</a>.</p>	Not available when creating a DB cluster.
<b>Retention period</b>	<p>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.</p>	<p>Using the AWS CLI, run <code>create-db-cluster</code> and set the <code>--backup-retention-period</code> option.</p> <p>Using the RDS API, call <code>CreateDBCluster</code> and set the <code>BackupRetentionPeriod</code> parameter.</p>

Console setting	Setting description	CLI option and RDS API parameter
<b>Turn on Performance Insights</b>	Choose <b>Turn on Performance Insights</b> to turn on Amazon RDS Performance Insights. For more information, see <a href="#">Monitoring DB load with Performance Insights on Amazon Aurora (p. 461)</a> .	<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>
<b>Turn on DevOps Guru</b>	Choose <b>Turn on DevOps Guru</b> to turn on Amazon DevOps Guru for your Aurora database. For DevOps Guru for RDS to provide detailed analysis of performance anomalies, Performance Insights must be turned on. For more information, see <a href="#">Setting up DevOps Guru for RDS (p. 513)</a> .	You can turn on DevOps Guru for RDS from within the RDS console, but not by using the RDS API or CLI. For more information about turning on DevOps Guru, see the <a href="#">Amazon DevOps Guru User Guide</a> .
<b>Virtual Private Cloud (VPC)</b>	Choose the VPC to host the DB cluster. Choose <b>Create a New VPC</b> to have Amazon RDS create a VPC for you. For more information, see <a href="#">DB cluster prerequisites (p. 127)</a> .	For the AWS CLI and API, you specify the VPC security group IDs.
<b>VPC security group (firewall)</b>	<p>Choose <b>Create new</b> to have Amazon RDS create a VPC security group for you. Or choose <b>Choose existing</b> and specify one or more VPC security groups to secure network access to the DB cluster.</p> <p>When you choose <b>Create new</b> 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 <a href="#">DB cluster prerequisites (p. 127)</a>.</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>

## Settings that don't apply to Amazon Aurora for DB clusters

The following settings in the AWS CLI command [create-db-cluster](#) and the RDS API operation [CreateDBCluster](#) don't apply to Amazon Aurora DB clusters.

**Note**

The AWS Management Console doesn't show these settings for Aurora DB clusters.

AWS CLI setting	RDS API setting
--allocated-storage	AllocatedStorage
--auto-minor-version-upgrade   --no-auto-minor-version-upgrade	AutoMinorVersionUpgrade
--db-cluster-instance-class	DBClusterInstanceClass
--enable-performance-insights   --no-enable-performance-insights	EnablePerformanceInsights
--iops	Iops
--monitoring-interval	MonitoringInterval
--monitoring-role-arn	MonitoringRoleArn
--option-group-name	OptionGroupName
--performance-insights-kms-key-id	PerformanceInsightsKMSKeyId
--performance-insights-retention-period	PerformanceInsightsRetentionPeriod
--publicly-accessible   --no-publicly-accessible	PubliclyAccessible
--storage-type	StorageType

## Settings that don't apply to Amazon Aurora DB instances

The following settings in the AWS CLI command [create-db-instance](#) and the RDS API operation [CreateDBInstance](#) don't apply to DB instances Amazon Aurora DB cluster.

**Note**

The AWS Management Console doesn't show these settings for Aurora DB instances.

AWS CLI setting	RDS API setting
--allocated-storage	AllocatedStorage
--availability-zone	AvailabilityZone
--backup-retention-period	BackupRetentionPeriod

AWS CLI setting	RDS API setting
--backup-target	BackupTarget
--character-set-name	CharacterSetName
--character-set-name	CharacterSetName
--custom-iam-instance-profile	CustomIamInstanceProfile
--db-security-groups	DBSecurityGroups
--deletion-protection   --no-deletion-protection	DeletionProtection
--domain	Domain
--domain-iam-role-name	DomainIAMRoleName
--enable-cloudwatch-logs-exports	EnableCloudwatchLogsExports
--enable-customer-owned-ip   --no-enable-customer-owned-ip	EnableCustomerOwnedIp
--enable-iam-database-authentication   --no-enable-iam-database-authentication	EnableIAMDatabaseAuthentication
--engine-version	EngineVersion
--iops	Iops
--kms-key-id	KmsKeyId
--license-model	LicenseModel
--master-username	MasterUsername
--master-user-password	MasterUserPassword
--max-allocated-storage	MaxAllocatedStorage
--multi-az   --no-multi-az	MultiAZ
--nchar-character-set-name	NcharCharacterSetName
--network-type	NetworkType
--option-group-name	OptionGroupName
--preferred-backup-window	PreferredBackupWindow
--processor-features	ProcessorFeatures
--storage-encrypted   --no-storage-encrypted	StorageEncrypted
--storage-type	StorageType
--tde-credential-arn	TdeCredentialArn
--tde-credential-password	TdeCredentialPassword

AWS CLI setting	RDS API setting
--timezone	Timezone
--vpc-security-group-ids	VpcSecurityGroupIds

# 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 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

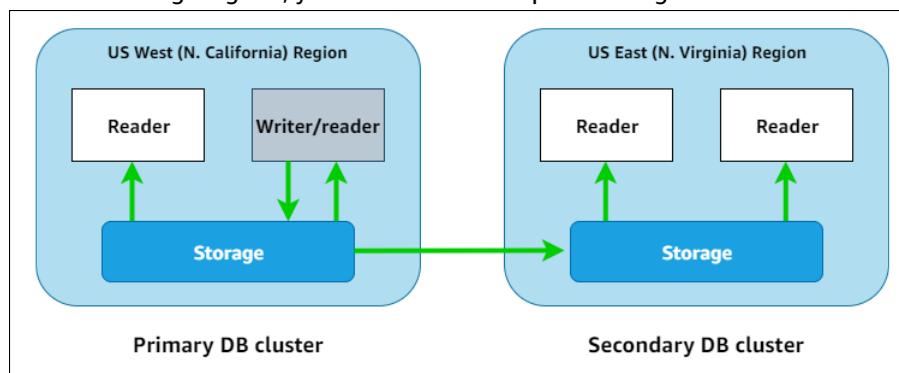
- [Overview of Amazon Aurora global databases \(p. 151\)](#)
- [Advantages of Amazon Aurora global databases \(p. 152\)](#)
- [Limitations of Amazon Aurora global databases \(p. 152\)](#)
- [Getting started with Amazon Aurora global databases \(p. 154\)](#)
- [Managing an Amazon Aurora global database \(p. 175\)](#)
- [Connecting to an Amazon Aurora global database \(p. 180\)](#)
- [Using write forwarding in an Amazon Aurora global database \(p. 181\)](#)
- [Using failover in an Amazon Aurora global database \(p. 192\)](#)
- [Monitoring an Amazon Aurora global database \(p. 202\)](#)
- [Using Amazon Aurora global databases with other AWS services \(p. 205\)](#)
- [Upgrading an Amazon Aurora global database \(p. 206\)](#)

## 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 \(p. 67\)](#).

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 \(p. 181\)](#).

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 \(p. 193\)](#).

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 \(p. 194\)](#).

## 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 \(p. 154\)](#).
- 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 \(p. 619\)](#).

- The following limitations currently apply to upgrading Aurora global databases:
  - You can't apply a custom parameter group to the global database cluster while you're performing a major version upgrade of that Aurora global database. You create your custom parameter groups in each Region of the global cluster and you apply them manually to the Regional clusters after the upgrade.
  - 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 \(p. 198\)](#).

For information about upgrading an Aurora global database, see [Upgrading an Amazon Aurora global database \(p. 206\)](#).

- 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 \(p. 244\)](#).
- 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 \(p. 73\)](#). 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 \(p. 193\)](#).
- 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 \(p. 155\)](#).

## 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 \(p. 155\)](#). 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 \(p. 168\)](#).
- **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 \(p. 168\)](#).

Before creating an Aurora global database, we recommend that you understand all configuration requirements.

### Topics

- [Configuration requirements of an Amazon Aurora global database \(p. 154\)](#)
- [Creating an Amazon Aurora global database \(p. 155\)](#)
- [Adding an AWS Region to an Amazon Aurora global database \(p. 168\)](#)
- [Creating a headless Aurora DB cluster in a secondary Region \(p. 172\)](#)
- [Using a snapshot for your Amazon Aurora global database \(p. 174\)](#)

## 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)

Description	Primary AWS Region	Secondary AWS Regions
Writer instances	1	0
Read-only instances (Aurora replicas), per Aurora DB cluster	15 (max)	16 (total)
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 \(p. 87\)](#). You also need to complete other preliminary steps for creating any Aurora DB cluster. To learn more, see [Creating an Amazon Aurora DB cluster \(p. 127\)](#).

## 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 \(p. 168\)](#).

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 \(p. 21\)](#).

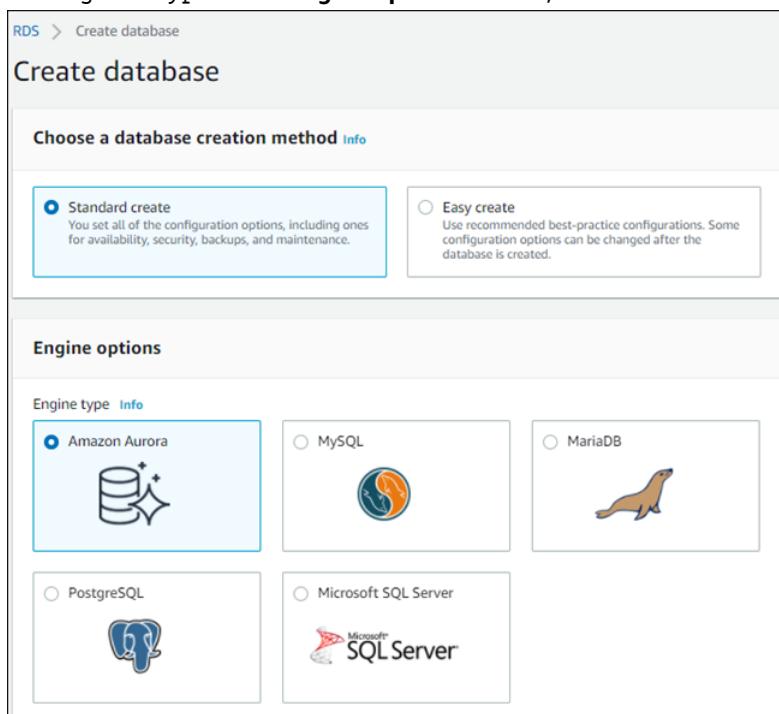
One of the following steps is choosing a virtual private cloud (VPC) based on Amazon VPC for your Aurora DB cluster. To use your own VPC, we recommend that you create it in advance so it's available for you to choose. At the same time, create any related subnets, and as needed a subnet group and security group. To learn how, see [Tutorial: Create an Amazon VPC for use with a DB instance](#).

For general information about creating an Aurora DB cluster, see [Creating an Amazon Aurora DB cluster \(p. 127\)](#).

### 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**.



Continue creating your Aurora global database by using the steps from the following procedures:

- [Creating a global database using Aurora MySQL \(p. 156\)](#)
- [Creating a global database using Aurora PostgreSQL \(p. 160\)](#)

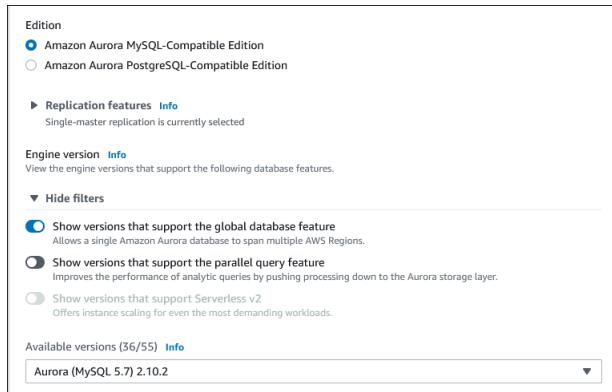
### [Creating a global database using Aurora MySQL](#)

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 \(p. 158\)](#).

#### **To create an Aurora global database using Aurora MySQL**

Complete the **Create database** page.

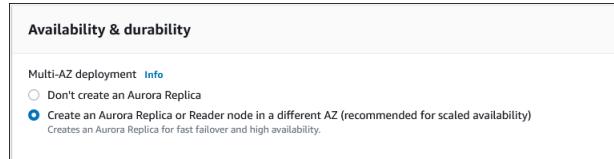
1. For **Engine options**, choose the following:
  - a. For **Edition**, choose **Amazon Aurora MySQL-Compatible Edition**.
  - b. Leave **Replication features** set to the default (single-master replication).
  - c. Expand **Show filters**, and then turn on **Show versions that support the global database feature**.
  - d. For **Engine version**, choose the version of Aurora MySQL that you want to use for your Aurora global database.



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.

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.

5. 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.



6. 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.
7. 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.
8. For **Additional configuration**, do the following:
  - a. 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.
  - b. 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**.
9. 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="C"/> <input type="button" value="Modify"/> Actions <input type="button" value="Restore from S3"/> <input type="button" value="Create database"/>						
<input type="button" value="Filter databases"/>						
	lab-demo-db-cluster	Regional	Aurora PostgreSQL	us-west-1	1 instance	<span style="color: green;">Available</span>
	lab-west-db-cluster	Regional	Aurora MySQL	us-west-1	2 instances	<span style="color: green;">Available</span>
	lab-west-db-cluster-instance-1	Writer	Aurora MySQL	us-west-1b	db.r4.large	<span style="color: green;">Available</span>
	lab-west-db-cluster-instance-1-us-west-1c	Reader	Aurora MySQL	us-west-1c	db.r4.large	<span style="color: green;">Available</span>

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 \(p. 168\)](#).

### 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 \(p. 156\)](#).

#### To create an Aurora global database using Aurora MySQL 5.6.10a

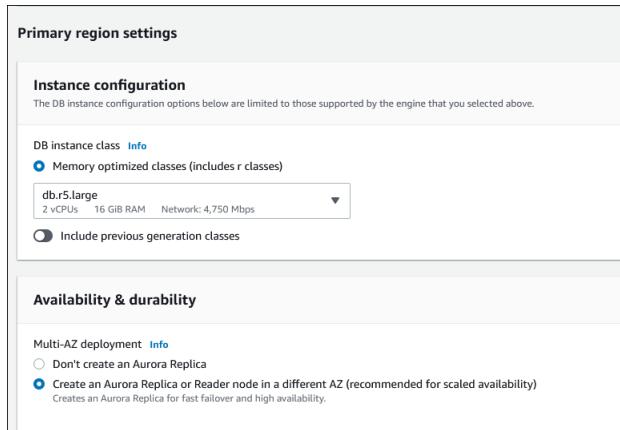
Complete the **Create database** page.

1. For **Engine options**, choose the following:
  - a. For **Edition**, choose **Amazon Aurora MySQL-Compatible Edition**.

- b. Leave **Replication features** set to the default (single-master replication).
  - c. Expand **Show filters**, and then turn on **Show versions that support the global database feature**.
  - d. For **Engine 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 **admin** 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.

The screenshot shows the 'Global database settings' configuration page. The 'Settings' tab is selected. Under 'Global database identifier', the value 'lab-ams5610a-global-database' is entered. Under 'Master username', the value 'admin' is entered. Under 'Master password', the value '\*\*\*\*\*' is shown. There are also sections for 'Confirm password' and 'Auto generate a password'.

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
	Primary	Aurora MySQL	us-west-1	2 instances	
	Reader	Aurora MySQL	us-west-1b	db.r4.large	
	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 \(p. 168\)](#).

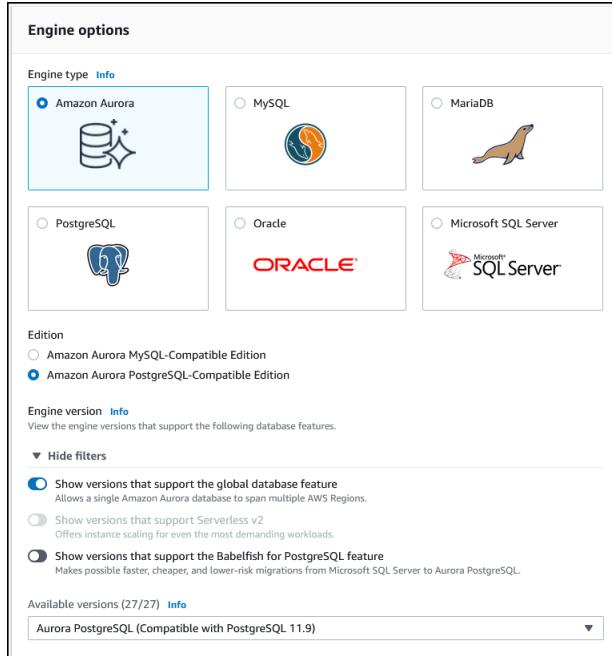
### Creating a global database using Aurora PostgreSQL

#### 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 PostgreSQL-Compatible Edition**.

- b. Expand **Show filters**, and then turn on **Show versions that support the global database feature**.
- c. For **Engine 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](#)

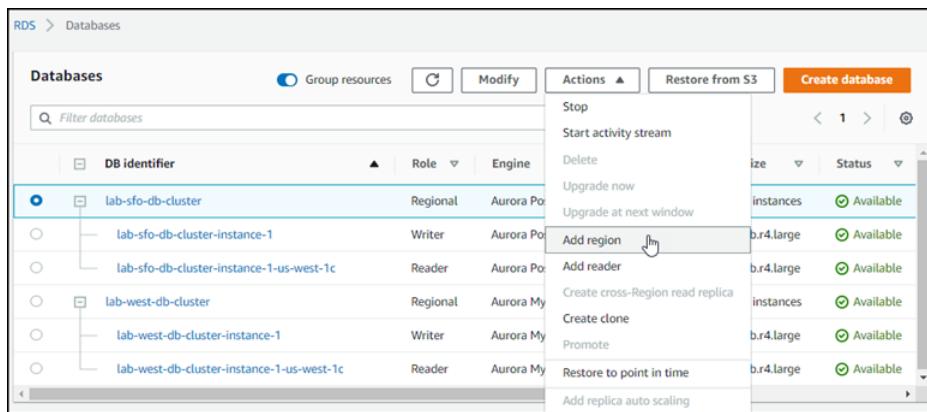
- For **DB instance class**, choose `db.r5.large` or another memory optimized DB instance class. We recommend that you use a `db.r5` or higher instance class.

**Instance configuration**  
The DB Instance configuration options below are limited to those supported by the engine that you selected above.

**DB instance class** [Info](#)  
 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

- 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.
- (Optional) Complete the **Database authentication** settings. Password authentication is always enabled. To simplify the process, you can skip this section 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 Encryption, 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 \(p. 168\)](#).

### 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 \(p. 168\)](#).

Follow the procedure for your Aurora database engine.

#### CLI examples

- [Creating a global database using Aurora MySQL \(p. 163\)](#)
- [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 \(p. 168\)](#).

### 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 \(p. 168\)](#).

## 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 \(p. 154\)](#).

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.

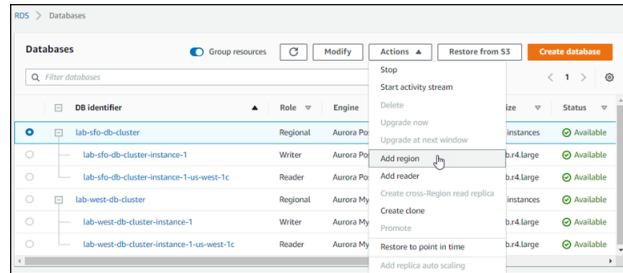
#### Note

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 parameter groups \(p. 215\)](#).

#### 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**.



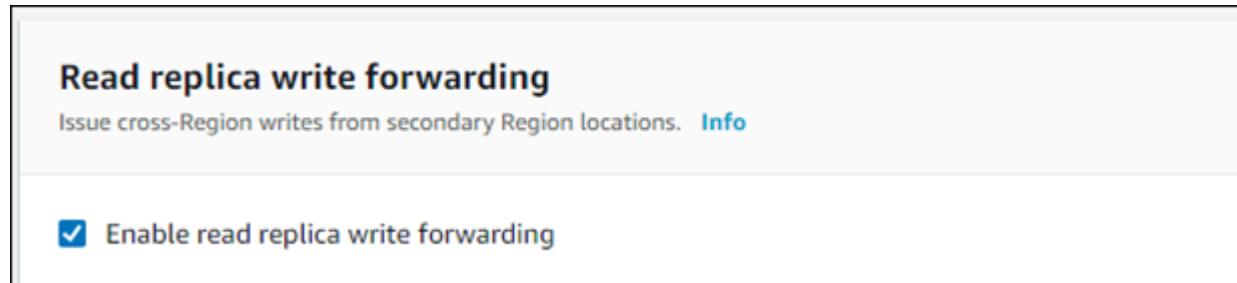
5. On the **Add a region** page, choose the secondary AWS Region.

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 fields for 'Global database identifier' (set to 'lab-east-west-global'), 'Region' (set to 'US East (N. Virginia)'), and other optional settings like 'Secondary region'.

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 \(p. 181\)](#).



## 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		<input type="checkbox"/> Group resources		Modify	Actions ▾	Restore from S3
		<input type="text"/> 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 \(p. 168\)](#).

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	<span>Available</span>
ams57west	Primary	Aurora MySQL	us-west-1	2 instances	<span>Available</span>
ams57west-instance-1	Writer	Aurora MySQL	us-west-1b	db.r5.large	<span>Available</span>
ams57west-instance-1-us-west-1c	Reader	Aurora MySQL	us-west-1c	db.r5.large	<span>Available</span>
west-coast-global-cluster-1	Secondary	Aurora MySQL	us-west-2	1 instance	<span>Available</span>
west-coast-global-instance-1	Reader	Aurora MySQL	us-west-2a	db.r5.large	<span>Available</span>

- 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	<span>Available</span>
ams57west	Primary	Aurora MySQL	us-west-1	2 instances	<span>Available</span>
west-coast-global-cluster-1	Secondary	Aurora MySQL	us-west-2	1 instance	<span>Available</span>
west-coast-global-instance-1	Reader	Aurora MySQL	us-west-2a	db.r5.large	<span>Available</span>

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	<span style="color: green;">Available</span>
west-coast-global		Global	Aurora MySQL	2 regions	2 clusters	<span style="color: green;">Available</span>
ams57west		Primary	Aurora MySQL	us-west-1	2 instances	<span style="color: green;">Available</span>
ams57west-instance-1		Writer	Aurora MySQL	us-west-1b	db.r5.large	<span style="color: green;">Available</span>
ams57west-instance-1-us-west-1c		Reader	Aurora MySQL	us-west-1c	db.r5.large	<span style="color: green;">Available</span>
west-coast-global-cluster-1		Secondary	Aurora MySQL	us-west-2	0 instances	<span style="color: green;">Available</span>

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 \(p. 193\)](#) 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 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.

The screenshot shows the 'Restore snapshot' configuration page. At the top, it says 'RDS > Snapshots > Restore snapshot'. The main title is 'Restore snapshot'. A note below says: '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.' The 'DB specifications' section includes:

- Engine:** Amazon Aurora with MySQL compatibility (selected)
- Capacity type:** Provisioned (selected) - 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:** Single-master replication is currently selected.
- Engine version:** Aurora (MySQL 5.6) 1.22.1 (selected)
- Other options:** Show versions that support the global database feature (selected), Show versions that support the parallel query feature.

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 \(p. 194\)](#).

To recover an Aurora global database from an unplanned outage in its primary Region, see [Using failover in an Amazon Aurora global database \(p. 192\)](#).

### Topics

- [Modifying an Amazon Aurora global database \(p. 175\)](#)
- [Modifying parameters for an Aurora global database \(p. 176\)](#)
- [Removing a cluster from an Amazon Aurora global database \(p. 177\)](#)
- [Deleting an Amazon Aurora global database \(p. 179\)](#)

## 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-cont-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 \(p. 193\)](#).

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 \(p. 179\)](#).

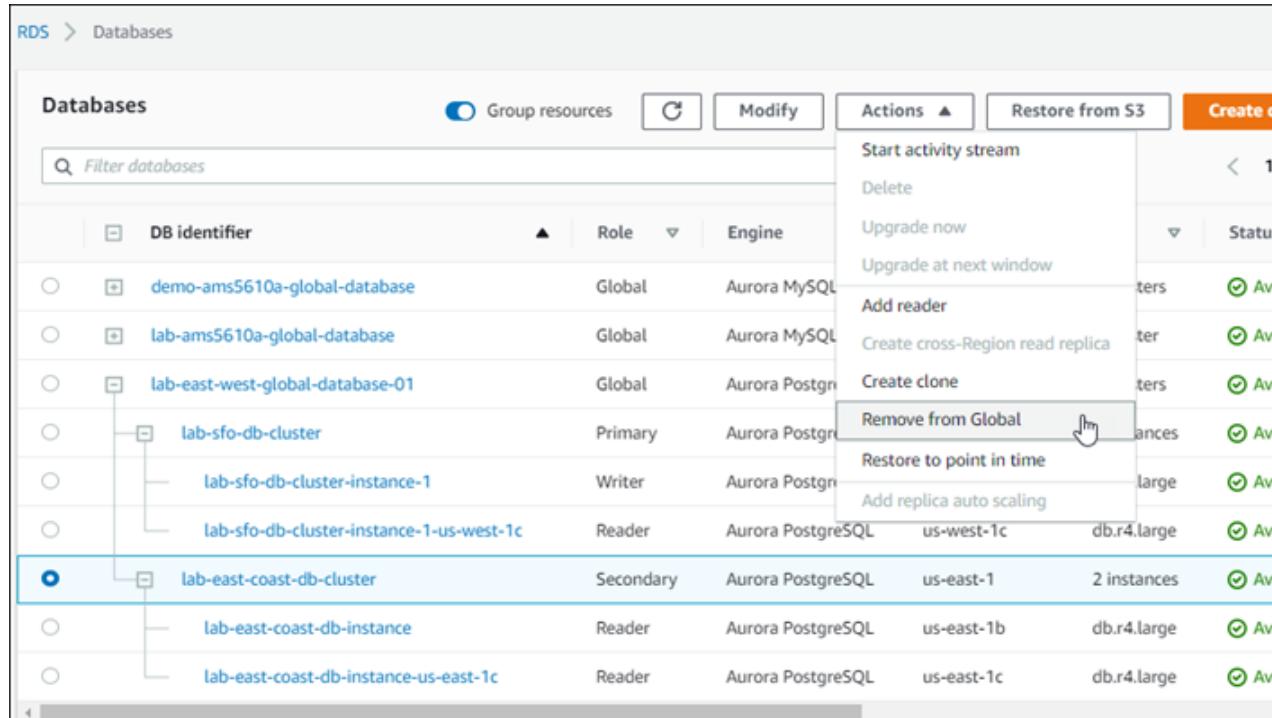
When an Aurora DB cluster is detached from the Aurora global database, it's no longer synchronized with the primary. It becomes a standalone provisioned Aurora DB cluster with full read/write capabilities.

You can remove Aurora DB clusters from your Aurora global database using the AWS Management Console, the AWS CLI, or the RDS API.

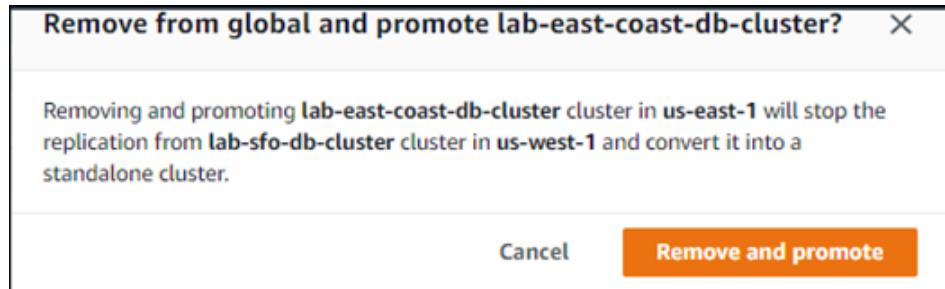
### Console

#### 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.

		Regional	Aurora PostgreSQL	us-east-1	2 instances	
○	lab-east-coast-db-cluster					
○	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 \(p. 179\)](#).

## 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 \(p. 177\)](#).
- 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 \(p. 350\)](#).

To delete an Aurora global database, you can use the AWS Management Console, the AWS CLI, or the RDS API.

### Console

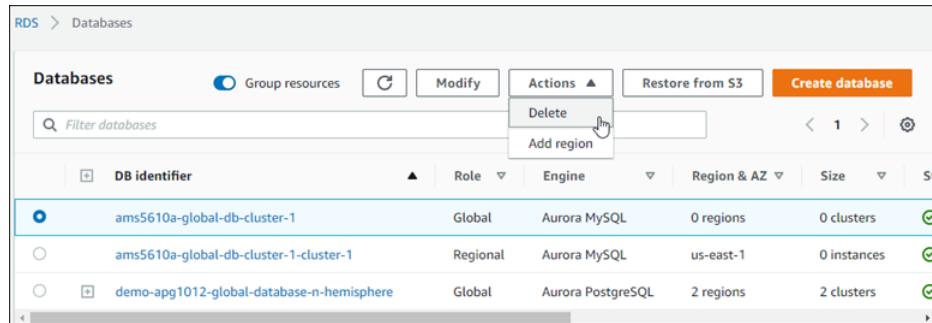
#### To delete an Aurora global database

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 \(p. 177\)](#).

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

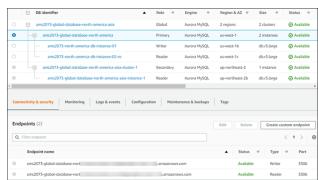
To delete a cluster that is part of an Aurora global database, run the [DeleteGlobalCluster](#) API operation.

# 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 \(p. 181\)](#)
- [Checking if a secondary cluster has write forwarding enabled \(p. 183\)](#)
- [Application and SQL compatibility with write forwarding \(p. 184\)](#)
- [Isolation and consistency for write forwarding \(p. 185\)](#)
- [Running multipart statements with write forwarding \(p. 188\)](#)
- [Transactions with write forwarding \(p. 188\)](#)
- [Configuration parameters for write forwarding \(p. 188\)](#)
- [Amazon CloudWatch metrics for write forwarding \(p. 189\)](#)

## 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 \(p. 185\)](#).

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 \(p. 978\)](#).

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 \(p. 188\)](#).

## 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 \(p. 185\)](#). 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.

<b>CloudWatch Metric (Aurora MySQL status variable)</b>	<b>Units and description</b>
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.

<b>CloudWatch Metric</b>	<b>Units and description</b>
<b>(Aurora MySQL status variable)</b>	
–  (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.

<b>CloudWatch Metric</b>	<b>Unit and description</b>
<b>(Aurora MySQL status variable)</b>	
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 \(p. 71\)](#). 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 \(p. 198\)](#).

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 \(p. 193\)](#). 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 \(p. 194\)](#).

## Topics

- [Recovering an Amazon Aurora global database from an unplanned outage \(p. 193\)](#)
- [Performing managed planned failovers for Amazon Aurora global databases \(p. 194\)](#)
- [Managing RPOs for Aurora PostgreSQL-based global databases \(p. 198\)](#)

## 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 \(p. 177\)](#).

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 \(p. 168\)](#).

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 \(p. 194\)](#).

## 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 \(p. 193\)](#).

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 \(p. 525\)](#).

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 \(p. 176\)](#).
- **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 \(p. 205\)](#). 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.

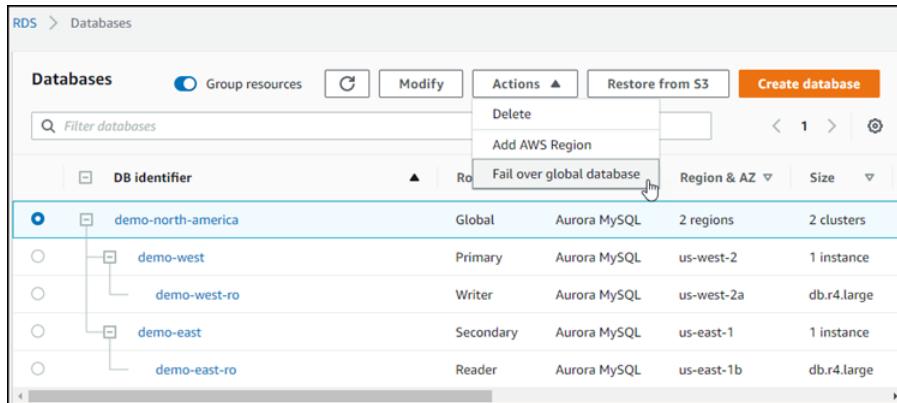
For example, the secondary cluster's endpoint `my-global.cluster-ro-aaaaaabbbbb.us-west-1.rds.amazonaws.com` becomes `my-global.cluster-aaaaaabbbbb.us-west-1.rds.amazonaws.com` when that cluster is promoted to primary.

You can fail over your Aurora global database using the AWS Management Console, the AWS CLI, or the RDS API.

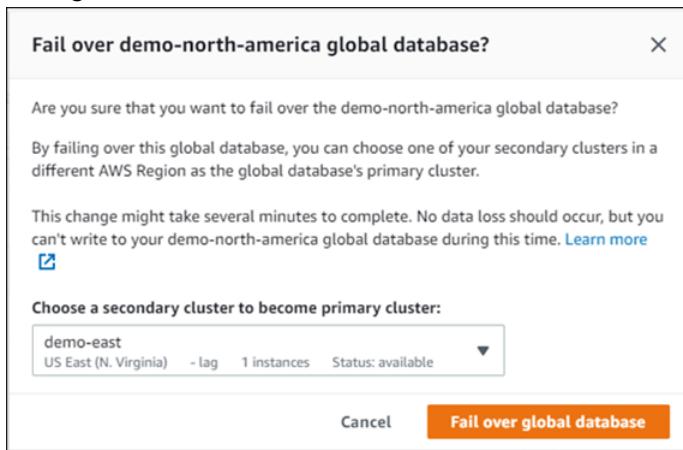
## Console

### 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 \(p. 203\)](#).

### Topics

- [Setting the recovery point objective \(p. 199\)](#)
- [Viewing the recovery point objective \(p. 200\)](#)
- [Disabling the recovery point objective \(p. 201\)](#)

## 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 \(p. 221\)](#).

### 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 \(p. 226\)](#).

## 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 \(p. 221\)](#).

### 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

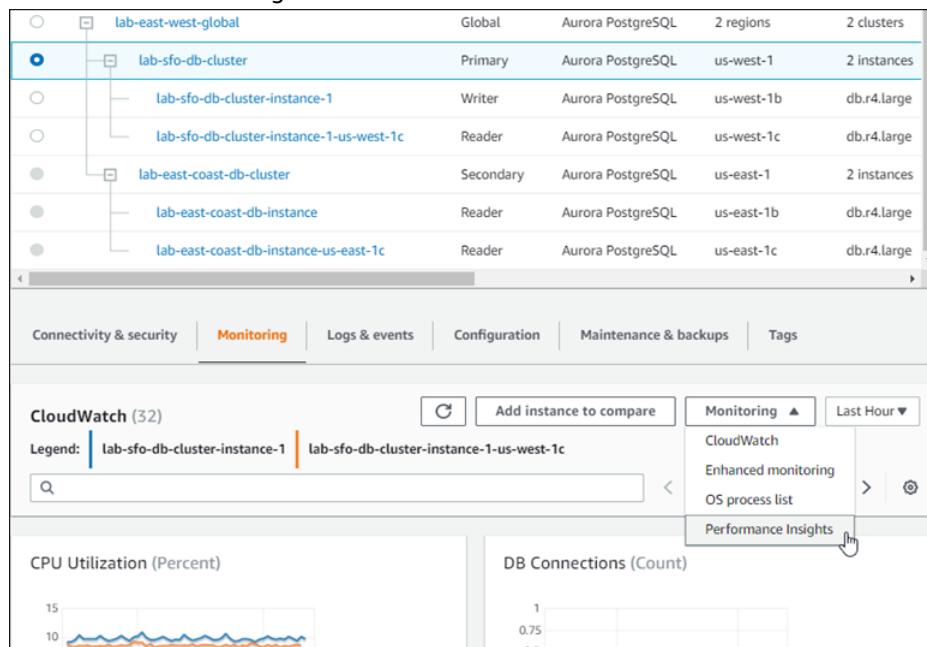
To reset the `rds.global_db_rpo` parameter, use the Amazon RDS API [ResetDBClusterParameterGroup](#) operation.

# 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 \(p. 203\)](#).
- 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 \(p. 203\)](#).

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 \(p. 427\)](#).

## 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 \(p. 461\)](#).

## 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 \(p. 180\)](#).
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 \(p. 917\)](#).
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 \(p. 897\)](#) 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 \(p. 902\)](#).

### 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 \(p. 903\)](#).

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 \(p. 897\)](#) 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 \(p. 902\)](#).

### 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 \(p. 911\)](#).
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 \(p. 897\)](#) 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 \(p. 902\)](#).

## 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 [Major upgrades for global databases \(p. 1424\)](#). To learn how to upgrade an Aurora MySQL global database to a higher major version, see [In-place major upgrades for global databases \(p. 1009\)](#).

#### 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 [Upgrading the Aurora PostgreSQL engine to a new major version \(p. 1422\)](#). To learn how to upgrade an Aurora MySQL global database to a higher minor version, see [Upgrading Aurora MySQL by modifying the engine version \(p. 996\)](#).

# 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 DB clusters, you connect to the database endpoint rather than to the DB instance. You can find the database endpoint for an Aurora Serverless DB cluster on the **Connectivity & security** tab of the AWS Management Console. For more information, see [Using Amazon Aurora Serverless v1 \(p. 1543\)](#).

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.

For more information, see [Troubleshooting Aurora connection failures \(p. 214\)](#).

## Topics

- [Connecting to an Amazon Aurora MySQL DB cluster \(p. 207\)](#)
- [Connecting to an Amazon Aurora PostgreSQL DB cluster \(p. 212\)](#)
- [Troubleshooting Aurora connection failures \(p. 214\)](#)

# 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 \(p. 1683\)](#).

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 MySQL 8.0 Community Edition \(p. 663\)](#).

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 \(p. 680\)](#).

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 managing databases. The main navigation bar at the top includes 'RDS', 'Databases', and 'aurora-cl-mysql'. Below the navigation, the database name 'aurora-cl-mysql' is displayed with 'Modify' and 'Actions' buttons. A 'Related' section allows filtering databases. The main table lists database identifiers, their roles, engines, regions, and sizes. The 'aurora-cl-mysql' cluster is listed as Regional, Aurora MySQL, us-east-1, with 3 instances. It contains four sub-instances: dbinstance4 (Writer, us-east-1a, db.r5.large), dbinstance1 (Reader, us-east-1b, db.r5.large), and dbinstance2 (Reader, 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' section shows two entries: 'aurora-cl-mysql.cluster-ro' (available, Reader, 3306) and 'aurora-cl-mysql.cluster' (available, Writer, 3306). The 'aurora-cl-mysql.cluster' endpoint is highlighted with a red border.

DB identifier	Role	Engine	Region & AZ	Size
aurora-cl-mysql	Regional	Aurora MySQL	us-east-1	3 instances
dbinstance4	Writer	Aurora MySQL	us-east-1a	db.r5.large
dbinstance1	Reader	Aurora MySQL	us-east-1b	db.r5.large
dbinstance2	Reader	Aurora MySQL	us-east-1b	db.r5.large

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

## Topics

- [Connection utilities for Aurora MySQL \(p. 209\)](#)
- [Connecting with Aurora MySQL using the MySQL utility \(p. 210\)](#)
- [Connecting with the Amazon Web Services JDBC Driver for MySQL \(p. 211\)](#)
- [Connecting with SSL for Aurora MySQL \(p. 211\)](#)

## 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 AWS JDBC Driver for MySQL to connect your client applications to an Aurora MySQL DB cluster. 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](#).

**Note**

Version 3.0.3 of the MariaDB Connector/J utility drops support for Aurora DB clusters, so we highly recommend moving to the AWS JDBC Driver for MySQL. The AWS JDBC Driver for MySQL offers improved failover speed for Aurora MySQL DB clusters by caching DNS connections for quick use.

If you are using an Aurora Serverless DB cluster, the failover benefits don't apply, but you can disable the feature by setting the `failureDetectionEnabled` parameter to `false`. To review a complete list of configuration options, see the [AWS JDBC Driver for MySQL GitHub repository](#).

## Connecting with Aurora MySQL using the MySQL utility

Use the following procedure. It assumes that you configured your DB cluster in a private subnet in your VPC. You connect using an Amazon EC2 instance that you configured according to the tutorials in [Tutorial: Create a web server and an Amazon Aurora DB cluster \(p. 107\)](#).

**Note**

This procedure doesn't require installing the web server in the tutorial, but it does require installing MariaDB 10.5.

### To connect to a DB cluster using the MySQL utility

1. Log in to the EC2 instance that you're using to connect to your DB cluster.

You should see output similar to the following.

```
Last login: Thu Jun 23 13:32:52 2022 from xxx.xxx.xxx.xxx
      _\   _ ) 
      _ \  /   Amazon Linux 2 AMI
      __| \__|_|_
https://aws.amazon.com/amazon-linux-2/
[ec2-user@ip-10-0-xxx.xxx ~]$
```

2. Type the following command at the command prompt to connect to the primary DB instance of your DB cluster.

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.

```
mysql -h primary-instance-endpoint.AWS_account.AWS_Region.rds.amazonaws.com -P 3306 -
u database_user -p
```

For example:

```
mysql -h my-aurora-cluster-instance.c1xy5example.123456789012.eu-
central-1.rds.amazonaws.com -P 3306 -u admin -p
```

3. Enter the password for the database user.

You should see output similar to the following.

```
Welcome to the MariaDB monitor. Commands end with ; or \g.
```

```
Your MySQL connection id is 1770
Server version: 8.0.23 Source distribution

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MySQL [(none)]>
```

4. Enter your SQL commands.

## Connecting with the Amazon Web Services JDBC Driver for MySQL

The AWS JDBC Driver for MySQL is a client driver designed for the high availability of Aurora MySQL. The driver is drop-in compatible with the MySQL Connector/J driver. To install or upgrade your connector, replace the MySQL connector .jar file (located in the application CLASSPATH) with the AWS JDBC Driver for MySQL .jar file, and update the connection URL prefix from `jdbc:mysql://` to `jdbc:mysql:aws://`.

The AWS JDBC Driver for MySQL takes full advantage of the failover capabilities of Aurora MySQL. In the event of a failover, the driver queries the cluster directly for the new topology instead of using DNS resolution. By querying the cluster directly, the driver is able to connect to the new primary faster in a reliable and predictable manner, avoiding potential delays caused by DNS resolution.

The AWS JDBC Driver for MySQL supports IAM database authentication. For more information, see [AWS IAM Database Authentication](#) in the AWS JDBC Driver for MySQL GitHub repository. For more information about IAM database authentication, see [IAM database authentication \(p. 1683\)](#).

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 with SSL for Aurora MySQL

You can use SSL encryption on connections to an Aurora MySQL DB instance. For information, see [Using SSL/TLS with Aurora MySQL DB clusters \(p. 683\)](#).

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 \(p. 1683\)](#).

### 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 \(p. 35\)](#).

### 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 \(p. 1642\)](#).

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]global-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 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 an Aurora PostgreSQL database. At the top, the navigation path is RDS > Databases > aurora-cl-postgresql. The main title is "aurora-cl-postgresql". On the right, there are "Modify" and "Actions" buttons. Below the title, a "Related" section and a search bar ("Filter databases") are visible. The main content area displays a table with columns: DB identifier, Role, Engine, Region & AZ, and Size. The table lists three items: "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, tabs for Connectivity & security, Monitoring, Logs & events, Configuration, Maintenance & backups, and Tags are present. The "Connectivity & security" tab is selected. Under "Endpoints (2)", a table shows two entries: "aurora-cl-postgresql.cluster-ro-...us-east-1.rds.amazonaws.com" (Available, Reader, 5432) and "aurora-cl-postgresql.cluster-...us-east-1.rds.amazonaws.com" (Available, Writer, 5432). The "Writer" endpoint is highlighted with a red border. At the bottom, there is a "Manage IAM roles" button.

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

Endpoint name	Status	Type	Port
aurora-cl-postgresql.cluster-ro-...us-east-1.rds.amazonaws.com	Available	Reader	5432
aurora-cl-postgresql.cluster-...us-east-1.rds.amazonaws.com	Available	Writer	5432

## 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 [Tutorial: Create a VPC for use with a DB cluster \(IPv4 only\) \(p. 1744\)](#).
- **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 \(p. 1683\)](#).

For more information about troubleshooting Aurora DB connection issues, see [Can't connect to Amazon RDS DB instance \(p. 1761\)](#).

# Working with 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. Aurora 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 DB clusters to use your own parameter groups.

For information about modifying a DB cluster or DB instance, see [Modifying an Amazon Aurora DB cluster \(p. 248\)](#).

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.

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.

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.

To use your own parameter group, you create a new parameter group and modify the parameters that you want to modify. 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 an Aurora 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:

- DB cluster parameters are either *static* or *dynamic*. When you change a static parameter and save the DB cluster parameter group, the parameter change takes effect after you manually reboot the DB instances in each associated DB cluster.

When you change a dynamic parameter, by default the parameter change is applied to your DB cluster immediately, without requiring a reboot. To defer the parameter change until after the DB instances in an associated DB cluster are rebooted, use the AWS CLI or RDS API, and set the `ApplyMethod` to `pending-reboot` for the parameter change.

When you use the AWS Management Console to change DB cluster parameter values, it always uses `immediate` for the `ApplyMethod` for dynamic parameters. For static parameters, the AWS Management Console always uses `pending-reboot` for the `ApplyMethod`.

For more information about using the AWS CLI to change a parameter value, see [modify-db-cluster-parameter-group](#). For more information about using the RDS API to change a parameter value, see [ModifyDBClusterParameterGroup](#).

- DB instance 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 associated DB instances.

When you change a dynamic parameter, by default the parameter change is applied to your DB instance immediately, without requiring a reboot. To defer the parameter change until after an associated DB instance is rebooted, use the AWS CLI or RDS API, and set the `ApplyMethod` to `pending-reboot` for the parameter change.

When you use the AWS Management Console to change DB instance parameter values, it always uses `immediate` for the `ApplyMethod` for dynamic parameters. For static parameters, the AWS Management Console always uses `pending-reboot` for the `ApplyMethod`.

For more information about using the AWS CLI to change a parameter value, see [modify-db-parameter-group](#). For more information about using the RDS API to change a parameter value, see [ModifyDBParameterGroup](#).

- 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 \(p. 248\)](#).
- After you change the DB cluster parameter group associated with a DB cluster, reboot the DB instances in the DB cluster to apply the changes to all of the DB instances in the DB cluster.

To determine whether the DB instances 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 DB instances of the DB cluster.

- In many cases, you can specify integer and Boolean parameter values using expressions, formulas, and functions. Functions can include a mathematical log expression. However, not all parameters support expressions, formulas, and functions for parameter values. For more information, see [Specifying DB parameters \(p. 238\)](#).
- Set any parameters that relate to the character set or collation of your database in your parameter group before creating the DB cluster and before you create a database in it. This ensures that the default database and new databases use the character set and collation values that you specify. If you change character set or collation parameters, the parameter changes aren't 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 be cautious 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 or DB cluster before applying those parameter group changes to a production DB instance or DB cluster.
- 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 instance or DB cluster. For more information, see [Viewing parameter values for a DB parameter group \(p. 237\)](#) and [Viewing parameter values for a DB cluster parameter group \(p. 226\)](#).

#### Topics

- [Working with DB cluster parameter groups \(p. 217\)](#)
- [Working with DB parameter groups \(p. 228\)](#)
- [Comparing parameter groups \(p. 238\)](#)
- [Specifying DB parameters \(p. 238\)](#)

## Working with DB cluster parameter groups

Amazon Aurora DB clusters use DB cluster parameter groups. The following sections describe configuring and managing DB cluster parameter groups.

#### Topics

- [Amazon Aurora DB cluster and DB instance parameters \(p. 217\)](#)
- [Creating a DB cluster parameter group \(p. 219\)](#)
- [Associating a DB cluster parameter group with a DB cluster \(p. 220\)](#)
- [Modifying parameters in a DB cluster parameter group \(p. 221\)](#)
- [Resetting parameters in a DB cluster parameter group \(p. 223\)](#)
- [Copying a DB cluster parameter group \(p. 224\)](#)
- [Listing DB cluster parameter groups \(p. 225\)](#)
- [Viewing parameter values for a DB cluster parameter group \(p. 226\)](#)

## Amazon Aurora DB cluster and DB instance parameters

Aurora uses a two-level system of configuration settings:

- 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. This parameter group assigns default values for every configuration value for the corresponding DB engine. The cluster parameter group includes defaults for both the cluster-level and instance-level parameters. Each DB instance within a provisioned or Aurora Serverless v2 cluster inherits the settings from that DB cluster parameter group.

Each DB instance is also associated with a DB parameter group. The values in the DB parameter group can override default values from the cluster parameter group. For example, if one instance in a cluster experienced issues, you might assign a custom DB parameter group to that instance. The custom parameter group might have specific settings for parameters related to debugging or performance tuning.

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 specify custom parameter groups instead. You create those parameter groups yourself, and you can edit the parameter values. You can specify these custom parameter groups at creation time. You can also modify a DB cluster or instance later to use a custom parameter group.

For provisioned and Aurora Serverless v2 instances, 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 operation. 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	<p>See <a href="#">Aurora MySQL configuration parameters (p. 949)</a>.</p> <p>For Aurora Serverless clusters, see additional details in <a href="#">Working with parameter groups for Aurora Serverless v2 (p. 1535)</a> and <a href="#">Parameter groups for Aurora Serverless v1 (p. 1554)</a>.</p>
Aurora PostgreSQL	<p>See <a href="#">Amazon Aurora PostgreSQL parameters (p. 1369)</a>.</p> <p>For Aurora Serverless clusters, see additional details in <a href="#">Working with parameter groups for Aurora Serverless v2 (p. 1535)</a> and <a href="#">Parameter groups for Aurora Serverless v1 (p. 1554)</a>.</p>

#### Note

Aurora Serverless v1 clusters have only DB cluster parameter groups, not DB parameter groups. For Aurora Serverless v2 clusters, you make all your changes to custom parameters in the DB cluster parameter group.

Aurora Serverless v2 uses both DB cluster parameter groups and DB parameter groups. With Aurora Serverless v2, you can modify almost all of the configuration parameters. Aurora Serverless v2 overrides the settings of some capacity-related configuration parameters so that your workload isn't interrupted when Aurora Serverless v2 instances scale down. To learn more about Aurora Serverless configuration settings and which settings you can modify, see [Working with parameter groups for Aurora Serverless v2 \(p. 1535\)](#) and [Parameter groups for Aurora Serverless v1 \(p. 1554\)](#).

## 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.

After you create a DB cluster parameter group, wait at least 5 minutes before creating a DB cluster that uses that DB cluster parameter group. Doing this allows Amazon RDS to fully create the parameter group before it is used by the new DB cluster. You can use the **Parameter groups** page in the [Amazon RDS console](#) or the `describe-db-cluster-parameters` command to verify that your DB cluster parameter group is created.

### 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 Aurora MySQL version 5.7 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 aurora-mysql5.7 \
    --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 aurora-mysql5.7 ^
    --description "My new cluster parameter group"
```

This command produces output similar to the following:

```
{
    "DBClusterParameterGroup": {
        "DBClusterParameterGroupName": "mydbclusterparametergroup",
        "DBParameterGroupFamily": "aurora-mysql5.7",
        "Description": "My new cluster parameter group",
        "DBClusterParameterGroupArn": "arn:aws:rds:us-east-1:123456789012:cluster-
pg:mydbclusterparametergroup"
    }
}
```

## 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 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 \(p. 219\)](#). For information about creating a DB cluster, see [Creating an Amazon Aurora DB cluster \(p. 127\)](#). For information about modifying a DB cluster, see [Modifying an Amazon Aurora DB cluster \(p. 248\)](#).

### 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 `mydbcluster` DB cluster.

### Example

For Linux, macOS, or Unix:

```
aws rds modify-db-cluster \
  --db-cluster-identifier mydbcluster \
  --db-cluster-parameter-group-name mydbclpg
```

For Windows:

```
aws rds modify-db-cluster ^
  --db-cluster-identifier mydbcluster ^
  --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 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 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 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 a DB cluster that uses that DB cluster parameter group. Doing this allows Amazon RDS to fully copy the parameter group before it is used by the new DB cluster. You can use the **Parameter groups** page in 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 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 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": [
        {
            "DBClusterParameterGroupName": "mydbclusterparametergroup",
            "DBParameterGroupFamily": "aurora-mysql5.7",
            "Description": "My new cluster parameter group",
            "DBClusterParameterGroupArn": "arn:aws:rds:us-east-1:123456789012:cluster-
pg:mydbclusterparametergroup"
        }
    ]
}
```

## 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 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 *mydbclusterparametergroup*, in JSON format.

The command returns a response like the following:

```
aws rds describe-db-cluster-parameters --db-cluster-parameter-group-name mydbclusterparametergroup
```

```
{  
    "Parameters": [  
        {  
            "ParameterName": "allow-suspicious-udfs",  
            "Description": "Controls whether user-defined functions that have only an xxx symbol for the main function can be loaded",  
            "Source": "engine-default",  
            "ApplyType": "static",  
            "DataType": "boolean",  
            "AllowedValues": "0,1",  
            "IsModifiable": false,  
            "ApplyMethod": "pending-reboot",  
            "SupportedEngineModes": [  
                "provisioned"  
            ]  
        },  
        {  
            "ParameterName": "aurora_binlog_read_buffer_size",  
            "ParameterValue": "5242880",  
            "Description": "Read buffer size used by master dump thread when the switch aurora_binlog_use_large_read_buffer is ON.",  
            "Source": "engine-default",  
            "ApplyType": "dynamic",  
            "DataType": "integer",  
            "AllowedValues": "8192-536870912",  
            "IsModifiable": true,  
            "ApplyMethod": "pending-reboot",  
            "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`

# Working with DB parameter groups

DB instances use DB parameter groups. The following sections describe configuring and managing DB instance parameter groups.

## Topics

- [Creating a DB parameter group \(p. 228\)](#)
- [Associating a DB parameter group with a DB instance \(p. 229\)](#)
- [Modifying parameters in a DB parameter group \(p. 231\)](#)
- [Resetting parameters in a DB parameter group to their default values \(p. 233\)](#)
- [Copying a DB parameter group \(p. 235\)](#)
- [Listing DB parameter groups \(p. 236\)](#)
- [Viewing parameter values for a DB parameter group \(p. 237\)](#)

## 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 8.0 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`

## 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 \(p. 228\)](#). For information about modifying a DB instance, see [Modify a DB instance in a DB cluster \(p. 249\)](#).

**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`

## 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. The top navigation bar includes 'RDS > Databases > cluster-2 > cluster-2-instance-1'. The main title is 'cluster-2-instance-1'. Below it is a 'Related' section with a search bar for 'Filter databases'. A table lists DB identifiers, including '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 border. Below the table, there are tabs for 'Connectivity & security', 'Monitoring', 'Logs & events', 'Configuration' (highlighted), 'Maintenance', and 'Tags'. The 'Instance' section contains detailed configuration information for the DB instance, such as 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-...'), and Created time ('Fri Apr 03 2020 10:48:37 GMT-0400 (Eastern Daylight Time)'). At the bottom of the instance details, the 'Parameter group' is listed as 'test-aurora56-instance (pending-reboot)', which is also highlighted with a red box.

### Console

#### To modify 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 parameter group that you want to modify.
4. For **Parameter group actions**, choose **Edit**.
5. 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.
6. 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`

## 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. Below this, the instance name 'cluster-2-instance-1' is displayed. A 'Related' section lists 'DB identifier' entries: '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 main content area has tabs for Connectivity & security, Monitoring, Logs & events, Configuration (which is highlighted with a red box), Maintenance, and Tags. The 'Instance' tab is selected. On the left, under 'Configuration', details are shown: 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-...), and Created time (Fri Apr 03 2020 10:48:37 GMT-0400 (Eastern Daylight Time)). On the right, under 'Instance class', details are shown: Instance class (db.t2.small), vCPU (1), RAM (2 GB), Availability, and Failover priority (1). At the bottom, a 'Parameter group' section shows 'test-aurora56-instance (pending-reboot)' with a red box around it.

**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.

## 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`

## 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.mysql8.0      mysql8.0  Default parameter group for MySQL8.0
DBPARAMETERGROUP mydbparametergroup     mysql8.0  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  mysql8.0  My new parameter group
```

## RDS API

To list all DB parameter groups for an AWS account, use the RDS API [DescribeDBParameterGroups](#) operation.

## 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
Type	Is Modifiable				
DBPARAMETER	allow-suspicious-udfs		engine-default	boolean	static
	false				
DBPARAMETER	auto_increment_increment		engine-default	integer	dynamic
	true				
DBPARAMETER	auto_increment_offset		engine-default	integer	dynamic
	true				
DBPARAMETER	binlog_cache_size	32768	system	integer	dynamic
	true				
DBPARAMETER	socket	/tmp/mysql.sock	system	string	static
	false				

## RDS API

To view the parameter values for a DB parameter group, use the RDS API [DescribeDBParameters](#) command with the following required parameter.

- `DBParameterGroupName`

## 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

The specified parameter groups must be for the same DB engine and version. If they aren't equivalent, you can't choose **Compare**. For example, you can't compare a MySQL 5.7 and a MySQL 8.0 parameter group because the DB engine version is different. In addition, they both must be DB parameter groups, or they both must be DB cluster parameter groups.

For example, you can't compare an Aurora MySQL 8.0 DB parameter group and an Aurora MySQL 8.0 DB cluster parameter group, even though the DB engine and version are the same.

## 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 \(p. 239\)](#)
  - [DB parameter formula variables \(p. 239\)](#)
  - [DB parameter formula operators \(p. 240\)](#)
- [DB parameter functions \(p. 240\)](#)
- [DB parameter log expressions \(p. 241\)](#)
- [DB parameter value examples \(p. 241\)](#)

## 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 \(p. 56\)](#). 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 \(p. 239\)](#).

## 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 \(p. 690\)](#).

### 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 \(p. 1048\)](#).

# 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 \(p. 244\)](#)
- [Modifying an Amazon Aurora DB cluster \(p. 248\)](#)
- [Adding Aurora Replicas to a DB cluster \(p. 270\)](#)
- [Managing performance and scaling for Aurora DB clusters \(p. 274\)](#)
- [Cloning a volume for an Amazon Aurora DB cluster \(p. 280\)](#)
- [Integrating Aurora with other AWS services \(p. 304\)](#)
- [Maintaining an Amazon Aurora DB cluster \(p. 321\)](#)
- [Rebooting an Amazon Aurora DB cluster or Amazon Aurora DB instance \(p. 329\)](#)
- [Deleting Aurora DB clusters and DB instances \(p. 345\)](#)
- [Tagging Amazon RDS resources \(p. 352\)](#)
- [Working with Amazon Resource Names \(ARNs\) in Amazon RDS \(p. 360\)](#)
- [Amazon Aurora updates \(p. 367\)](#)

# 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 \(p. 244\)](#)
- [Limitations for stopping and starting Aurora DB clusters \(p. 244\)](#)
- [Stopping an Aurora DB cluster \(p. 245\)](#)
- [Possible operations while an Aurora DB cluster is stopped \(p. 246\)](#)
- [Starting an Aurora DB cluster \(p. 246\)](#)

## 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.

### Important

You can stop a DB cluster for up to seven days. If you don't manually start your DB cluster after seven days, your DB cluster is automatically started 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 \(p. 151\)](#).
- You can't stop and start a cluster that has a cross-Region read replica.

- For a cluster that uses the [Aurora parallel query \(p. 790\)](#) feature, the minimum Aurora MySQL versions are 1.23.0 and 2.09.0.
- You can't stop and start an [Aurora Serverless v1 cluster \(p. 1543\)](#). With [Aurora Serverless v2 \(p. 1482\)](#), you can stop and start the cluster.
- You can't stop and start an [Aurora multi-master cluster \(p. 867\)](#).

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 \(p. 244\)](#).
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 \(p. 371\)](#).

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 mydbccluster
```

## 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 \(p. 251\)](#).

### 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 \(p. 251\)](#). 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 \(p. 249\)](#).

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 \(p. 251\)](#).

**Note**

Some settings only apply to DB instances. To change those settings, follow the instructions in [Modify a DB instance in a DB cluster \(p. 249\)](#).

**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 \(p. 251\)](#).

**Note**

Some settings only apply to DB instances. To change those settings, follow the instructions in [Modify a DB instance in a DB cluster \(p. 249\)](#).

## 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 \(p. 251\)](#).

**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 \(p. 248\)](#).

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 \(p. 251\)](#).

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 \(p. 251\)](#).

**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 \(p. 248\)](#).

### 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 \(p. 251\)](#).

**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 \(p. 248\)](#).

## 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 v1 or Aurora Serverless v2 DB cluster. For information about these settings, see [Modifying an Aurora Serverless v1 DB cluster \(p. 1570\)](#) and [Managing Aurora Serverless v2 \(p. 1509\)](#).

Some settings aren't available for Aurora Serverless v1 and Aurora Serverless v2 because of their limitations. For more information, see [Limitations of Aurora Serverless v1 \(p. 1544\)](#) and [Requirements for Aurora Serverless v2 \(p. 1490\)](#).

Setting and description	Method	Scope	Downtime notes
<p><b>Auto minor version upgrade</b></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 <a href="#">Amazon Aurora PostgreSQL updates (p. 1413)</a> and <a href="#">Database engine updates for Amazon Aurora MySQL (p. 990)</a>. For more information about the <b>Auto minor version upgrade</b> setting for Aurora MySQL, see <a href="#">Enabling automatic upgrades between minor Aurora MySQL versions (p. 997)</a>.</p>	<p><b>Note</b></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, <a href="#">Modify a DB instance in a DB cluster (p. 249)</a>.</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>Using the AWS CLI, run <a href="#">modify-db-instance</a> and set the --auto-minor-version-upgrade --no-auto-minor-version-upgrade option.</p> <p>Using the RDS API, call <a href="#">ModifyDBInstance</a> and set the AutoMinorVersionUpgrade parameter.</p>		
<p><b>Backup retention period</b></p> <p>The number of days that automatic backups are retained. The minimum value is 1.</p> <p>For more information, see <a href="#">Backups (p. 369)</a>.</p>	<p>Using the AWS Management Console, <a href="#">Modifying the DB cluster by using the console, CLI, and API (p. 248)</a>.</p> <p>Using the AWS CLI, run <a href="#">modify-db-cluster</a> and set the --backup-retention-period option.</p> <p>Using the RDS API, call <a href="#">ModifyDBCluster</a> and set the BackupRetentionPeriod parameter.</p>	The entire DB cluster	An outage doesn't occur during this change.

Setting and description	Method	Scope	Downtime notes
<p><b>Backup window (Start time)</b></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 <a href="#">Backup window (p. 369)</a>.</p>	<p>Using the AWS Management Console, <a href="#">Modifying the DB cluster by using the console, CLI, and API (p. 248)</a>.</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><b>Capacity range</b></p> <p>The database capacity measured in Aurora Capacity Units (ACUs).</p> <p>For more information, see <a href="#">Setting the Aurora Serverless v2 capacity range for a cluster (p. 1510)</a>.</p>	<p>Using the AWS Management Console, <a href="#">Modify a DB instance in a DB cluster (p. 249)</a>.</p> <p>Using the AWS CLI, run <code>modify-db-instance</code> and set the <code>--serverless-v2-scaling-configuration</code> option.</p> <p>Using the RDS API, call <code>ModifyDBInstance</code> and set the <code>ServerlessV2ScalingConfiguration</code> parameter.</p>	The entire DB cluster.	An outage doesn't occur during this change.

Setting and description	Method	Scope	Downtime notes
<b>Certificate Authority</b>  The certificate that you want to use.	Using the AWS Management Console, <a href="#">Modify a DB instance in a DB cluster (p. 249)</a> .  Using the AWS CLI, run <code>modify-db-instance</code> and set the <code>--ca-certificate-identifier</code> option.  Using the RDS API, call <code>ModifyDBInstance</code> and set the <code>CACertificateIdentifier</code> parameter.	Only the specified DB instance	An outage occurs during this change. The DB instance is rebooted.
<b>Copy tags to snapshots</b>  Select to specify that tags defined for this DB cluster are copied to DB snapshots created from this DB cluster. For more information, see <a href="#">Tagging Amazon RDS resources (p. 352)</a> .	Using the AWS Management Console, <a href="#">Modifying the DB cluster by using the console, CLI, and API (p. 248)</a> .  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.
<b>Data API</b>  You can access Aurora Serverless v1 with web services-based applications, including AWS Lambda and AWS AppSync. This setting only applies to an Aurora Serverless v1 DB cluster.  For more information, see <a href="#">Using the Data API for Aurora Serverless v1 (p. 1581)</a> .	Using the AWS Management Console, <a href="#">Modifying the DB cluster by using the console, CLI, and API (p. 248)</a> .  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><b>Database authentication</b> The database authentication you want to use.</p> <p>For MySQL:</p> <ul style="list-style-type: none"> <li>Choose <b>Password authentication</b> to authenticate database users with database passwords only.</li> <li>Choose <b>Password and IAM database authentication</b> to authenticate database users with database passwords and user credentials through IAM users and roles. For more information, see <a href="#">IAM database authentication (p. 1683)</a>.</li> </ul> <p>For PostgreSQL:</p> <ul style="list-style-type: none"> <li>Choose <b>IAM database authentication</b> to authenticate database users with database passwords and user credentials through IAM users and roles. For more information, see <a href="#">IAM database authentication (p. 1683)</a>.</li> <li>Choose <b>Kerberos authentication</b> to authenticate database passwords and user credentials using Kerberos authentication. For more information, see <a href="#">Using Kerberos authentication with Aurora PostgreSQL (p. 1035)</a>.</li> </ul>	<p>Using the AWS Management Console, <a href="#">Modifying the DB cluster by using the console, CLI, and API (p. 248)</a>.</p> <p>Using the AWS CLI, run <code>modify-db-cluster</code> and set the following options:</p> <ul style="list-style-type: none"> <li>For IAM authentication, set the <code>--enable-iam-database-authentication --no-enable-iam-database-authentication</code> option.</li> <li>For Kerberos authentication, set the <code>--domain</code> and <code>--domain-iam-role-name</code> options.</li> </ul> <p>Using the RDS API, call <code>ModifyDBCluster</code> and set the following parameters:</p> <ul style="list-style-type: none"> <li>For IAM authentication, set the <code>EnableIAMDatabaseAuthentication</code> parameter.</li> <li>For Kerberos authentication, set the <code>Domain</code> and <code>DomainIAMRoleName</code> parameters.</li> </ul>	The entire DB cluster	An outage doesn't occur during this change.

<b>Setting and description</b>	<b>Method</b>	<b>Scope</b>	<b>Downtime notes</b>
<b>Database port</b>  The port that you want to use to access the DB cluster.	<p>Using the AWS Management Console, <a href="#">Modifying the DB cluster by using the console, CLI, and API (p. 248)</a>.</p> <p>Using the AWS CLI, run <code>modify-db-cluster</code> and set the <code>--port</code> option.</p> <p>Using the RDS API, call <code>ModifyDBCluster</code> and set the <code>Port</code> parameter.</p>	The entire DB cluster	An outage occurs during this change. All of the DB instances in the DB cluster are rebooted immediately.
<b>DB cluster identifier</b>  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.	<p>Using the AWS Management Console, <a href="#">Modifying the DB cluster by using the console, CLI, and API (p. 248)</a>.</p> <p>Using the AWS CLI, run <code>modify-db-cluster</code> and set the <code>--new-db-cluster-identifier</code> option.</p> <p>Using the RDS API, call <code>ModifyDBCluster</code> and set the <code>NewDBClusterIdentifier</code> parameter.</p>	The entire DB cluster	An outage doesn't occur during this change.
<b>DB cluster parameter group</b>  The DB cluster parameter group that you want associated with the DB cluster.  For more information, see <a href="#">Working with parameter groups (p. 215)</a> .	<p>Using the AWS Management Console, <a href="#">Modifying the DB cluster by using the console, CLI, and API (p. 248)</a>.</p> <p>Using the AWS CLI, run <code>modify-db-cluster</code> and set the <code>--db-cluster-parameter-group-name</code> option.</p> <p>Using the RDS API, call <code>ModifyDBCluster</code> and set the <code>DBClusterParameterGroupName</code> parameter.</p>	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.

<b>Setting and description</b>	<b>Method</b>	<b>Scope</b>	<b>Downtime notes</b>
<b>DB instance class</b>  The DB instance class that you want to use.  For more information, see <a href="#">Aurora DB instance classes (p. 56)</a> .	Using the AWS Management Console, <a href="#">Modify a DB instance in a DB cluster (p. 249)</a> .  Using the AWS CLI, run <code>modify-db-instance</code> and set the <code>--db-instance-class</code> option.  Using the RDS API, call <a href="#">ModifyDBInstance</a> and set the <code>DBInstanceClass</code> parameter.	Only the specified DB instance	An outage occurs during this change.
<b>DB instance identifier</b>  The DB instance identifier. This value is stored as a lowercase string.	Using the AWS Management Console, <a href="#">Modify a DB instance in a DB cluster (p. 249)</a> .  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 <a href="#">ModifyDBInstance</a> 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
<p><b>DB parameter group</b></p> <p>The DB parameter group that you want associated with the DB instance.</p> <p>For more information, see <a href="#">Working with parameter groups (p. 215)</a>.</p>	<p>Using the AWS Management Console, <a href="#">Modify a DB instance in a DB cluster (p. 249)</a>.</p> <p>Using the AWS CLI, run <code>modify-db-instance</code> and set the <code>--db-parameter-group-name</code> option.</p> <p>Using the RDS API, call <a href="#">ModifyDBInstance</a> and set the <code>DBParameterGroupName</code> parameter.</p>	Only the specified DB instance	<p>An outage doesn't occur during this change.</p> <p>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.</p> <p>For more information, see <a href="#">Working with parameter groups (p. 215)</a> and <a href="#">Rebooting an Amazon Aurora DB cluster or Amazon Aurora DB instance (p. 329)</a>.</p>
<p><b>Deletion protection</b></p> <p><b>Enable deletion protection</b> to prevent your DB cluster from being deleted. For more information, see <a href="#">Deletion protection for Aurora clusters (p. 349)</a>.</p>	<p>Using the AWS Management Console, <a href="#">Modifying the DB cluster by using the console, CLI, and API (p. 248)</a>.</p> <p>Using the AWS CLI, run <code>modify-db-cluster</code> and set the <code>--deletion-protection --no-deletion-protection</code> option.</p> <p>Using the RDS API, call <a href="#">ModifyDBCluster</a> and set the <code>DeletionProtection</code> parameter.</p>	The entire DB cluster	An outage doesn't occur during this change.

Setting and description	Method	Scope	Downtime notes
<p><b>Engine version</b></p> <p>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.</p>	<p>Using the AWS Management Console, <a href="#">Modifying the DB cluster by using the console, CLI, and API (p. 248)</a>.</p> <p>Using the AWS CLI, run <code>modify-db-cluster</code> and set the <code>--engine-version</code> option.</p> <p>Using the RDS API, call <a href="#"><code>ModifyDBCluster</code></a> and set the <code>EngineVersion</code> parameter.</p>	The entire DB cluster	An outage occurs during this change.
<p><b>Enhanced monitoring</b></p> <p><b>Enable enhanced monitoring</b> to enable gathering metrics in real time for the operating system that your DB instance runs on.</p> <p>For more information, see <a href="#">Monitoring OS metrics with Enhanced Monitoring (p. 518)</a>.</p>	<p>Using the AWS Management Console, <a href="#">Modify a DB instance in a DB cluster (p. 249)</a>.</p> <p>Using the AWS CLI, run <code>modify-db-instance</code> and set the <code>--monitoring-role-arn</code> and <code>--monitoring-interval</code> options.</p> <p>Using the RDS API, call <a href="#"><code>ModifyDBInstance</code></a> and set the <code>MonitoringRoleArn</code> and <code>MonitoringInterval</code> parameters.</p>	Only the specified DB instance	An outage doesn't occur during this change.

<b>Setting and description</b>	<b>Method</b>	<b>Scope</b>	<b>Downtime notes</b>
<b>Log exports</b>  Select the log types to publish to Amazon CloudWatch Logs.  For more information, see <a href="#">Aurora MySQL database log files (p. 604)</a> .	Using the AWS Management Console, <a href="#">Modifying the DB cluster by using the console, CLI, and API (p. 248)</a> .  Using the AWS CLI, run <code>modify-db-cluster</code> and set the <code>--cloudwatch-logs-export-configuration</code> option.  Using the RDS API, call <a href="#"><code>ModifyDBCluster</code></a> and set the <code>CloudwatchLogsExportConfiguration</code> parameter.	The entire DB cluster	An outage doesn't occur during this change.

Setting and description	Method	Scope	Downtime notes
<p><b>Maintenance window</b></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 <a href="#">The Amazon RDS maintenance window (p. 324)</a>.</p>	<p>To change the maintenance window for the DB cluster using the AWS Management Console, <a href="#">Modifying the DB cluster by using the console, CLI, and API (p. 248)</a>.</p> <p>To change the maintenance window for a DB instance using the AWS Management Console, <a href="#">Modify a DB instance in a DB cluster (p. 249)</a>.</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 <a href="#">ModifyDBCluster</a> and set the <code>PreferredMaintenanceWindow</code> parameter.</p> <p>To change the maintenance window for a DB instance using the RDS API, call <a href="#">ModifyDBInstance</a> 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
<p><b>Network type</b> The IP addressing protocols supported by the DB cluster.</p> <p><b>IPv4</b> to specify that resources can communicate with the DB cluster only over the IPv4 addressing protocol.</p> <p><b>Dual-stack mode</b> to specify that resources can communicate with the DB cluster over IPv4, IPv6, or both. Use dual-stack mode if you have any resources that must communicate with your DB cluster over the IPv6 addressing protocol. To use dual-stack mode, make sure at least two subnets spanning two Availability Zones that support both the IPv4 and IPv6 network protocol. Also, make sure you associate an IPv6 CIDR block with subnets in the DB subnet group you specify.</p> <p>For more information, see <a href="#">Amazon Aurora IP addressing (p. 1731)</a>.</p>	<p>Using the AWS Management Console, <a href="#">Modifying the DB cluster by using the console, CLI, and API (p. 248)</a>.</p> <p>Using the AWS CLI, run <code>modify-db-cluster</code> and set the <code>-network-type</code> option.</p> <p>Using the RDS API, call <code>ModifyDBCluster</code> and set the <code>NetworkType</code> parameter.</p>	The entire DB cluster	An outage doesn't occur during this change.

Setting and description	Method	Scope	Downtime notes
<b>New master password</b>  The password for your master user. The password must contain from 8 to 41 alphanumeric characters.	Using the AWS Management Console, <a href="#">Modifying the DB cluster by using the console, CLI, and API (p. 248)</a> .  Using the AWS CLI, run <code>modify-db-cluster</code> and set the <code>--master-user-password</code> option.  Using the RDS API, call <a href="#">ModifyDBCluster</a> and set the <code>MasterUserPassword</code> parameter.	The entire DB cluster	An outage doesn't occur during this change.
<b>Performance Insights</b>  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 <a href="#">Monitoring DB load with Performance Insights on Amazon Aurora (p. 461)</a> .	Using the AWS Management Console, <a href="#">Modify a DB instance in a DB cluster (p. 249)</a> .  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 <a href="#">ModifyDBInstance</a> 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
<b>Performance Insights AWS KMS key</b>  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 <a href="#">Turning Performance Insights on and off (p. 466)</a> .	Using the AWS Management Console, <a href="#">Modify a DB instance in a DB cluster (p. 249)</a> .  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 <a href="#"><code>ModifyDBInstance</code></a> and set the <code>PerformanceInsightsKMSKeyId</code> parameter.	Only the specified DB instance	An outage doesn't occur during this change.
<b>Performance Insights retention period</b>  The amount of time, in days, to retain Performance Insights data. The retention setting in the free tier is <b>Default (7 days)</b> . To retain your performance data for longer, specify 1–24 months. For more information about retention periods, see <a href="#">Pricing and data retention for Performance Insights (p. 465)</a> .  For more information, see <a href="#">Turning Performance Insights on and off (p. 466)</a> .	Using the AWS Management Console, <a href="#">Modify a DB instance in a DB cluster (p. 249)</a> .  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 <a href="#"><code>ModifyDBInstance</code></a> and set the <code>PerformanceInsightsRetentionPeriod</code> parameter.	Only the specified DB instance	An outage doesn't occur during this change.

<b>Setting and description</b>	<b>Method</b>	<b>Scope</b>	<b>Downtime notes</b>
<b>Promotion tier</b>  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 <a href="#">Fault tolerance for an Aurora DB cluster (p. 72)</a> .	Using the AWS Management Console, <a href="#">Modify a DB instance in a DB cluster (p. 249)</a> .  Using the AWS CLI, run <code>modify-db-instance</code> and set the <code>--promotion-tier</code> option.  Using the RDS API, call <a href="#"><code>ModifyDBInstance</code></a> 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><b>Public access</b></p> <p><b>Publicly accessible</b> 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><b>Not publicly accessible</b> to make the DB instance accessible only from inside the VPC.</p> <p>For more information, see <a href="#">Hiding a DB cluster in a VPC from the internet (p. 1735)</a>.</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 <a href="#">Can't connect to Amazon RDS DB instance (p. 1761)</a>.</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 <a href="#">Internetwork traffic privacy (p. 1652)</a>.</p>	<p>Using the AWS Management Console, <a href="#">Modify a DB instance in a DB cluster (p. 249)</a>.</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 <a href="#">ModifyDBInstance</a> 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
<p><b>Scaling configuration</b></p> <p>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.</p> <p>For information about Aurora Serverless v1, see <a href="#">Using Amazon Aurora Serverless v1 (p. 1543)</a>.</p>	<p>Using the AWS Management Console, <a href="#">Modifying the DB cluster by using the console, CLI, and API (p. 248)</a>.</p> <p>Using the AWS CLI, run <code>modify-db-cluster</code> and set the <code>--scaling-configuration</code> option.</p> <p>Using the RDS API, call <code>ModifyDBCluster</code> and set the <code>ScalingConfiguration</code> parameter.</p>	The entire DB cluster	An outage doesn't occur during this change.
<p><b>Security group</b></p> <p>The security group you want associated with the DB cluster.</p> <p>For more information, see <a href="#">Controlling access with security groups (p. 1721)</a>.</p>	<p>Using the AWS Management Console, <a href="#">Modifying the DB cluster by using the console, CLI, and API (p. 248)</a>.</p> <p>Using the AWS CLI, run <code>modify-db-cluster</code> and set the <code>--vpc-security-group-ids</code> option.</p> <p>Using the RDS API, call <code>ModifyDBCluster</code> and set the <code>VpcSecurityGroupIds</code> parameter.</p>	The entire DB cluster	An outage doesn't occur during this change.

Setting and description	Method	Scope	Downtime notes
<b>Target Backtrack window</b>  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, <a href="#">Modifying the DB cluster by using the console, CLI, and API (p. 248)</a> .  Using the AWS CLI, run <code>modify-db-cluster</code> and set the <code>--backtrack-window</code> option.  Using the RDS API, call <a href="#"><code>ModifyDBCluster</code></a> and set the <code>BacktrackWindow</code> parameter.	The entire DB cluster	An outage doesn't occur during this change.

## Settings that don't apply to Amazon Aurora DB clusters

The following settings in the AWS CLI command `modify-db-cluster` and the RDS API operation `ModifyDBCluster` don't apply to Amazon Aurora DB clusters.

**Note**

You can't use the AWS Management Console to modify these settings for Aurora DB clusters.

AWS CLI setting	RDS API setting
<code>--allocated-storage</code>	<code>AllocatedStorage</code>
<code>--auto-minor-version-upgrade   --no-auto-minor-version-upgrade</code>	<code>AutoMinorVersionUpgrade</code>
<code>--db-cluster-instance-class</code>	<code>DBClusterInstanceClass</code>
<code>--enable-performance-insights   --no-enable-performance-insights</code>	<code>EnablePerformanceInsights</code>
<code>--iops</code>	<code>Iops</code>
<code>--monitoring-interval</code>	<code>MonitoringInterval</code>
<code>--monitoring-role-arn</code>	<code>MonitoringRoleArn</code>
<code>--option-group-name</code>	<code>OptionGroupName</code>
<code>--performance-insights-kms-key-id</code>	<code>PerformanceInsightsKMSKeyId</code>
<code>--performance-insights-retention-period</code>	<code>PerformanceInsightsRetentionPeriod</code>
<code>--storage-type</code>	<code>StorageType</code>

## Settings that don't apply to Amazon Aurora DB instances

The following settings in the AWS CLI command [modify-db-instance](#) and the RDS API operation [ModifyDBInstance](#) don't apply to Amazon Aurora DB instances.

**Note**

You can't use the AWS Management Console to modify these settings for Aurora DB instances.

AWS CLI setting	RDS API setting
--allocated-storage	AllocatedStorage
--allow-major-version-upgrade  --no-allow-major-version-upgrade	AllowMajorVersionUpgrade
--copy-tags-to-snapshot  --no-copy-tags-to-snapshot	CopyTagsToSnapshot
--domain	Domain
--db-security-groups	DBSecurityGroups
--db-subnet-group-name	DBSubnetGroupName
--domain-iam-role-name	DomainIAMRoleName
--multi-az  --no-multi-az	MultiAZ
--iops	Iops
--license-model	LicenseModel
--network-type	NetworkType
--option-group-name	OptionGroupName
--processor-features	ProcessorFeatures
--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 \(p. 73\)](#).

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.

However, you can add reader instances to Aurora Serverless v2 DB clusters. For more information, see [Adding an Aurora Serverless v2 reader \(p. 1517\)](#).

- 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 \(p. 350\)](#).

## 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 \(p. 73\)](#).

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
<b>Availability zone</b>	Determine if you want to specify a particular Availability Zone. The list includes only those Availability Zones that are mapped to the DB subnet group that you chose when you created the DB cluster. For more information about Availability Zones, see <a href="#">Regions and Availability Zones (p. 11)</a> .
<b>Publicly accessible</b>	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 <a href="#">Hiding a DB cluster in a VPC from the internet (p. 1735)</a> .
<b>Encryption</b>	Select Enable encryption to enable encryption at rest for this Aurora Replica. For more information, see <a href="#">Encrypting Amazon Aurora resources (p. 1638)</a> .
<b>DB instance class</b>	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 <a href="#">Aurora DB instance classes (p. 56)</a> .
<b>Aurora replica source</b>	Select the identifier of the primary instance to create an Aurora Replica for.
<b>DB instance identifier</b>	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 <b>aurora-read-instance1</b> .
<b>Priority</b>	Choose a failover priority for the instance. If you don't select a value, the default is <b>tier-1</b> . This priority determines the order in which Aurora Replicas are promoted when recovering from a primary instance failure. For more information, see <a href="#">Fault tolerance for an Aurora DB cluster (p. 72)</a> .
<b>Database port</b>	The port for an Aurora Replica is the same as the port for the DB cluster.
<b>DB parameter group</b>	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 <a href="#">Working with parameter groups (p. 215)</a> .
<b>Performance Insights</b>	The <b>Turn on Performance Insights</b> check box is selected by default. The value isn't inherited from the writer instance. For more information, see <a href="#">Monitoring DB load with Performance Insights on Amazon Aurora (p. 461)</a> .
<b>Enhanced monitoring</b>	Choose <b>Enable enhanced monitoring</b> to enable gathering metrics in real time for the operating system that your DB cluster runs on. For more information, see <a href="#">Monitoring OS metrics with Enhanced Monitoring (p. 518)</a> .

For this option	Do this
<b>Monitoring Role</b>	Only available if <b>Enhanced Monitoring</b> is set to <b>Enable enhanced monitoring</b> . Choose the IAM role that you created to permit Amazon RDS to communicate with Amazon CloudWatch Logs for you, or choose <b>Default</b> to have RDS create a role for you named <code>rds-monitoring-role</code> . For more information, see <a href="#">Monitoring OS metrics with Enhanced Monitoring (p. 518)</a> .
<b>Granularity</b>	Only available if <b>Enhanced Monitoring</b> is set to <b>Enable enhanced monitoring</b> . Set the interval, in seconds, between when metrics are collected for your DB cluster.
<b>Auto minor version upgrade</b>	Select <b>Enable auto minor version upgrade</b> if you want to enable your Aurora DB cluster to receive minor DB Engine version upgrades automatically when they become available.  The <b>Auto minor version upgrade</b> 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.  For more information about engine updates for Aurora PostgreSQL, see <a href="#">Amazon Aurora PostgreSQL updates (p. 1413)</a> .  For more information about engine updates for Aurora MySQL, see <a href="#">Database engine updates for Amazon Aurora MySQL (p. 990)</a> .

- 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 \(p. 274\)](#)
- [Instance scaling \(p. 278\)](#)
- [Read scaling \(p. 278\)](#)
- [Managing connections \(p. 278\)](#)
- [Managing query execution plans \(p. 279\)](#)

## 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 \(p. 67\)](#). For details about the maximum size for a specific version, see [Amazon Aurora size limits \(p. 1759\)](#).

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 14 and 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*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 <a href="#">Scaling Aurora MySQL DB instances (p. 721)</a>
Amazon Aurora PostgreSQL	See <a href="#">Scaling Aurora PostgreSQL DB instances (p. 1143)</a>

## 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 \(p. 270\)](#).

## 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 <a href="#">Maximum connections to an Aurora MySQL DB instance (p. 722)</a>
Amazon Aurora PostgreSQL	See <a href="#">Maximum connections to an Aurora PostgreSQL DB instance (p. 1144)</a>

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 \(p. 1430\)](#).

## 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 \(p. 1290\)](#).

# Cloning a volume for an Amazon Aurora DB cluster

By using Aurora cloning, you can create a new cluster that uses the same Aurora cluster volume and has the same data as the original. The process is designed to be fast and cost-effective. 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.
- 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.
- Clusters with Aurora Serverless v2 instances follow the same rules as provisioned 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 \(p. 1543\)](#).

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 \(p. 293\)](#).

Cross-account cloning currently doesn't support cloning Aurora Serverless v1 DB clusters. For more information, see [Limitations of cross-account cloning \(p. 293\)](#).

## Topics

- [Overview of Aurora cloning \(p. 280\)](#)
- [Limitations of Aurora cloning \(p. 281\)](#)
- [How Aurora cloning works \(p. 281\)](#)
- [Creating an Amazon Aurora clone \(p. 284\)](#)
- [Cross-account cloning with AWS RAM and Amazon Aurora \(p. 293\)](#)

## 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 \(p. 281\)](#).

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 create as many clones as you want, up to the maximum number of DB clusters allowed in the AWS Region. However, after you create 15 clones, the next clone is a full copy. The cloning operation acts like a point-in-time recovery.
- You can't create a clone in a different AWS Region from 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 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 \(p. 67\)](#).

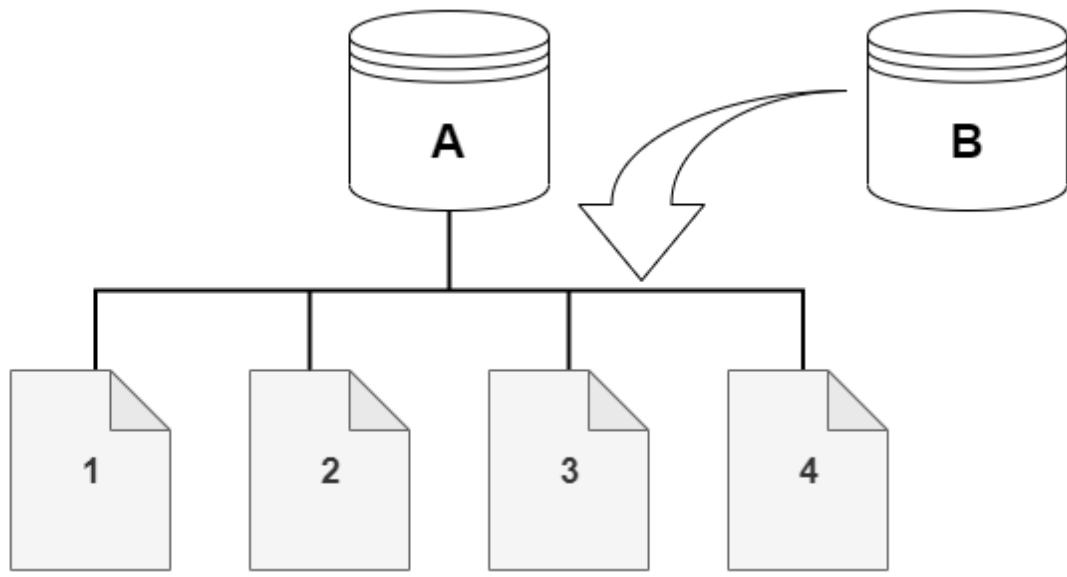
### Topics

- [Understanding the copy-on-write protocol \(p. 281\)](#)
- [Deleting a source cluster volume \(p. 284\)](#)

## Understanding the copy-on-write protocol

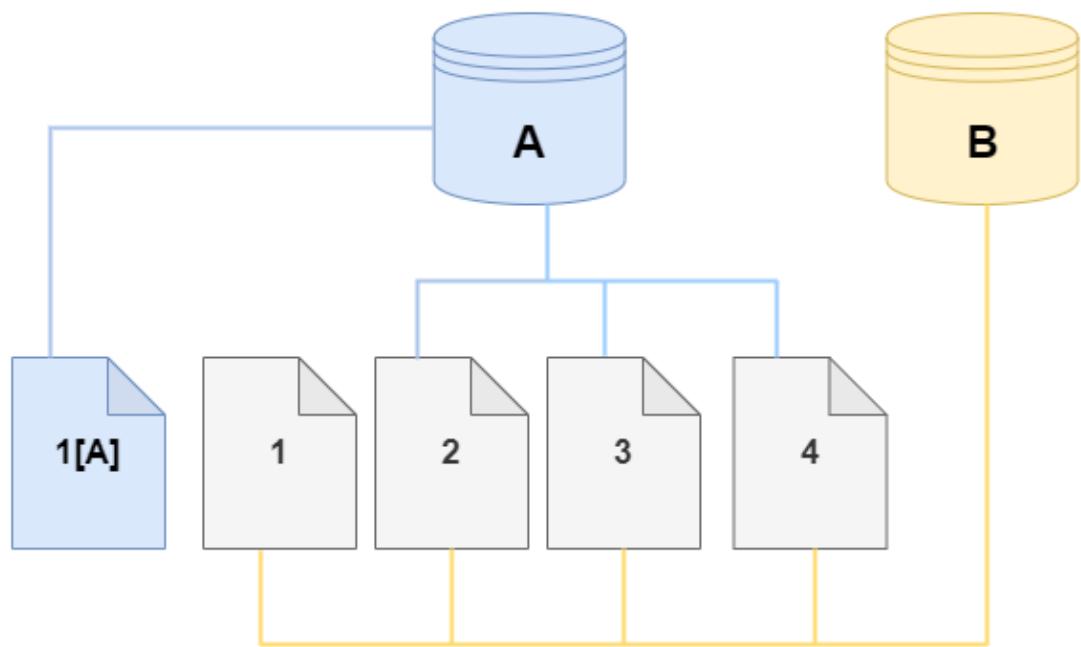
An Aurora DB cluster stores data in pages in the underlying Aurora storage volume.

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.

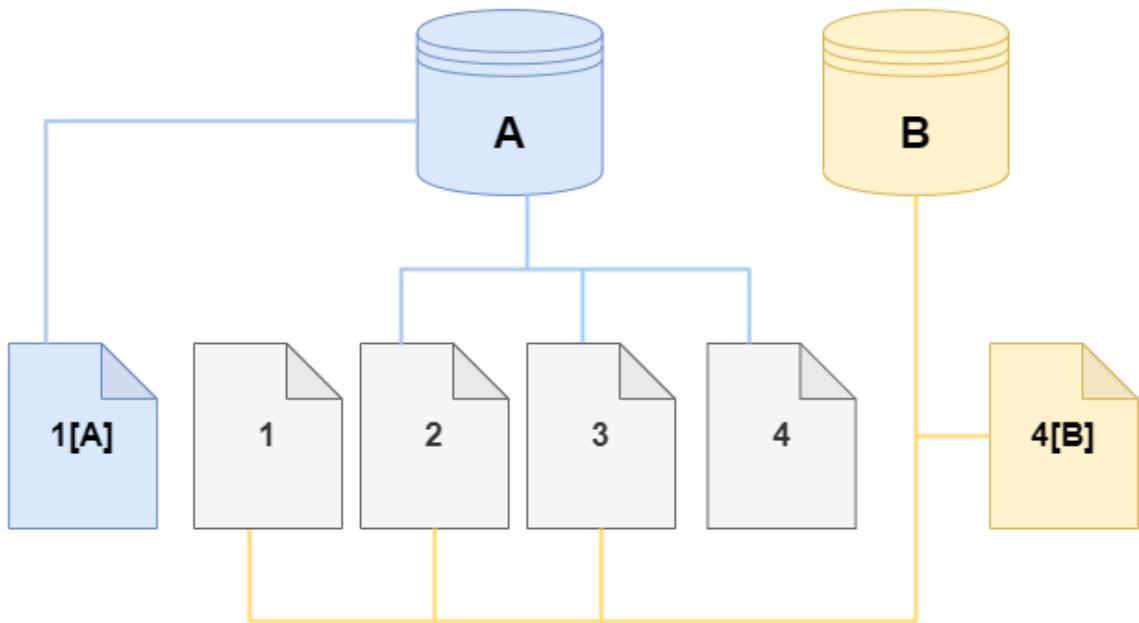


When the clone is created, no additional storage is usually needed. The copy-on-write protocol uses the same segment on the physical storage media as the source segment. Additional storage is required only if the capacity of the source segment isn't sufficient for the entire clone segment. If that's 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 \(p. 293\)](#).

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

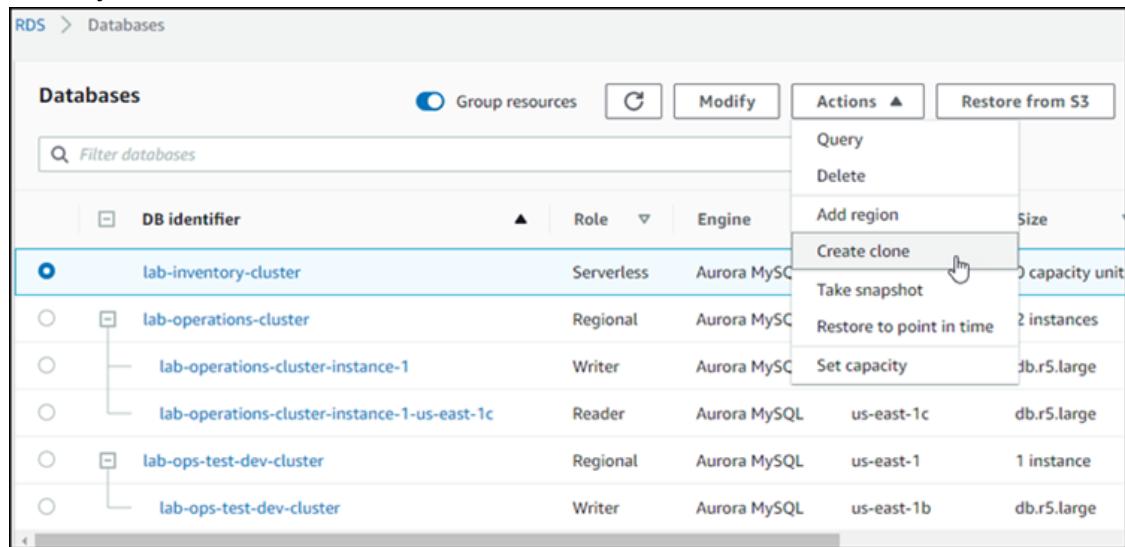
The following procedure describes how to clone an Aurora DB cluster using the AWS Management Console.

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 \(p. 293\)](#) 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 <a href="#">Info</a>	Maximum Aurora capacity unit <a href="#">Info</a>
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 \(p. 127\)](#).
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 \(p. 288\)](#)
- [Checking the status and getting clone details \(p. 291\)](#)
- [Creating the Aurora DB instance for your clone \(p. 291\)](#)
- [Parameters to use for cloning \(p. 292\)](#)

### 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 \(p. 291\)](#).

## 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 \(p. 291\)](#) 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"> <li>Use <code>provisioned</code> to create a provisioned Aurora DB cluster clone from an Aurora Serverless DB cluster.</li> <li>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</li> </ul>
--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.
--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 \(p. 280\)](#).

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 \(p. 293\)](#)
- [Allowing other AWS accounts to clone your cluster \(p. 294\)](#)
- [Cloning a cluster that is owned by another AWS account \(p. 296\)](#)

## 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 create a maximum of 15 cross-account clones from any single Aurora DB cluster.
- Each of the 15 cross-account 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.
- After you clone a cluster, the original cluster and its clone are considered to be the same for purposes of enforcing limits on cross-account clones. You can't create cross-account clones of both the original cluster and the cloned cluster within the same AWS account. The total number of cross-account clones for the original cluster and any of its clones can't exceed 15.
- 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 \(p. 294\)](#)
- [Checking if a cluster that you own is shared with other AWS accounts \(p. 296\)](#)

## 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 \(p. 375\)](#). 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 \(p. 417\)](#). 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 \(p. 297\)](#)
- [Accepting invitations to share clusters owned by other AWS accounts \(p. 297\)](#)
- [Cloning an Aurora cluster that is owned by another AWS account \(p. 298\)](#)
- [Checking if a DB cluster is a cross-account clone \(p. 301\)](#)

## 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 \(p. 285\)](#) 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-abcdefg",  
            "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 \(p. 304\)](#)
- [Integrating AWS services with Amazon Aurora PostgreSQL \(p. 304\)](#)
- [Using Amazon Aurora Auto Scaling with Aurora replicas \(p. 305\)](#)
- [Using machine learning \(ML\) capabilities with Amazon Aurora \(p. 320\)](#)

## 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 \(p. 305\)](#).

For more information about integrating Aurora MySQL with other AWS services, see [Integrating Amazon Aurora MySQL with other AWS services \(p. 890\)](#).

## 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 \(p. 305\)](#).

For more information about integrating Aurora PostgreSQL with other AWS services, see [Integrating Amazon Aurora PostgreSQL with other AWS services \(p. 1230\)](#).

# 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 \(p. 305\)](#)
- [Aurora Auto Scaling policies \(p. 306\)](#)
- [Adding a scaling policy \(p. 307\)](#)
- [Editing a scaling policy \(p. 316\)](#)
- [Deleting a scaling policy \(p. 318\)](#)
- [DB instance IDs and tagging \(p. 319\)](#)

## 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 \(p. 127\)](#).

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 \(p. 56\)](#). 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 \(p. 72\)](#).

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 AWS JDBC Driver for MySQL. For more information, see [Connecting to an Amazon Aurora DB cluster \(p. 207\)](#).

### 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 \(p. 307\)](#)
- [Adding a scaling policy using the AWS CLI or the Application Auto Scaling API \(p. 310\)](#)

## 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 \(p. 306\)](#).
- `--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 \(p. 306\)](#).

### 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 \(p. 306\)](#).
- **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 \(p. 306\)](#).

## 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 \(p. 312\)](#)
- [Using a custom metric \(p. 313\)](#)
- [Using cooldown periods \(p. 313\)](#)
- [Disabling scale-in activity \(p. 314\)](#)

### 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 \(p. 525\)](#).

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 \(p. 314\)](#).

## 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 \(p. 352\)](#).

## 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 \(p. 927\)](#)
- [Using machine learning \(ML\) with Aurora PostgreSQL \(p. 1272\)](#)

# 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 \(p. 248\)](#).

## 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 the Amazon RDS console interface. At the top, there are four buttons: 'Modify' (blue), 'Actions ▾' (grey), 'Restore from S3' (grey), and 'Create database' (orange). Below these are navigation controls: a search bar, a page number '1', and icons for back, forward, and refresh. The main area is a table with the following columns: '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 'Maintenance & backups' tab selected in the top navigation bar. Below it, the 'Pending maintenance' section displays a single item: 'Automatic minor version upgrade to postgres 9.6.11'. This item is listed under the 'Pending maintenance' column. At the bottom of this section, there are three buttons: 'C' (cancel), 'Apply now', and 'Apply at next maintenance window'. A search bar labeled 'Filter pending maintenance' and a page navigation area with a single page indicator are also visible.

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)

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 \(p. 324\)](#).

For information about updates to Amazon Aurora engines and instructions for upgrading and patching them, see [Database engine updates for Amazon Aurora MySQL \(p. 990\)](#) and [Amazon Aurora PostgreSQL updates \(p. 1413\)](#).

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 \(p. 251\)](#). 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 \(p. 1413\)](#).

For more information about the **Auto minor version upgrade** setting for Aurora MySQL, see [Enabling automatic upgrades between minor Aurora MySQL versions \(p. 997\)](#). For general information about engine updates for Aurora MySQL, see [Database engine updates for Amazon Aurora MySQL \(p. 990\)](#).

## 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 \(p. 993\)](#).

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 parameter groups \(p. 215\)](#).

#### Topics

- [Rebooting a DB instance within an Aurora cluster \(p. 329\)](#)
- [Rebooting an Aurora cluster \(Aurora PostgreSQL and Aurora MySQL before version 2.10\) \(p. 330\)](#)
- [Rebooting an Aurora MySQL cluster \(version 2.10 and higher\) \(p. 330\)](#)
- [Checking uptime for Aurora clusters and instances \(p. 331\)](#)
- [Examples of Aurora reboot operations \(p. 333\)](#)

## 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 \(p. 329\)](#).

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 \(version 2.10 and higher\) \(p. 330\)](#).

## 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 \(p. 329\)](#).

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 \(Aurora PostgreSQL and Aurora MySQL before version 2.10\) \(p. 330\)](#) 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 parameter groups \(p. 215\)](#). 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 \(p. 950\)](#).

**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 \(p. 334\)](#)
- [Rebooting a single reader instance \(p. 334\)](#)
- [Rebooting the writer instance \(p. 335\)](#)
- [Rebooting the writer and readers independently \(p. 336\)](#)
- [Applying a cluster parameter change to an Aurora MySQL version 2.10 cluster \(p. 339\)](#)

## 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 \(p. 331\)](#).

```
$ 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[*]]" \
--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 \(p. 345\)](#)
- [Deletion protection for Aurora clusters \(p. 349\)](#)
- [Deleting a stopped Aurora cluster \(p. 350\)](#)
- [Deleting Aurora MySQL clusters that are read replicas \(p. 350\)](#)
- [The final snapshot when deleting a cluster \(p. 350\)](#)
- [Deleting a DB instance from an Aurora DB cluster \(p. 350\)](#)

## 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 \(p. 350\)](#). 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 \(p. 350\)](#).

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 \(p. 346\)](#)
- [Deleting an Aurora cluster with a single DB instance \(p. 346\)](#)
- [Deleting an Aurora cluster with multiple DB instances \(p. 347\)](#)

## 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 \(p. 350\)](#).
- For the procedure to delete the writer DB instance in an Aurora cluster, see [Deleting an Aurora cluster with a single DB instance \(p. 346\)](#).
- For the procedure to delete an empty Aurora cluster, see [Deleting an empty Aurora cluster \(p. 346\)](#).

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 \(p. 248\)](#).

**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 \(p. 246\)](#).

## 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 \(p. 831\)](#).

## 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.

**Note**

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 during the grace period. When the grace period ends, the Aurora Replica is shut down and deleted.

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 \(p. 346\)](#) and [Deleting an empty Aurora cluster \(p. 346\)](#).

**Note**

You can't delete a DB cluster when deletion protection is enabled for it. For more information, see [Deletion protection for Aurora clusters \(p. 349\)](#).

You can disable deletion protection by modifying the DB cluster. For more information, see [Modifying an Amazon Aurora DB cluster \(p. 248\)](#).

## 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 \(p. 1642\)](#).

# 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 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 \(p. 352\)](#)
- [Using tags for access control with IAM \(p. 353\)](#)
- [Using tags to produce detailed billing reports \(p. 353\)](#)
- [Adding, listing, and removing tags \(p. 354\)](#)
- [Using the AWS Tag Editor \(p. 356\)](#)
- [Copying tags to DB cluster snapshots \(p. 356\)](#)
- [Tutorial: Use tags to specify which Aurora DB clusters to stop \(p. 357\)](#)

## 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: `^([\\p{L}\\p{Z}\\p{N}_:=+\\-@]*$)`).
- The tag value is an optional string value of the tag. The string value can be from 1 to 256 Unicode characters in length and cannot be prefixed with `aws:`. The string can contain only the set of Unicode letters, digits, white-space, `'_`, `'!`, `'.'`, `'/`, `'=`, `'+'`, `'-`, `'@'` (Java regex: `^([\\p{L}\\p{Z}\\p{N}_:=+\\-@]*$)`).

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 \(p. 360\)](#).

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 for Amazon Aurora \(p. 1653\)](#).

## 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 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.

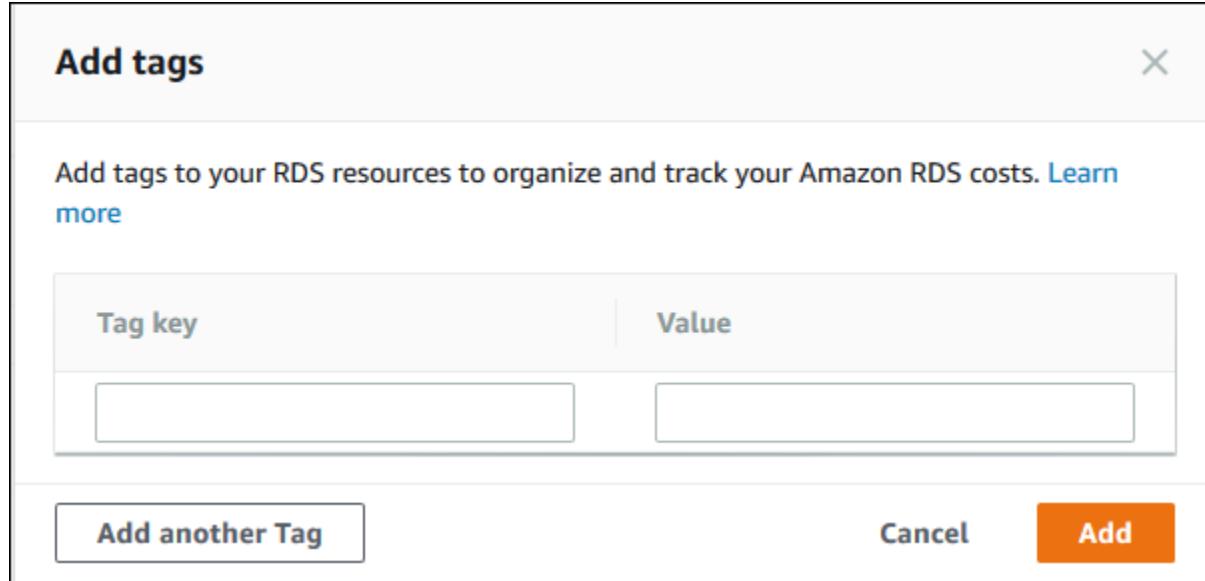
#### To add a tag to a DB instance

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**.

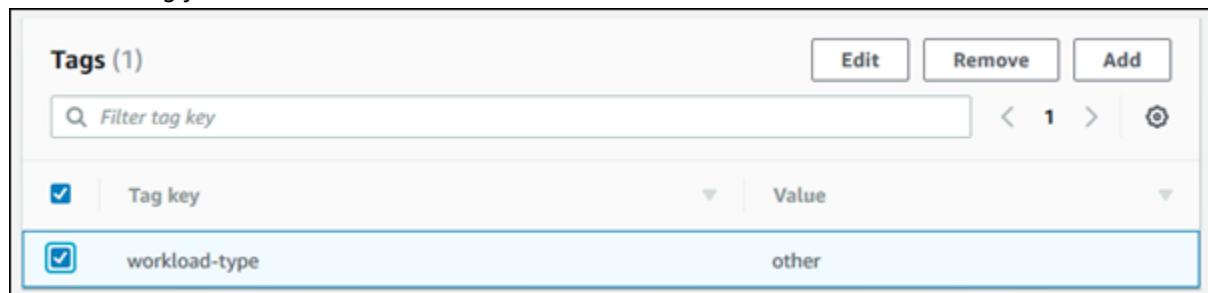
### To delete a tag from a DB instance

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 \(p. 360\)](#).

### 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 \(p. 360\)](#).

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 project/Trinity and project/Xanadu.
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 project/Trinity and cost-center/Trinity.

## 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	HTTPS	
		rds-fips.us-east-2.api.aws	HTTPS	
		rds.us-east-2.api.aws	HTTPS	
		rds-fips.us-east-2.amazonaws.com	HTTPS	
US East (N. Virginia)	us-east-1	rds.us-east-1.amazonaws.com	HTTPS	
		rds-fips.us-east-1.api.aws	HTTPS	
		rds-fips.us-east-1.amazonaws.com	HTTPS	
		rds.us-east-1.api.aws	HTTPS	
US West (N. California)	us-west-1	rds.us-west-1.amazonaws.com	HTTPS	
		rds.us-west-1.api.aws	HTTPS	
		rds-fips.us-west-1.amazonaws.com	HTTPS	
		rds-fips.us-west-1.api.aws	HTTPS	
US West (Oregon)	us-west-2	rds.us-west-2.amazonaws.com	HTTPS	
		rds-fips.us-west-2.amazonaws.com	HTTPS	
		rds.us-west-2.api.aws	HTTPS	
		rds-fips.us-west-2.api.aws	HTTPS	
Africa (Cape Town)	af-south-1	rds.af-south-1.amazonaws.com	HTTPS	
		rds.af-south-1.api.aws	HTTPS	
Asia Pacific	ap-east-1	rds.ap-east-1.amazonaws.com	HTTPS	
		rds.ap-east-1.api.aws	HTTPS	

<b>Region Name</b>	<b>Region</b>	<b>Endpoint</b>	<b>Protocol</b>	
(Hong Kong)				
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 rds.ap-south-1.api.aws	HTTPS HTTPS	
Asia Pacific (Osaka)	ap-northeast-3	rds.ap-northeast-3.amazonaws.com rds.ap-northeast-3.api.aws	HTTPS HTTPS	
Asia Pacific (Seoul)	ap-northeast-2	rds.ap-northeast-2.amazonaws.com rds.ap-northeast-2.api.aws	HTTPS HTTPS	
Asia Pacific (Singapore)	ap-southeast-1	rds.ap-southeast-1.amazonaws.com rds.ap-southeast-1.api.aws	HTTPS HTTPS	
Asia Pacific (Sydney)	ap-southeast-2	rds.ap-southeast-2.amazonaws.com rds.ap-southeast-2.api.aws	HTTPS HTTPS	
Asia Pacific (Tokyo)	ap-northeast-1	rds.ap-northeast-1.amazonaws.com rds.ap-northeast-1.api.aws	HTTPS HTTPS	
Canada (Central)	ca-central-1	rds.ca-central-1.amazonaws.com rds.ca-central-1.api.aws rds-fips.ca-central-1.api.aws rds-fips.ca-central-1.amazonaws.com	HTTPS HTTPS HTTPS HTTPS	
Europe (Frankfurt)	eu-central-1	rds.eu-central-1.amazonaws.com rds.eu-central-1.api.aws	HTTPS HTTPS	
Europe (Ireland)	eu-west-1	rds.eu-west-1.amazonaws.com rds.eu-west-1.api.aws	HTTPS HTTPS	
Europe (London)	eu-west-2	rds.eu-west-2.amazonaws.com rds.eu-west-2.api.aws	HTTPS HTTPS	
Europe (Milan)	eu-south-1	rds.eu-south-1.amazonaws.com rds.eu-south-1.api.aws	HTTPS HTTPS	
Europe (Paris)	eu-west-3	rds.eu-west-3.amazonaws.com rds.eu-west-3.api.aws	HTTPS HTTPS	

Region Name	Region	Endpoint	Protocol	
Europe (Stockholm)	eu-north-1	rds.eu-north-1.amazonaws.com rds.eu-north-1.api.aws	HTTPS HTTPS	
Middle East (Bahrain)	me-south-1	rds.me-south-1.amazonaws.com rds.me-south-1.api.aws	HTTPS HTTPS	
South America (São Paulo)	sa-east-1	rds.sa-east-1.amazonaws.com rds.sa-east-1.api.aws	HTTPS 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	

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	arn:aws:rds:<region>:<account>:db:<name>  For example:  <code>arn:aws:rds:us-east-2:123456789012:db:my-mysql-instance-1</code>
DB cluster	arn:aws:rds:<region>:<account>:cluster:<name>  For example:  <code>arn:aws:rds:us-east-2:123456789012:cluster:my-aurora-cluster-1</code>
Event subscription	arn:aws:rds:<region>:<account>:es:<name>  For example:  <code>arn:aws:rds:us-east-2:123456789012:es:my-subscription</code>
DB parameter group	arn:aws:rds:<region>:<account>:pg:<name>  For example:  <code>arn:aws:rds:us-east-2:123456789012:pg:my-param-enable-logs</code>
DB cluster parameter group	arn:aws:rds:<region>:<account>:cluster-pg:<name>

Resource type	ARN format
	For example: <div style="border: 1px solid black; padding: 5px; background-color: #f0f0f0;"> <code>arn:aws:rds:us-east-2:123456789012:cluster-pg:my-cluster-param-timezone</code> </div>
Reserved DB instance	<code>arn:aws:rds:&lt;region&gt;:&lt;account&gt;:ri:&lt;name&gt;</code> For example: <div style="border: 1px solid black; padding: 5px; background-color: #f0f0f0;"> <code>arn:aws:rds:us-east-2:123456789012:ri:my-reserved-postgresql</code> </div>
DB security group	<code>arn:aws:rds:&lt;region&gt;:&lt;account&gt;:secgrp:&lt;name&gt;</code> For example: <div style="border: 1px solid black; padding: 5px; background-color: #f0f0f0;"> <code>arn:aws:rds:us-east-2:123456789012:secgrp:my-public</code> </div>
Automated DB snapshot	<code>arn:aws:rds:&lt;region&gt;:&lt;account&gt;:snapshot:rds:&lt;name&gt;</code> For example: <div style="border: 1px solid black; padding: 5px; background-color: #f0f0f0;"> <code>arn:aws:rds:us-east-2:123456789012:snapshot:rds:my-mysql-db-2019-07-22-07-23</code> </div>
Automated DB cluster snapshot	<code>arn:aws:rds:&lt;region&gt;:&lt;account&gt;:cluster-snapshot:rds:&lt;name&gt;</code> For example: <div style="border: 1px solid black; padding: 5px; background-color: #f0f0f0;"> <code>arn:aws:rds:us-east-2:123456789012:cluster-snapshot:rds:my-aurora-cluster-2019-07-22-16-16</code> </div>
Manual DB snapshot	<code>arn:aws:rds:&lt;region&gt;:&lt;account&gt;:snapshot:&lt;name&gt;</code> For example: <div style="border: 1px solid black; padding: 5px; background-color: #f0f0f0;"> <code>arn:aws:rds:us-east-2:123456789012:snapshot:my-mysql-db-snap</code> </div>
Manual DB cluster snapshot	<code>arn:aws:rds:&lt;region&gt;:&lt;account&gt;:cluster-snapshot:&lt;name&gt;</code> For example: <div style="border: 1px solid black; padding: 5px; background-color: #f0f0f0;"> <code>arn:aws:rds:us-east-2:123456789012:cluster-snapshot:my-aurora-cluster-snap</code> </div>
DB subnet group	<code>arn:aws:rds:&lt;region&gt;:&lt;account&gt;:subgrp:&lt;name&gt;</code> For example: <div style="border: 1px solid black; padding: 5px; background-color: #f0f0f0;"> <code>arn:aws:rds:us-east-2:123456789012:subgrp:my-subnet-10</code> </div>

## 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 DB instance details are listed under the "Instance" section. The "Configuration" section is expanded, showing various configuration parameters. At the bottom of the configuration list, there is an "ARN" field containing the value "arn:aws:rds:us-west-2:XXXXXXXXXX:db:oracle-instance1", which is also circled in red.

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:XXXXXXXXXX:db:oracle-instance1
Resource id	XXXXXXXXXX

## 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
<code>describe-event-subscriptions</code>	<code>EventSubscriptionArn</code>
<code>describe-certificates</code>	<code>CertificateArn</code>
<code>describe-db-parameter-groups</code>	<code>DBParameterGroupArn</code>
<code>describe-db-cluster-parameter-groups</code>	<code>DBClusterParameterGroupArn</code>
<code>describe-db-instances</code>	<code>DBInstanceArn</code>
<code>describe-db-security-groups</code>	<code>DBSecurityGroupArn</code>
<code>describe-db-snapshots</code>	<code>DBSnapshotArn</code>
<code>describe-events</code>	<code>SourceArn</code>
<code>describe-reserved-db-instances</code>	<code>ReservedDBInstanceArn</code>
<code>describe-db-subnet-groups</code>	<code>DBSubnetGroupArn</code>
<code>describe-db-clusters</code>	<code>DBClusterArn</code>
<code>describe-db-cluster-snapshots</code>	<code>DBClusterSnapshotArn</code>

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
<a href="#">DescribeEventSubscriptions</a>	EventSubscriptionArn
<a href="#">DescribeCertificates</a>	CertificateArn
<a href="#">DescribeDBParameterGroups</a>	DBParameterGroupArn
<a href="#">DescribeDBClusterParameterGroups</a>	DBClusterParameterGroupArn
<a href="#">DescribeDBInstances</a>	DBInstanceArn
<a href="#">DescribeDBSecurityGroups</a>	DBSecurityGroupArn
<a href="#">DescribeDBSnapshots</a>	DBSnapshotArn
<a href="#">DescribeEvents</a>	SourceArn
<a href="#">DescribeReservedDBInstances</a>	ReservedDBInstanceArn
<a href="#">DescribeDBSubnetGroups</a>	DBSubnetGroupArn
<a href="#">DescribeDBClusters</a>	DBClusterArn
<a href="#">DescribeDBClusterSnapshots</a>	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 <a href="#">Database engine updates for Amazon Aurora MySQL (p. 990)</a>
Amazon Aurora PostgreSQL	See <a href="#">Amazon Aurora PostgreSQL updates (p. 1413)</a>

## 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

These topics provide information about backing up and restoring Amazon Aurora DB clusters.

## 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 \(p. 71\)](#).

## Topics

- [Overview of backing up and restoring an Aurora DB cluster \(p. 369\)](#)
- [Understanding Aurora backup storage usage \(p. 372\)](#)
- [Creating a DB cluster snapshot \(p. 373\)](#)
- [Restoring from a DB cluster snapshot \(p. 375\)](#)
- [Copying a DB cluster snapshot \(p. 378\)](#)
- [Sharing a DB cluster snapshot \(p. 388\)](#)
- [Exporting DB cluster snapshot data to Amazon S3 \(p. 396\)](#)
- [Restoring a DB cluster to a specified time \(p. 415\)](#)
- [Deleting a DB cluster snapshot \(p. 417\)](#)
- [Tutorial: Restore an Amazon Aurora DB cluster from a DB cluster snapshot \(p. 419\)](#)

# 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.

## 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 \(p. 372\)](#). 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.

<b>Region Name</b>	<b>Region</b>	<b>Time Block</b>
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
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 \(p. 433\)](#). 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 \(p. 415\)](#).

## 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 Amazon Aurora DB cluster \(p. 280\)](#).

## 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 \(p. 725\)](#).

# 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 \(p. 369\)](#). 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 \(p. 542\)](#).

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 cluster snapshot data to Amazon S3 \(p. 396\)](#).

You can create a DB cluster snapshot using the AWS Management Console, the AWS CLI, or the RDS API.

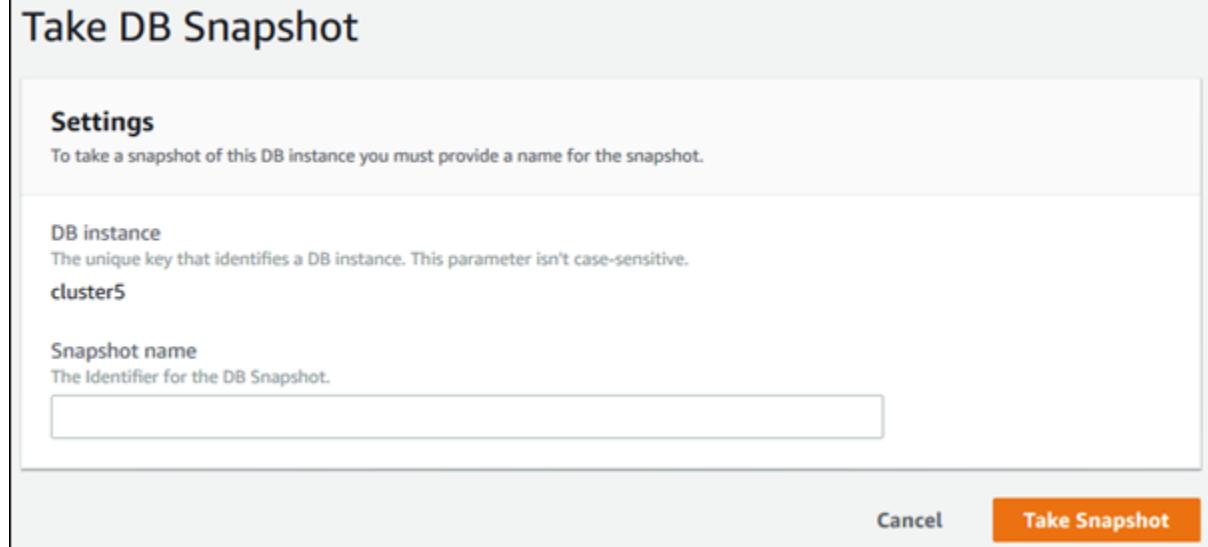
## Console

### 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 \(p. 388\)](#).

## 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 parameter groups \(p. 215\)](#).

## 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 \(p. 248\)](#).

## 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 \(p. 1566\)](#).

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 \(p. 790\)](#).

## 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. Specify other settings.  
For information about each setting, see [Settings for Aurora DB clusters \(p. 137\)](#).
7. 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`.

You can specify other settings. For information about each setting, see [Settings for Aurora DB clusters \(p. 137\)](#).

#### 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 Aurora, 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 \(p. 388\)](#).

## 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 \(p. 372\)](#). 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 following AWS Regions:
  - China (Beijing)
  - China (Ningxia)
  - Asia Pacific (Jakarta)
- 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.
- When you request multiple snapshot copies for the same source DB instance, they're queued internally. The copies requested later won't start until the previous snapshot copies are completed. For more information, see [Why is my EC2 AMI or EBS snapshot creation slow?](#) in the AWS Knowledge Center.
- 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 \(p. 389\)](#).

## 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 \(p. 1640\)](#).

### 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-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.
- `--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 \  
--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 ^  
--kms-key-id my-us-east-1-key
```

The `--source-region` parameter is required when you're copying an encrypted DB cluster snapshot between the AWS GovCloud (US-East) and AWS GovCloud (US-West) Regions. For `--source-region`, specify the AWS Region of the source DB instance. The AWS Region specified in `source-db-cluster-snapshot-identifier` must match the AWS Region specified for `--source-region`.

If `--source-region` isn't specified, specify a `--pre-signed-url` value. A *presigned URL* is a URL that contains a Signature Version 4 signed request for the `copy-db-cluster-snapshot` command that's called in the source AWS Region. To learn more about the `pre-signed-url` option, see [copy-db-cluster-snapshot](#) in the *AWS CLI Command Reference*.

## 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](#).

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
%2525FAction%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
```

The `PreSignedUrl` parameter is required when you are copying an encrypted DB cluster snapshot between the AWS GovCloud (US-East) and AWS GovCloud (US-West) Regions. The `PreSignedUrl` value must be a URL that contains a Signature Version 4 signed request for the `CopyDBClusterSnapshot` operation 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](#) in the *Amazon RDS API Reference*.

To automatically rather than manually generate a presigned URL, use the AWS CLI [copy-db-cluster-snapshot](#) command with the `--source-region` option instead.

## 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=ef38f1ce3dab4e1dbf113d8d2a265c67d17ece1999ffd36be85714ed36dddbb3
```

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 \(p. 389\)](#).
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 \(p. 390\)](#).
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/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
```

# 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 \(p. 378\)](#). For more information on restoring a DB instance from a DB cluster snapshot, see [Restoring from a DB cluster snapshot \(p. 375\)](#).

For more information on restoring a DB cluster from a DB cluster snapshot, see [Overview of backing up and restoring an Aurora DB cluster \(p. 369\)](#).

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 \(p. 1638\)](#). 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 \(p. 389\)](#) 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

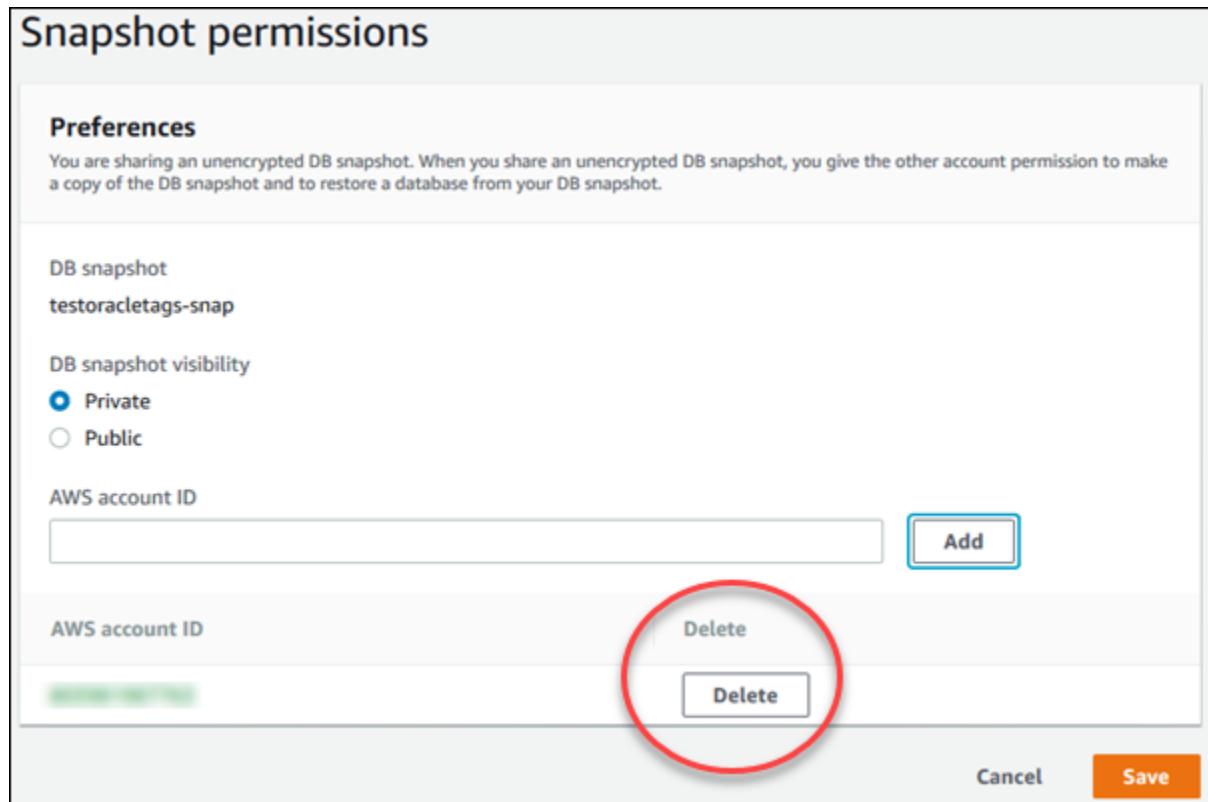
AWS account ID	Delete
Please add AWS account ID	

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 cluster snapshot data to Amazon S3

You can export DB cluster 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 cluster 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 \(p. 990\)](#).

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

All currently available Aurora PostgreSQL engine versions support exporting snapshot data to Amazon S3. Versions include Aurora PostgreSQL 14.3, 13.3, 12.4, 11.4, 10.6, 9.6.11 and all higher minor versions. For more information about versions, see the [Release Notes for Aurora PostgreSQL](#).

## Topics

- [Limitations \(p. 397\)](#)
- [Overview of exporting snapshot data \(p. 397\)](#)
- [Setting up access to an Amazon S3 bucket \(p. 398\)](#)
- [Using a cross-account AWS KMS key for encrypting Amazon S3 exports \(p. 401\)](#)
- [Exporting a snapshot to an Amazon S3 bucket \(p. 402\)](#)
- [Monitoring snapshot exports \(p. 404\)](#)
- [Canceling a snapshot export task \(p. 406\)](#)
- [Failure messages for Amazon S3 export tasks \(p. 406\)](#)
- [Troubleshooting PostgreSQL permissions errors \(p. 407\)](#)
- [File naming convention \(p. 408\)](#)
- [Data conversion when exporting to an Amazon S3 bucket \(p. 408\)](#)

## Limitations

Exporting DB snapshot data to Amazon S3 has the following limitations:

- The following characters in the S3 file path are converted to underscores (\_) during export:

```
\ \ ^ " (space)
```

- 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)
```

- Tables with slashes (/) in their names are skipped during export.
- 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.
- We strongly recommend that you use a unique name for each export task. If you don't use a unique task name, you might receive the following error message:

`ExportTaskAlreadyExistsFault: An error occurred (ExportTaskAlreadyExists) when calling the StartExportTask operation: The export task with the ID xxxxx already exists.`

- You can delete a snapshot while you're exporting its data to S3, but you're still charged for the storage costs for that snapshot until the export task has completed.
- You can't restore exported snapshot data from S3 to a new DB cluster.

## 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.

- Identify the snapshot to export.

Use an existing automated or manual snapshot, or create a manual snapshot of a DB instance.

- 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:

- 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 \(p. 398\)](#).
- 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 \(p. 398\)](#).

3. Create a symmetric encryption 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 \(p. 1638\)](#).

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 \(p. 401\)](#).

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 \(p. 402\)](#).
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 \(p. 398\)](#)
- [Providing access to an Amazon S3 bucket using an IAM role \(p. 398\)](#)
- [Using a cross-account Amazon S3 bucket \(p. 400\)](#)

## 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 \(p. 402\)](#).

## 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 \(p. 1686\)](#). 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": {  
                "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 \(p. 398\)](#), 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",
            "TaskId": "my-export-task"
        }
    ]
}
```

```

    "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 edgucuc-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 [Working with Amazon RDS event notification \(p. 572\)](#).

## Canceling 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.
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"> <li>• <code>AWSSecurityTokenServiceException</code> with the error code <code>AccessDenied</code></li> <li>• <code>AmazonS3Exception</code> with the error code <code>NoSuchBucket</code>, <code>AccessDenied</code>, <code>KMS.KMSInvalidStateException</code>, <code>403 Forbidden</code>, or <code>KMS.DisabledException</code></li> </ul> <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 \(p. 1723\)](#).

## 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 \(p. 409\)](#)
- [PostgreSQL data type mapping to Parquet \(p. 412\)](#)

## 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
<b>Numeric data types</b>			
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).

Source data type	Parquet primitive type	Logical type annotation	Conversion notes
	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		
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		
<b>String data types</b>			
BINARY	BYTE_ARRAY		
BLOB	BYTE_ARRAY		
CHAR	BYTE_ARRAY		
ENUM	BYTE_ARRAY	STRING	

Source data type	Parquet primitive type	Logical type annotation	Conversion notes
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	
TINYBLOB	BYTE_ARRAY		
TINYTEXT	BYTE_ARRAY	STRING	
VARBINARY	BYTE_ARRAY		
VARCHAR	BYTE_ARRAY	STRING	
<b>Date and time data types</b>			
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		
<b>Geometric data types</b>			
GEOMETRY	BYTE_ARRAY		
GEOMETRYCOLLECTION	BYTE_ARRAY		
MULTIPOINT	BYTE_ARRAY		
MULTIPOLYGON	BYTE_ARRAY		
POINT	BYTE_ARRAY		
POLYGON	BYTE_ARRAY		
<b>JSON data type</b>			
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
<b>Numeric data types</b>			
BIGINT	INT64		
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	
<b>String and related data types</b>			
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	

<b>PostgreSQL data type</b>	<b>Parquet primitive type</b>	<b>Logical type annotation</b>	<b>Mapping notes</b>
CHAR(N)	BYTE_ARRAY	STRING	
ENUM	BYTE_ARRAY	STRING	
NAME	BYTE_ARRAY	STRING	
TEXT	BYTE_ARRAY	STRING	
TEXT SEARCH	BYTE_ARRAY	STRING	
VARCHAR(N)	BYTE_ARRAY	STRING	
XML	BYTE_ARRAY	STRING	
<b>Date and time data types</b>			
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	
<b>Geometric data types</b>			
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	
<b>JSON data types</b>			
JSON	BYTE_ARRAY	STRING	
JSONB	BYTE_ARRAY	STRING	
<b>Other data types</b>			
BOOLEAN	BOOLEAN		
CIDR	BYTE_ARRAY	STRING	Network data type
COMPOSITE	BYTE_ARRAY	STRING	
DOMAIN	BYTE_ARRAY	STRING	

PostgreSQL data type	Parquet primitive type	Logical type annotation	Mapping notes
INET	BYTE_ARRAY	STRING	Network data type
MACADDR	BYTE_ARRAY	STRING	
OBJECT IDENTIFIER	N/A		
PG_LSN	BYTE_ARRAY	STRING	
RANGE	BYTE_ARRAY	STRING	
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 Aurora uploads log records 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 \(p. 369\)](#).

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 \(p. 1566\)](#).

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.

For information about each setting, see [Settings for Aurora DB clusters \(p. 137\)](#).

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.

You can specify other settings. For information about each setting, see [Settings for Aurora DB clusters \(p. 137\)](#).

### 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.

# Tutorial: Restore an Amazon Aurora DB cluster from a DB cluster snapshot

A common scenario when working with Amazon Aurora is to have a DB instance that you work with occasionally but that you don't need full time. For example, you might use a DB cluster to hold the data for a report that you run only quarterly. One way to save money on such a scenario is to take a DB cluster snapshot of the DB cluster after the report is completed. Then you delete the DB cluster, and restore it when you need to upload new data and run the report during the next quarter.

When you restore a DB cluster, you provide the name of the DB cluster snapshot to restore from. You then provide a name for the new DB cluster that's created from the restore operation. For more detailed information on restoring DB clusters from snapshots, see [Restoring from a DB cluster snapshot \(p. 375\)](#).

In this tutorial, we also upgrade the restored DB cluster from Aurora MySQL version 2 (compatible with MySQL 5.7) to Aurora MySQL version 3 (compatible with MySQL 8.0).

## Restoring a DB cluster from a DB cluster snapshot using the Amazon RDS console

When you restore a DB cluster from a snapshot using the AWS Management Console, the primary (reader) DB instance is also created.

### 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**.

Snapshot name	DB instance or cluster	Snapshot creation time
<input checked="" type="checkbox"/> my-57-cluster-snapshot	mynewdbcluster-cluster	July 05, 2022, 7:37:58 PM UTC
<input type="checkbox"/> database-2-instance-1-final-snapshot	database-2	June 28, 2022, 6:30:17 PM UTC

The **Restore snapshot** page appears.

5. Under **DB instance settings**, do the following:
  - a. Use the default setting for **DB engine**.
  - b. For **Available versions**, choose a MySQL-8.0 compatible version, such as **Aurora MySQL 3.02.0 (compatible with MySQL 8.0.23)**.

**Restore snapshot**

You are creating a new DB instance or DB cluster from a snapshot. The default VPC security group and parameter group are selected for the new DB instance or DB cluster, but you can change these settings.

**DB instance settings**

**DB engine**: Amazon Aurora MySQL-Compatible Edition

**Capacity type**: Provisioned

**Replication features**: Single-master replication is currently selected

**Engine version**: Aurora MySQL 3.02.0 (compatible with MySQL 8.0.23)

**Available versions (3/3)**

- Aurora MySQL 3.02.0 (compatible with MySQL 8.0.23)
- Aurora (MySQL 5.7) 2.10.2
- Aurora MySQL 3.01.1 (compatible with MySQL 8.0.23)
- Aurora MySQL 3.02.0 (compatible with MySQL 8.0.23)**

**Settings**

**DB snapshot ID**: my-57-cluster-snapshot

**DB cluster identifier**: mydbcluster

6. Under **Settings**, for **DB cluster identifier** enter the unique name that you want to use for the restored DB cluster, for example **my-80-cluster**.
7. Under **Connectivity**, use the default settings for the following:
  - **Virtual private cloud (VPC)**
  - **DB subnet group**
  - **Public access**
  - **VPC security group (firewall)**
8. Choose the **DB instance class**.

For this tutorial, choose **Burstable classes (includes t classes)**, and then choose **db.t3.medium**.

**Instance configuration**

The DB instance configuration options below are limited to those supported by the engine that you selected above.

DB instance class [Info](#)

Serverless

Memory optimized classes (includes r classes)

Burstable classes (includes t classes)

db.t3.medium  
 2 vCPUs   4 GiB RAM   Network: 2,085 Mbps

Include previous generation classes

9. For **Database authentication**, use the default setting.

10. For **Encryption**, use the default settings.

If the source DB cluster for the snapshot was encrypted, the restored DB cluster is also encrypted. You can't make it unencrypted.

11. Expand **Additional configuration** at the bottom of the page.

**▼ Additional configuration**

Database options, backup turned on, backtrack turned off, CloudWatch Logs, maintenance, delete protection turned off

**Database options**

DB cluster parameter group [Info](#)

default.aurora-mysql8.0

DB parameter group [Info](#)

default.aurora-mysql8.0

Option group [Info](#)

default:aurora-mysql-8-0

**Backup**

Copy tags to snapshots

**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

i Ensure that general, slow query, and audit logs are turned on. Error logs are enabled by default. [Learn more](#)

**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.

**Deletion protection**

Enable deletion protection

Protects the database from being deleted accidentally. While this option is enabled, you can't delete the database.

12. Make the following choices:

- a. For this tutorial, use the default value for **DB cluster parameter group**.
- b. For this tutorial, use the default value for **DB parameter group**.
- c. For **Log exports**, select all of the check boxes.

- d. For **Deletion protection**, select the **Enable deletion protection** check box.  
13. Choose **Restore DB instance**.

The **Databases** page displays the restored DB cluster, with a status of *Creating*.

DB identifier	DB cluster identifier	Role	Engine	Engine version
my-80-cluster-cluster	my-80-cluster-cluster	Regional cluster	Aurora MySQL	8.0.mysql_aurora.3.02.0
my-80-cluster	my-80-cluster-cluster	Reader instance	Aurora MySQL	8.0.mysql_aurora.3.02.0

While the primary DB instance is being created, it appears as a reader instance, but after creation it's a writer instance.

## Restoring a DB cluster from a DB cluster snapshot using the AWS CLI

Restoring a DB cluster from a snapshot using the AWS CLI has two steps:

1. Restore the DB cluster using the [restore-db-cluster-from-snapshot](#) command.
2. Create the primary (writer) DB instance using the [create-db-instance](#) command.

### Restoring the DB cluster

You use the `restore-db-cluster-from-snapshot` command. The following options are required:

- `--db-cluster-identifier` – The name of the restored DB cluster.
- `--snapshot-identifier` – The name of the DB snapshot to restore from.
- `--engine` – The database engine of the restored DB cluster. It must be compatible with the database engine of the source DB cluster.

The choices are the following:

- `aurora` – Aurora MySQL 5.6 compatible.
- `aurora-mysql` – Aurora MySQL 5.7 and 8.0 compatible.
- `aurora-postgresql` – Aurora PostgreSQL compatible.

In this example, we use `aurora-mysql`.

- `--engine-version` – The version of the restored DB cluster. In this example, we use a MySQL-8.0 compatible version.

The following example restores an Aurora MySQL 8.0-compatible DB cluster named `my-new-80-cluster` from a DB cluster snapshot named `my-57-cluster-snapshot`.

#### To restore the DB cluster

- Use one of the following commands.

For Linux, macOS, or Unix:

```
aws rds restore-db-cluster-from-snapshot \
--db-cluster-identifier my-new-80-cluster \
--snapshot-identifier my-57-cluster-snapshot \
--engine aurora-mysql \
--engine-version 8.0.mysql_aurora.3.02.0
```

For Windows:

```
aws rds restore-db-cluster-from-snapshot ^
--db-cluster-identifier my-new-80-cluster ^
--snapshot-identifier my-57-cluster-snapshot ^
--engine aurora-mysql ^
--engine-version 8.0.mysql_aurora.3.02.0
```

The output resembles the following.

```
{
  "DBCluster": {
    "AllocatedStorage": 1,
    "AvailabilityZones": [
      "eu-central-1b",
      "eu-central-1c",
      "eu-central-1a"
    ],
    "BackupRetentionPeriod": 14,
    "DatabaseName": "",
    "DBClusterIdentifier": "my-new-80-cluster",
    "DBClusterParameterGroup": "default.aurora-mysql8.0",
    "DBSubnetGroup": "default",
    "Status": "creating",
    "Endpoint": "my-new-80-cluster.cluster-#####.eu-central-1.rds.amazonaws.com",
    "ReaderEndpoint": "my-new-80-cluster.cluster-ro-#####.eu-central-1.rds.amazonaws.com",
    "MultiAZ": false,
    "Engine": "aurora-mysql",
    "EngineVersion": "8.0.mysql_aurora.3.02.0",
    "Port": 3306,
    "MasterUsername": "admin",
    "PreferredBackupWindow": "01:55-02:25",
    "PreferredMaintenanceWindow": "thu:21:14-thu:21:44",
    "ReadReplicaIdentifiers": [],
    "DBClusterMembers": [],
    "VpcSecurityGroups": [
      {
        "VpcSecurityGroupId": "sg-#####",
        "Status": "active"
      }
    ],
    "HostedZoneId": "Z1RLNU0EXAMPLE",
    "StorageEncrypted": true,
    "KmsKeyId": "arn:aws:kms:eu-central-1:123456789012:key/#####-5ccc-49cc-8aaa-#####
",
    "DbClusterResourceId": "cluster-ZZ12345678ITSJUSTANEXAMPLE",
    "DBClusterArn": "arn:aws:rds:eu-central-1:123456789012:cluster:my-new-80-cluster",
    "AssociatedRoles": [],
    "IAMDatabaseAuthenticationEnabled": false,
    "ClusterCreateTime": "2022-07-05T20:45:42.171000+00:00",
    "EngineMode": "provisioned",
    "DeletionProtection": false,
    "HttpEndpointEnabled": false,
```

```
        "CopyTagsToSnapshot": false,  
        "CrossAccountClone": false,  
        "DomainMemberships": [],  
        "TagList": []  
    }  
}
```

## Creating the primary (writer) DB instance

To create the primary (writer) DB instance, you use the `create-db-instance` command. The following options are required:

- `--db-cluster-identifier` – The name of the restored DB cluster.
- `--db-instance-identifier` – The name of the primary DB instance.
- `--db-instance-class` – The instance class of the primary DB instance. In this example, we use `db.t3.medium`.
- `--engine` – The database engine of the primary DB instance. It must be the same database engine as the restored DB cluster uses.

The choices are the following:

- `aurora` – Aurora MySQL 5.6 compatible.
- `aurora-mysql` – Aurora MySQL 5.7 and 8.0 compatible.
- `aurora-postgresql` – Aurora PostgreSQL compatible.

In this example, we use `aurora-mysql`.

The following example creates a primary (writer) DB instance named `my-new-80-cluster-instance` in the restored Aurora MySQL 8.0-compatible DB cluster named `my-new-80-cluster`.

### To create the primary DB instance

- Use one of the following commands.

For Linux, macOS, or Unix:

```
aws rds create-db-instance \  
  --db-cluster-identifier my-new-80-cluster \  
  --db-instance-identifier my-new-80-cluster-instance \  
  --db-instance-class db.t3.medium \  
  --engine aurora-mysql
```

For Windows:

```
aws rds create-db-instance ^  
  --db-cluster-identifier my-new-80-cluster ^  
  --db-instance-identifier my-new-80-cluster-instance ^  
  --db-instance-class db.t3.medium ^  
  --engine aurora-mysql
```

The output resembles the following.

```
{  
    "DBInstance": {  
        "DBInstanceIdentifier": "my-new-80-cluster-instance",  
        "DBInstanceClass": "db.t3.medium",
```

```

"Engine": "aurora-mysql",
"DBInstanceState": "creating",
"MasterUsername": "admin",
"AllocatedStorage": 1,
"PreferredBackupWindow": "01:55-02:25",
"BackupRetentionPeriod": 14,
"DBSecurityGroups": [],
"VpcSecurityGroups": [
    {
        "VpcSecurityGroupId": "sg-#####",
        "Status": "active"
    }
],
"DBParameterGroups": [
    {
        "DBParameterGroupName": "default.aurora-mysql8.0",
        "ParameterApplyStatus": "in-sync"
    }
],
"DBSubnetGroup": {
    "DBSubnetGroupName": "default",
    "DBSubnetGroupDescription": "default",
    "VpcId": "vpc-2305ca49",
    "SubnetGroupStatus": "Complete",
    "Subnets": [
        {
            "SubnetIdentifier": "subnet-#####",
            "SubnetAvailabilityZone": {
                "Name": "eu-central-1a"
            },
            "SubnetOutpost": {},
            "SubnetStatus": "Active"
        },
        {
            "SubnetIdentifier": "subnet-#####",
            "SubnetAvailabilityZone": {
                "Name": "eu-central-1b"
            },
            "SubnetOutpost": {},
            "SubnetStatus": "Active"
        },
        {
            "SubnetIdentifier": "subnet-#####",
            "SubnetAvailabilityZone": {
                "Name": "eu-central-1c"
            },
            "SubnetOutpost": {},
            "SubnetStatus": "Active"
        }
    ]
},
"PreferredMaintenanceWindow": "sat:02:41-sat:03:11",
"PendingModifiedValues": {},
"MultiAZ": false,
"EngineVersion": "8.0.mysql_aurora.3.02.0",
"AutoMinorVersionUpgrade": true,
"ReadReplicaDBInstanceIdentifiers": [],
"LicenseModel": "general-public-license",
"OptionGroupMemberships": [
    {
        "OptionGroupName": "default:aurora-mysql-8-0",
        "Status": "in-sync"
    }
],
"PubliclyAccessible": false,
"StorageType": "aurora",

```

```
        "DbInstancePort": 0,
        "DBClusterIdentifier": "my-new-80-cluster",
        "StorageEncrypted": true,
        "KmsKeyId": "arn:aws:kms:eu-central-1:534026745191:key/#####-5ccc-49cc-8aaa-
#####
        "DbiResourceId": "db-5C6UT5PU0YETANOTHEREXAMPLE",
        "CACertificateIdentifier": "rds-ca-2019",
        "DomainMemberships": [],
        "CopyTagsToSnapshot": false,
        "MonitoringInterval": 0,
        "PromotionTier": 1,
        "DBInstanceArn": "arn:aws:rds:eu-central-1:123456789012:db:my-new-80-cluster-
instance",
        "IAMDatabaseAuthenticationEnabled": false,
        "PerformanceInsightsEnabled": false,
        "DeletionProtection": false,
        "AssociatedRoles": [],
        "TagList": []
    }
}
```

# 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 \(p. 428\)](#)
- [Viewing cluster status and recommendations \(p. 432\)](#)
- [Viewing metrics in the Amazon RDS console \(p. 449\)](#)
- [Monitoring Amazon Aurora metrics with Amazon CloudWatch \(p. 452\)](#)
- [Monitoring DB load with Performance Insights on Amazon Aurora \(p. 461\)](#)
- [Analyzing performance anomalies with Amazon DevOps Guru for Amazon RDS \(p. 512\)](#)
- [Monitoring OS metrics with Enhanced Monitoring \(p. 518\)](#)
- [Metrics reference for Amazon Aurora \(p. 525\)](#)

# 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 \(p. 428\)](#)
- [Performance baseline \(p. 428\)](#)
- [Performance guidelines \(p. 428\)](#)
- [Monitoring tools \(p. 429\)](#)

## 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 parameter groups \(p. 215\)](#).
- **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 \(p. 429\)](#)
- [Manual monitoring tools \(p. 430\)](#)

## Automated monitoring tools

We recommend that you automate monitoring tasks as much as possible.

### Topics

- [Amazon Aurora cluster status and recommendations \(p. 429\)](#)
- [Amazon CloudWatch metrics for Amazon Aurora \(p. 430\)](#)
- [Amazon RDS Performance Insights and operating-system monitoring \(p. 430\)](#)
- [Integrated services \(p. 430\)](#)

### 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 \(p. 444\)](#).

## 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 \(p. 525\)](#)
  - **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 \(p. 461\)](#).
- **Amazon RDS Enhanced Monitoring** – Look at metrics in real time for the operating system. For more information, see [Monitoring OS metrics with Enhanced Monitoring \(p. 518\)](#).

## 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 \(p. 568\)](#).
- *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 \(p. 597\)](#).
- *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 \(p. 615\)](#).
- *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 \(p. 619\)](#).
- *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 Amazon DevOps Guru for Amazon RDS \(p. 512\)](#).

## 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 \(p. 433\)](#)
- [Viewing DB cluster status \(p. 439\)](#)
- [Viewing DB instance status in an Aurora cluster \(p. 441\)](#)
- [Viewing Amazon Aurora recommendations \(p. 444\)](#)

## 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.

#### To view or modify DB clusters in 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 **Databases**.
3. Choose the name of the Aurora DB cluster that you want to view from the 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.

The screenshot shows the AWS Management Console interface for an Aurora DB cluster named "aurora-test". In the top navigation bar, the cluster name "aurora-test" is visible. Below it, a "Related" section includes a search bar labeled "Filter databases". The main content area displays a table of DB instances under the heading "DB identifier". The table has columns for "Role", "Engine", and "Region & AZ". The data is as follows:

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

Below the table, there are tabs for "Connectivity & security" (which is highlighted in orange), "Monitoring", "Logs & events", "Configuration", "Maintenance & backups", and "Tags".

In the "Endpoints" section, there are two entries listed under "Endpoint name": "aurora-test.cluster-ro- .us-east-1.rds.amazonaws.com" and "aurora-test.cluster- .us-east-1.rds.amazonaws.com". A search bar labeled "Filter endpoint" is also present.

4. To modify a DB cluster, select the DB cluster from the list and choose **Modify**.

#### To view or modify DB instances of a DB cluster in 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 **Databases**.
3. Do one of the following:
  - To view a DB instance, choose one from the list that is a member of the Aurora DB cluster.

For example, if you choose the dbinstance4 DB instance identifier, the console shows the details page for the dbinstance4 DB instance, as shown in the following image.

Related			
<input type="text"/> 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	

<a href="#">Connectivity &amp; security</a>	<a href="#">Monitoring</a>	<a href="#">Logs &amp; events</a>	<a href="#">Configuration</a>	<a href="#">Maintenance</a>	<a href="#">Tags</a>
---	----------------------------	-----------------------------------	-------------------------------	-----------------------------	----------------------

## Connectivity & security

<b>Endpoint &amp; port</b>	Net
Endpoint	Available
dbinstance4. [REDACTED].us-east-1.rds.amazonaws.com	us-e
Port	VPC
3306	vpc-

- To modify a DB instance, choose the DB instance from the list and choose **Modify**. For more information about modifying a DB cluster, see [Modifying an Amazon Aurora DB cluster \(p. 248\)](#).

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 modify 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,  
      "ParameterGroups": [  
        {"ParameterGroup": "sample-cluster1", "Parameter": "characteristic", "Value": "value"}  
      ]  
    }  
  ]  
}
```

```
"DBClusterIdentifier": "sample-cluster1",
"MasterUsername": "mymasteruser",
"EarliestRestorableTime": "2016-03-30T03:35:42.563Z",
"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-cbfvmgb0y5fy.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 and the cluster instances by using the Amazon RDS console, the AWS CLI, or the API.

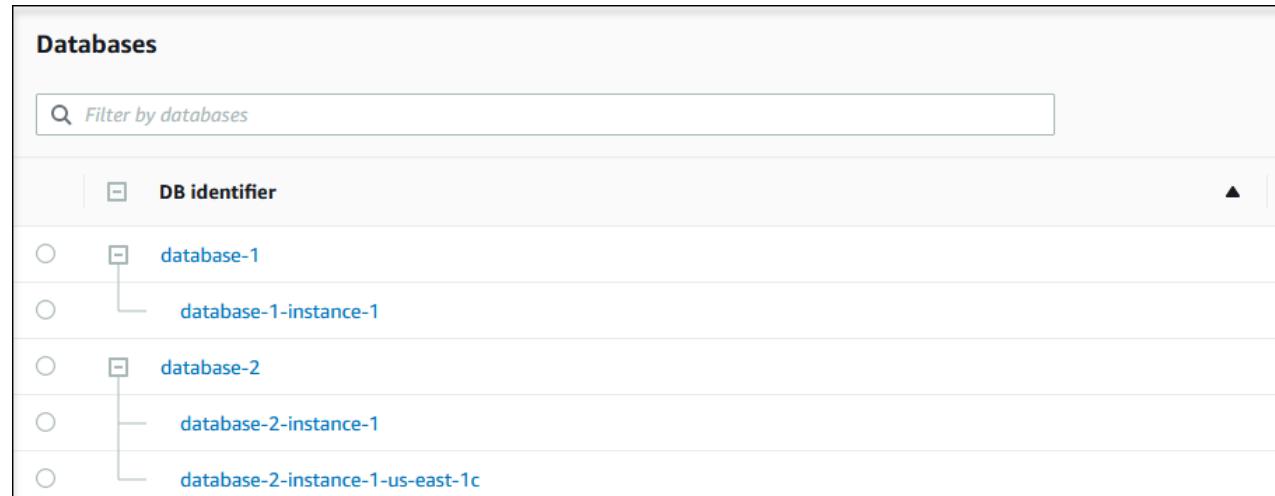
### Note

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 about maintenance status, see [Applying updates for a DB cluster \(p. 323\)](#).

### To view the status of 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**.

The **Databases page** appears with the list of DB clusters. For each DB cluster, the status value is displayed.



Find the possible status values for DB clusters in the following table.

DB cluster status	Billed	Description
<b>available</b>	Billed	The DB cluster is healthy and available. When an Aurora Serverless cluster is available and paused, you're billed for storage only.
<b>backing-up</b>	Billed	The DB cluster is currently being backed up.
<b>backtracking</b>	Billed	The DB cluster is currently being backtracked. This status only applies to Aurora MySQL.
<b>cloning-failed</b>	Not billed	Cloning a DB cluster failed.
<b>creating</b>	Not billed	The DB cluster is being created. The DB cluster is inaccessible while it is being created.
<b>deleting</b>	Not billed	The DB cluster is being deleted.

DB cluster status	Billed	Description
<b>failing-over</b>	Billed	A failover from the primary instance to an Aurora Replica is being performed.
<b>inaccessible-encryption-credentials</b>	Not billed	The AWS KMS key used to encrypt or decrypt the DB cluster can't be accessed or recovered.
<b>inaccessible-encryption-credentials-recoverable</b>	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 <a href="#">Encrypting an Amazon Aurora DB cluster (p. 1638)</a> .
<b>maintenance</b>	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.
<b>migrating</b>	Billed	A DB cluster snapshot is being restored to a DB cluster.
<b>migration-failed</b>	Not billed	A migration failed.
<b>modifying</b>	Billed	The DB cluster is being modified because of a customer request to modify the DB cluster.
<b>promoting</b>	Billed	A read replica is being promoted to a standalone DB cluster.
<b>renaming</b>	Billed	The DB cluster is being renamed because of a customer request to rename it.
<b>resetting-master-credentials</b>	Billed	The master credentials for the DB cluster are being reset because of a customer request to reset them.
<b>starting</b>	Billed for storage	The DB cluster is starting.
<b>stopped</b>	Billed for storage	The DB cluster is stopped.
<b>stopping</b>	Billed for storage	The DB cluster is being stopped.
<b>storage-optimization</b>	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 <b>storage-optimization</b> , 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.
<b>update-iam-db-auth</b>	Billed	IAM authorization for the DB cluster is being updated.
<b>upgrading</b>	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 about maintenance status, see [Applying updates for a DB cluster \(p. 323\)](#).

### To view the status of 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**.

The **Databases page** appears with the list of DB instances. For each DB instance in a cluster, the status value is displayed.

Databases			
<input type="text"/> Filter by databases			
DB identifier	Role	Status	
database-1	Regional cluster	 Available	
database-1-instance-1	Writer instance	 Available	
database-2	Regional cluster	 Available	
database-2-instance-1	Writer instance	 Available	
database-2-instance-1-us-east-1c	Reader instance	 Configuring-enhanced-monitoring	

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
<b>available</b>	Billed	The DB instance is healthy and available.
<b>backing-up</b>	Billed	The DB instance is currently being backed up.
<b>backtracking</b>	Billed	The DB instance is currently being backtracked. This status only applies to Aurora MySQL.
<b>configuring-enhanced-monitoring</b>	Billed	Enhanced Monitoring is being enabled or disabled for this DB instance.
<b>configuring-iam-database-auth</b>	Billed	AWS Identity and Access Management (IAM) database authentication is being enabled or disabled for this DB instance.
<b>configuring-log-exports</b>	Billed	Publishing log files to Amazon CloudWatch Logs is being enabled or disabled for this DB instance.

DB instance status	Billed	Description
<b>converting-to-vpc</b>	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.
<b>creating</b>	Not billed	The DB instance is being created. The DB instance is inaccessible while it is being created.
<b>deleting</b>	Not billed	The DB instance is being deleted.
<b>failed</b>	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.
<b>inaccessible-encryption-credentials</b>	Not billed	The AWS KMS key used to encrypt or decrypt the DB instance can't be accessed or recovered.
<b>inaccessible-encryption-credentials-recoverable</b>	Billed for storage	<p>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.</p> <p>For more information, see <a href="#">Encrypting an Amazon Aurora DB cluster (p. 1638)</a>.</p>
<b>incompatible-network</b>	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.
<b>incompatible-option-group</b>	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 <b>Recent Events</b> 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.
<b>incompatible-parameters</b>	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 <b>Recent Events</b> list for the DB instance.
<b>incompatible-restore</b>	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.
<b>insufficient-capacity</b>	Not billed	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.
<b>maintenance</b>	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.

DB instance status	Billed	Description
<b>modifying</b>	Billed	The DB instance is being modified because of a customer request to modify the DB instance.
<b>moving-to-vpc</b>	Billed	The DB instance is being moved to a new Amazon Virtual Private Cloud (Amazon VPC).
<b>rebooting</b>	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.
<b>resetting-master-credentials</b>	Billed	The master credentials for the DB instance are being reset because of a customer request to reset them.
<b>renaming</b>	Billed	The DB instance is being renamed because of a customer request to rename it.
<b>restore-error</b>	Billed	The DB instance encountered an error attempting to restore to a point-in-time or from a snapshot.
<b>starting</b>	Billed for storage	The DB instance is starting.
<b>stopped</b>	Billed for storage	The DB instance is stopped.
<b>stopping</b>	Billed for storage	The DB instance is being stopped.
<b>storage-full</b>	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.
<b>storage-optimization</b>	Billed	Amazon RDS is optimizing the storage of your DB instance. The DB instance is fully operational. The storage optimization process is usually short, but can sometimes take up to and even beyond 24 hours.
<b>upgrading</b>	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.

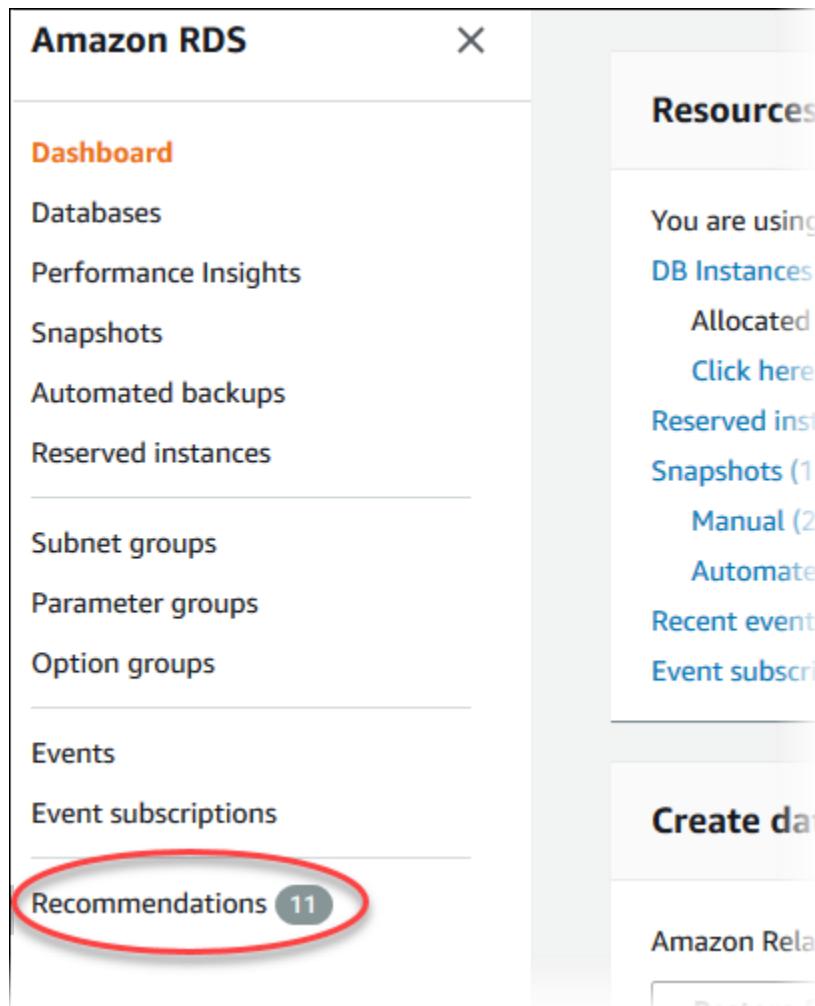
Type	Description	Recommendation	Additional information
Nondefault custom memory parameters	Your DB parameter group sets memory parameters that diverge too much from the default values.	Settings that diverge too much from the default values can cause poor performance and errors. We recommend setting custom memory parameters to their default values in the DB parameter group used by the DB instance.	<a href="#">Working with parameter groups (p. 215)</a>
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.	<a href="#">Best practices for configuring parameters for Amazon RDS for MySQL, part 1: Parameters related to performance</a> 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> .	<a href="#">Aurora MySQL database log files (p. 604)</a>
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.	<a href="#">High availability for Amazon Aurora (p. 71)</a>
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.	<a href="#">High availability for Amazon Aurora (p. 71)</a>
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	<a href="#">Maintaining an Amazon Aurora DB cluster (p. 321)</a>

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.	<a href="#">Working with parameter groups (p. 215)</a>
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.	<a href="#">Aurora Replicas (p. 73)</a>

Amazon Aurora generates recommendations for a resource when the resource is created or modified. Amazon Aurora also periodically scans your resources and generates recommendations.

### To view 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, it says 'Recommendations' with tabs for 'Active (2)', 'Dismissed (0)', 'Scheduled (0)', and 'Applied (0)'. There are two items listed: 1. 'Heterogeneous instance classes in DB cluster (1)' - DB clusters which have heterogeneous instance types. 2. 'Heterogeneous parameters for DB cluster (1)' - DB clusters which have heterogeneous parameter groups assigned to its instances. Each item has an 'Info' link.

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.

To configure preferences for displaying recommendations in each section, choose the **Preferences** icon.

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. (optional) Respond to your active recommendations as follows:
  - a. Choose **Active** and open one or more sections to view the recommendations in them.
  - b. 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 \(p. 248\)](#).

**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 \(p. 525\)](#).

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 \(p. 525\)](#).
- **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 \(p. 518\)](#). For a list of Enhanced Monitoring metrics, see [OS metrics in Enhanced Monitoring \(p. 558\)](#).
- **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 \(p. 461\)](#). For a list of Performance Insights metrics, see [Amazon CloudWatch metrics for Performance Insights \(p. 544\)](#).

## To view metrics for your Aurora DB cluster in the RDS console

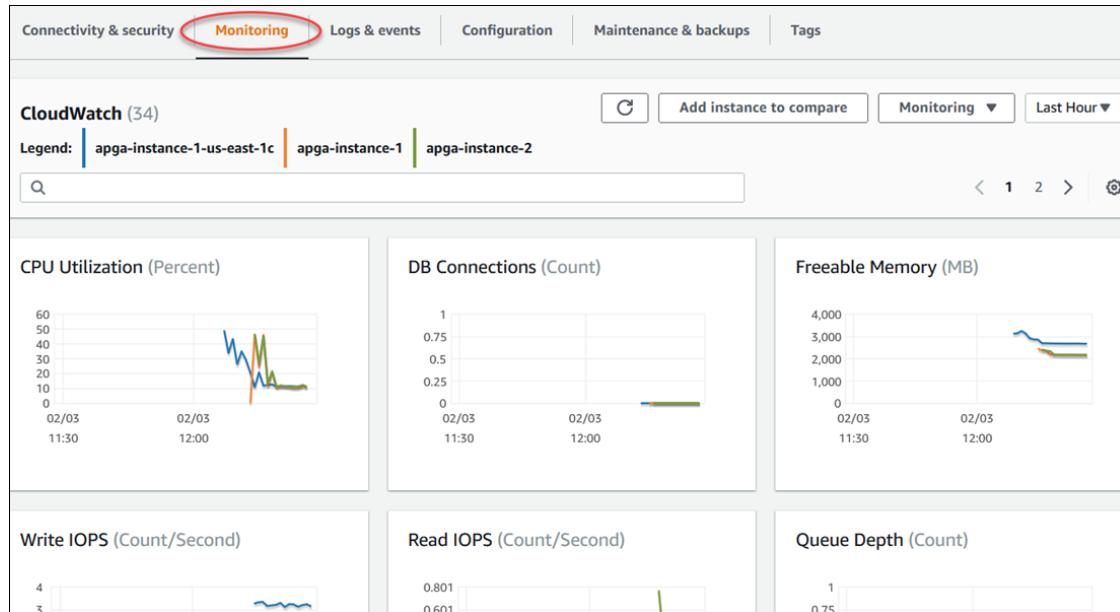
1. Sign in to the AWS Management Console and open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. In the navigation pane, choose **Databases**.
3. Choose the name of the Aurora DB cluster that you want to monitor.

The database page appears. The following example shows an Amazon Aurora PostgreSQL database named `apg`.

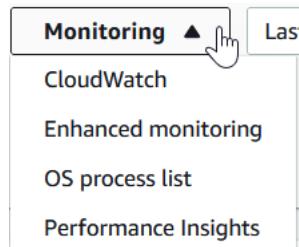
DB identifier	DB cluster identifier	Role	Engine
apg	apg	Regional cluster	Aurora PostgreSQL
apg-instance-1-us-east-1	apg	Writer instance	Aurora PostgreSQL
apg-instance-1	apg	Reader instance	Aurora PostgreSQL
apg-instance-2	apg	Reader instance	Aurora PostgreSQL

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 \(p. 525\)](#).

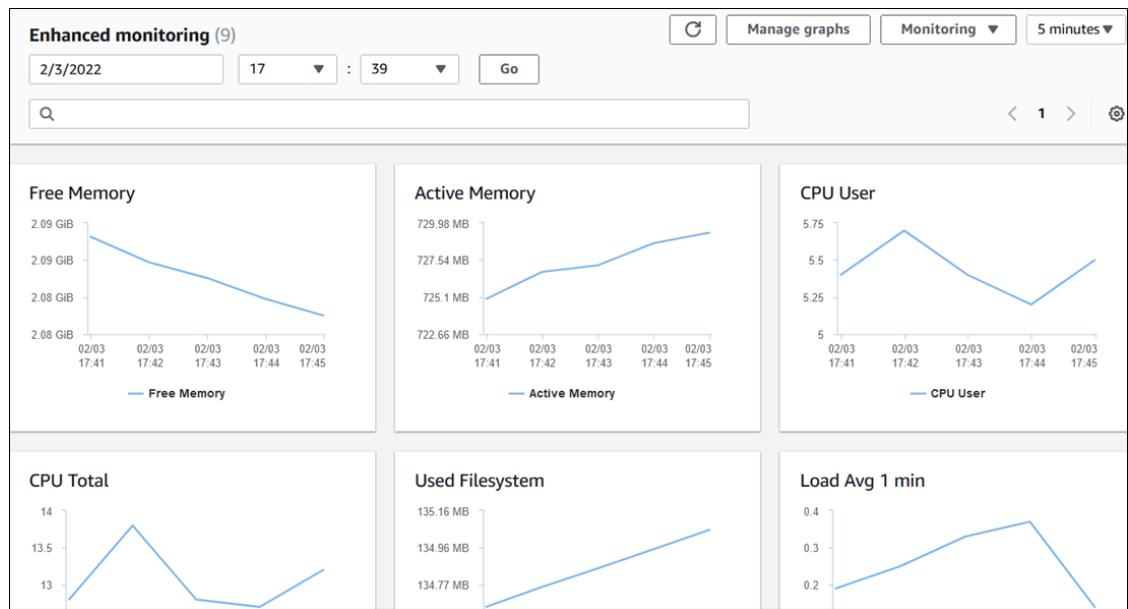


5. Choose **Monitoring** to see the metric categories.



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 \(p. 558\)](#).



**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 [Metrics reference for Amazon Aurora](#).

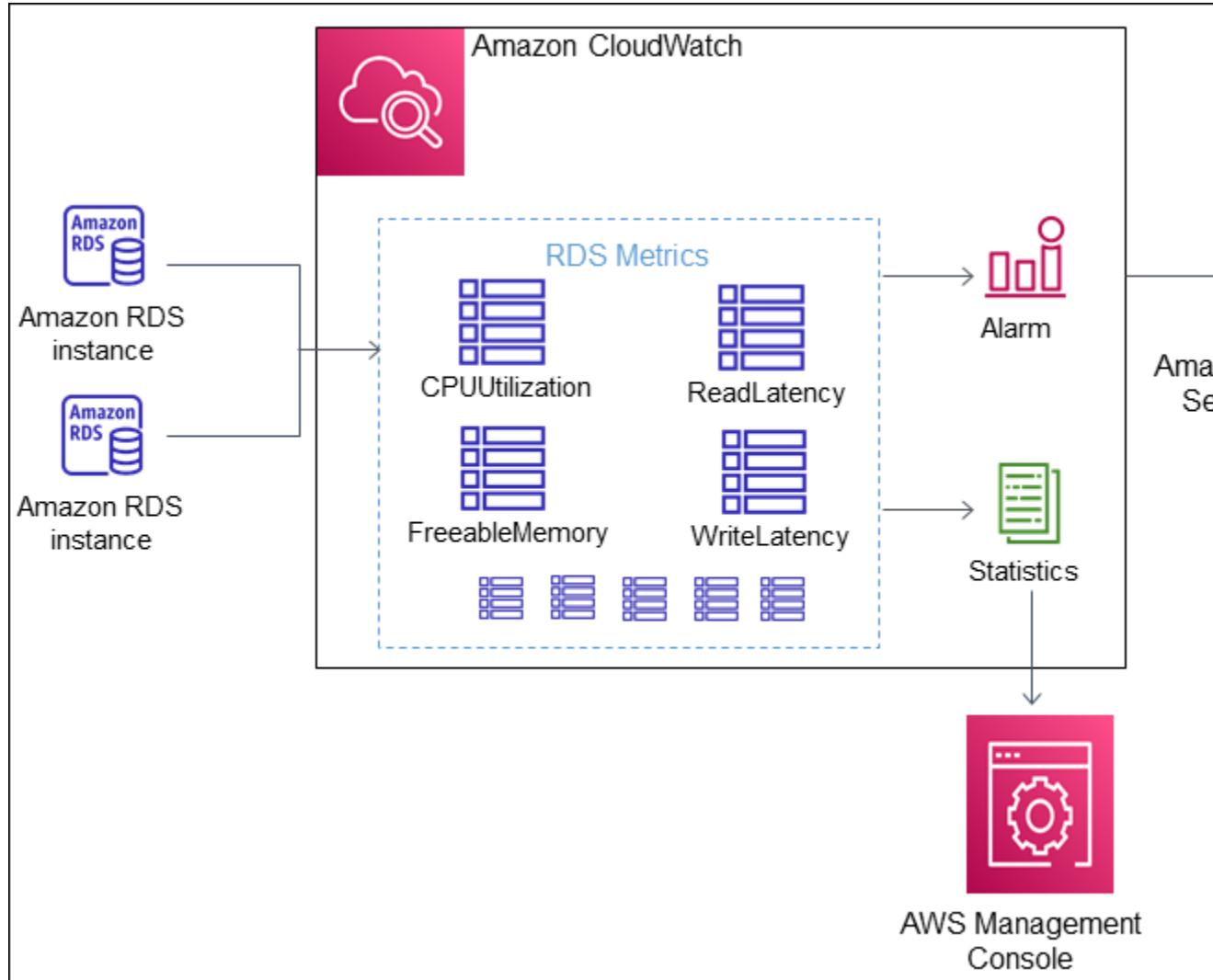
## Topics

- [Overview of Amazon Aurora and Amazon CloudWatch \(p. 453\)](#)
- [Viewing DB cluster metrics in the CloudWatch console and AWS CLI \(p. 454\)](#)
- [Creating CloudWatch alarms to monitor Amazon Aurora \(p. 459\)](#)

## Overview of Amazon Aurora and Amazon CloudWatch

By default, Amazon Aurora automatically sends metric data to CloudWatch in 1-minute periods. For example, the `CPUUtilization` metric records the percentage of CPU utilization for a DB instance over time. 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.

As shown in the following diagram, you can set up alarms for your CloudWatch metrics. For example, you might create an alarm that signals when the CPU utilization for an instance is over 70%. You can configure Amazon Simple Notification Service to email you when the threshold is passed.



Amazon RDS publishes the following types of metrics to Amazon CloudWatch:

- Aurora metrics at both the cluster and instance level

For a table of these metrics, see [Amazon CloudWatch metrics for Amazon Aurora \(p. 525\)](#).

- Performance Insights metrics

For a table of these metrics, see [Amazon CloudWatch metrics for Performance Insights \(p. 544\)](#) and [Performance Insights counter metrics \(p. 546\)](#).

- Enhanced Monitoring metrics (published to Amazon CloudWatch Logs)  
For a table of these metrics, see [OS metrics in Enhanced Monitoring \(p. 558\)](#).
- Usage metrics for the Amazon RDS service quotas in your AWS account  
For a table of these metrics, see [Amazon CloudWatch usage metrics for Amazon Aurora \(p. 541\)](#).  
For more information about Amazon RDS quotas, see [Quotas and constraints for Amazon Aurora \(p. 1756\)](#).

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](#).

## Viewing DB cluster metrics in the CloudWatch console and AWS 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 \(p. 518\)](#).

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.

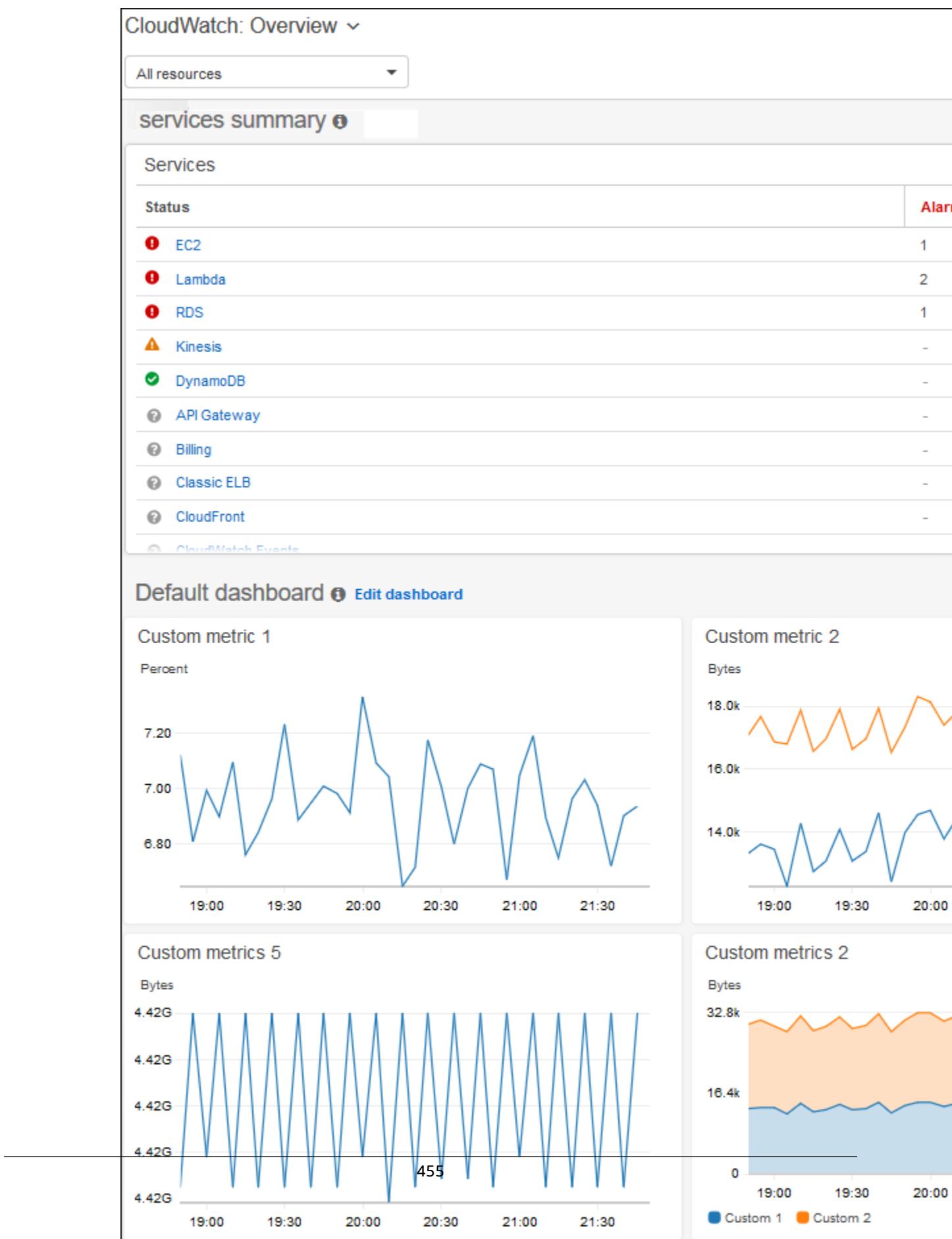
### Console

#### To view metrics using the Amazon CloudWatch console

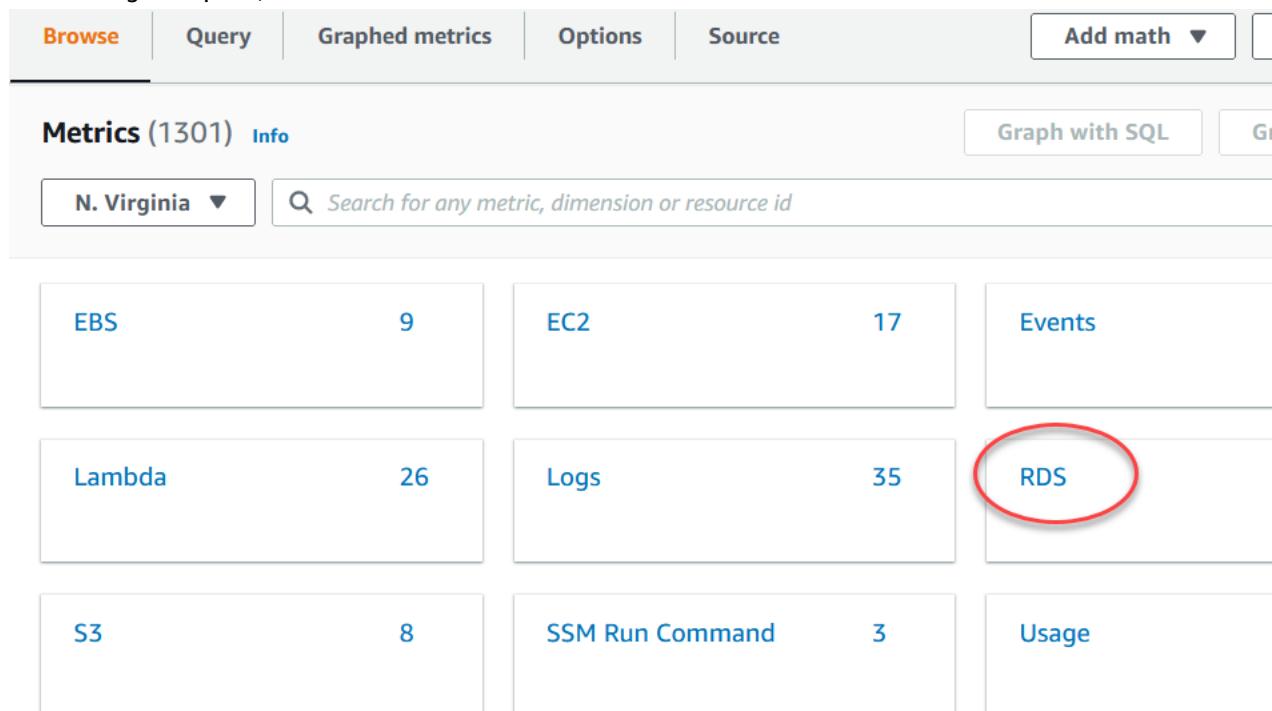
Metrics are grouped first by the service namespace, and then by the various dimension combinations within each namespace.

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.

The CloudWatch overview home page appears.

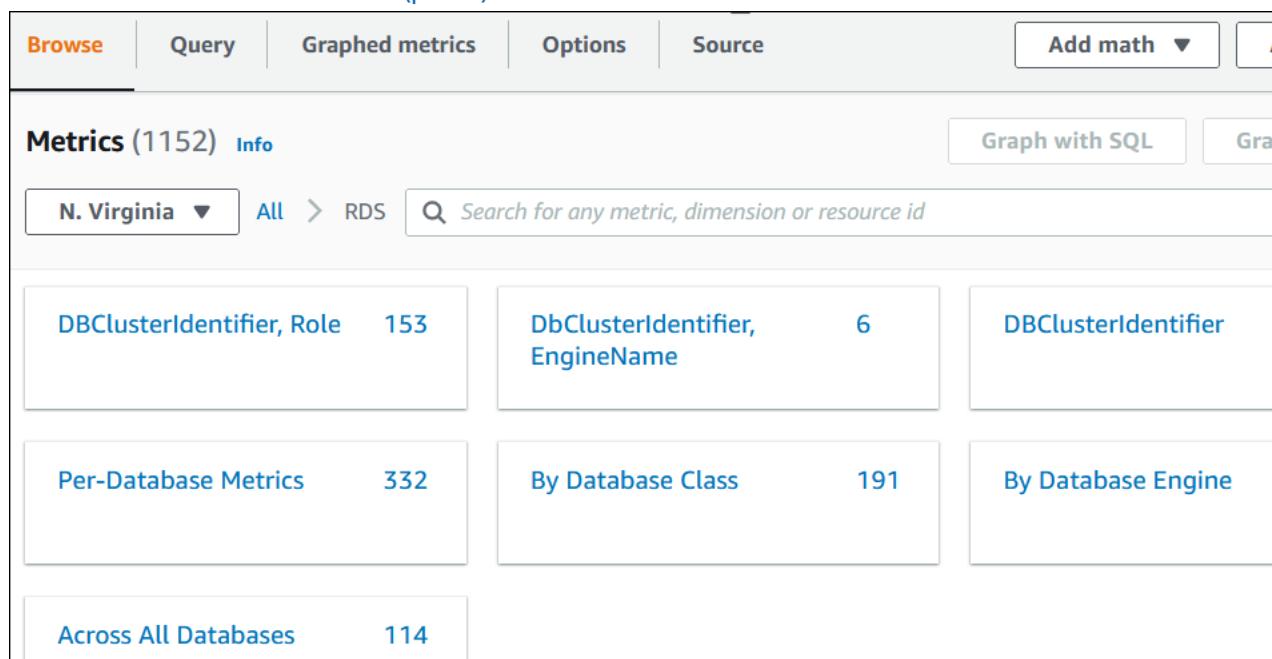


2. If necessary, change the AWS Region. From the navigation bar, choose the AWS Region where your AWS resources are. For more information, see [Regions and endpoints](#).
3. In the navigation pane, choose **Metrics** and then **All metrics**.



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 \(p. 541\)](#).



5. Choose a metric dimension, for example **By Database Class**.

Browse    Query    Graphed metrics (1)    Options    Source    Add math ▾    Add query ▾

**Metrics (191)** [Info](#)

N. Virginia ▾    All > RDS > By Database Class     Search for any metric, dimension or resource id

	Metric name
<input type="checkbox"/>	DatabaseClass (191)
<input type="checkbox"/>	db.r6g.large ▾
<input type="checkbox"/>	db.r6g.large ▾
<input type="checkbox"/>	db.r6g.large ▾

6. 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.

Untitled graph

Browse    Query    Graphed metrics (2/3)    Options    Source    Add math ▾    Add query ▾

**Metrics (17)** [Info](#)

N. Virginia ▾    All > RDS > By Database Class

CPU X

	Metric name
<input type="checkbox"/>	DatabaseClass (17)
<input type="checkbox"/>	db.t3.medium ▾
<input checked="" type="checkbox"/>	db.t3.medium ▾
<input type="checkbox"/>	db.t3.medium ▾

Add to search  
 Search for this only  
 Remove from graph  
 Graph this metric only  
 Graph all search results  
 Graph with SQL query  
 What is this?

## 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 \(p. 305\)](#).

### 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, All alarms**.

Choose **Create an alarm**.

This action launches a wizard.

3. Choose **Select metric**.
4. In **Browse**, choose **RDS**.
5. Search for the metric that you want to place an alarm on. For example, search for **CPUutilization**. To display only Amazon RDS metrics, search for the identifier of your resource.
6. Choose the metric to create an alarm on. Then choose **Select metric**.
7. In **Conditions**, define the alarm condition. Then choose **Next**.

For example, you can specify that the alarm should be set when CPU utilization is over 75%.

8. Choose your notification method. Then choose **Next**.

For example, to configure CloudWatch to send you an email when the alarm state is reached, do the following:

1. Choose **Create new topic** (if one doesn't exist).
2. Enter a topic name.
3. Enter the email endpoints.

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.

4. Choose **Create topic**.
5. Choose **Next**.
9. Enter a name and description for the alarm. The name must contain only ASCII characters. Then choose **Next**.

10. Preview the alarm that you're about to create. 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](#)

For more information about setting up Amazon SNS topics and creating alarms, see [Using Amazon CloudWatch alarms](#).

# Monitoring DB load with Performance Insights on Amazon Aurora

Performance Insights expands on existing Amazon Aurora monitoring features to illustrate and help you analyze your cluster performance. With the Performance Insights dashboard, you can visualize the database load on your Amazon Aurora cluster load and filter the load by waits, SQL statements, hosts, or users. For information about using Performance Insights with Amazon DocumentDB, see [Amazon DocumentDB Developer Guide](#).

## Topics

- [Overview of Performance Insights on Amazon Aurora \(p. 461\)](#)
- [Turning Performance Insights on and off \(p. 466\)](#)
- [Turning on the Performance Schema for Performance Insights on Aurora MySQL \(p. 469\)](#)
- [Configuring access policies for Performance Insights \(p. 473\)](#)
- [Analyzing metrics with the Performance Insights dashboard \(p. 476\)](#)
- [Retrieving metrics with the Performance Insights API \(p. 496\)](#)
- [Logging Performance Insights calls using AWS CloudTrail \(p. 510\)](#)

## Overview of Performance Insights on Amazon Aurora

By default, Performance Insights is turned on in the console create wizard for all Amazon RDS engines. If you turn on Performance Insights at the DB cluster level, Performance Insights is turned on for every DB instance in the cluster. If you have more than one database on a DB instance, Performance Insights aggregates performance data.

You can find an overview of Performance Insights for Amazon Aurora in the following video.

[Using Performance Insights to Analyze Performance of Amazon Aurora PostgreSQL](#)

## Topics

- [Database load \(p. 461\)](#)
- [Maximum CPU \(p. 464\)](#)
- [Amazon Aurora DB engine and instance class support for Performance Insights \(p. 464\)](#)
- [AWS Region support for Performance Insights \(p. 465\)](#)
- [Pricing and data retention for Performance Insights \(p. 465\)](#)

## 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 \(p. 462\)](#)
- [Average active sessions \(p. 462\)](#)
- [Average active executions \(p. 462\)](#)
- [Dimensions \(p. 463\)](#)

## 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

Elapsed time (sec)	Total execution time (sec)	AAE	Calculation
180	380	2.11	380 execution seconds/180 elapsed seconds
240	380	1.58	380 execution seconds/240 elapsed seconds
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 \(p. 463\)](#)
- [Top SQL \(p. 464\)](#)

For a complete list of dimensions for the Aurora engines, see [DB load sliced by dimensions \(p. 479\)](#).

### 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 \(p. 971\)](#). To learn how to tune using these wait events, see [Tuning Aurora MySQL with wait events and thread states \(p. 746\)](#).
- 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 \(p. 1395\)](#). To learn how to tune using these wait events, see [Tuning with wait events for Aurora PostgreSQL \(p. 1152\)](#).
- For information about all PostgreSQL wait events, see [The Statistics Collector > Wait Event tables](#) 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 and instance class 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	Performance Insights is supported for the following engine versions: <ul style="list-style-type: none"><li>• 3.0 and higher 3 versions (compatible with MySQL 8.0)</li><li>• 2.04.2 and higher 2 versions (compatible with MySQL 5.7)</li></ul>	Performance Insights with parallel query enabled is supported for the following engine versions: <ul style="list-style-type: none"><li>• 2.09.0 and higher 2 versions (compatible with MySQL 5.7)</li><li>• 1.23.0 and higher 1 versions (compatible with MySQL 5.6)</li></ul>	Performance Insights has the following engine class restrictions: <ul style="list-style-type: none"><li>• db.t2 – Not supported</li><li>• db.t3 – Not supported</li><li>• db.t4g – Supported only for 3 versions (compatible with MySQL 8.0) and 2.10.1 and higher 2 versions</li></ul>

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
	<ul style="list-style-type: none"><li>1.17.3 and higher 1 versions (compatible with MySQL 5.6)</li></ul>		(compatible with MySQL 5.7).
Amazon Aurora PostgreSQL-Compatible Edition	Performance Insights is supported for all engine versions.	N/A	N/A

**Note**

Aurora Serverless v2 supports Performance Insights for all MySQL-compatible and PostgreSQL-compatible versions. We recommend that you set the minimum capacity to at least 2 Aurora capacity units (ACUs).

Aurora Serverless v1 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)

## Pricing and data retention for Performance Insights

By default, Performance Insights offers a free tier that includes 7 days of performance data history and 1 million API requests per month. You can also purchase longer retention periods. For complete pricing information, see [Performance Insights Pricing](#).

In the RDS console, you can choose any of the following retention periods for your Performance Insights data:

- **Default (7 days)**
- **n months**, where **n** is a number from 1–24

## Performance Insights [Info](#)

Turn on Performance Insights [Info](#)

Retention period [Info](#)

7 days (free tier)

7 days (free tier)

1 month

2 months

3 months

4 months

5 months

6 months

7 months

8 months

9 months

10 months

11 months

12 months

13 months

14 months

To learn how to set a retention period using the AWS CLI, see [AWS CLI \(p. 468\)](#).

## Turning Performance Insights on and off

You can turn on Performance Insights for your DB cluster when you create it. If needed, you can turn it off later at the instance level for any instance in your DB cluster. Turning Performance Insights on and off 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 [Turning on the Performance Schema for Performance Insights on Aurora MySQL \(p. 469\)](#).

If you use Performance Insights with Aurora global databases, turn on 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 \(p. 203\)](#).

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 turn Performance Insights on or off when you create or modify a DB cluster.

### Turning Performance Insights on or off when creating a DB cluster

When you create a new DB cluster, turn on Performance Insights by choosing **Enable Performance Insights** in the **Performance Insights** section. Or choose **Disable Performance Insights**. To create a DB cluster, follow the instructions for your DB engine in [Creating an Amazon Aurora DB cluster \(p. 127\)](#).

The following screenshot shows the **Performance Insights** section.



If you choose **Enable Performance Insights**, you have the following options:

- **Retention** – The amount of time to retain Performance Insights data. The retention setting in the free tier is **Default (7 days)**. To retain your performance data for longer, specify 1–24 months. For more information about retention periods, see [Pricing and data retention for Performance Insights \(p. 465\)](#).
- **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 \(p. 475\)](#).

### Turning Performance Insights on or off when modifying a DB instance in your DB cluster

In the console, you can modify a DB instance in your DB cluster to turn Performance Insights on or off. You can't turn Performance Insights on or off at the cluster level: you must do it for each instance in the cluster.

#### To turn Performance Insights on or off for a DB instance in your 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 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. The retention setting in the free tier is **Default (7 days)**. To retain your performance data for longer, specify 1–24 months. For more information about retention periods, see [Pricing and data retention for Performance Insights \(p. 465\)](#).
  - **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 \(p. 1638\)](#).
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 turns on Performance Insights immediately.
  7. Choose **Modify instance**.

## AWS CLI

When you use the `create-db-instance` AWS CLI command, turn on Performance Insights by specifying `--enable-performance-insights`. Or turn off 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 turn Performance Insights on or off for an existing DB instance in your DB cluster using the AWS CLI.

### To turn Performance Insights on or off for a DB instance in your DB cluster 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 in your DB cluster.
  - `--enable-performance-insights` to turn on or `--no-enable-performance-insights` to turn off

The following example turns on 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 turn on Performance Insights in the CLI, you can optionally specify the number of days to retain Performance Insights data with the `--performance-insights-retention-period` option.

You can specify  $7, month * 31$  (where *month* is a number from 1–23), or 731. For example, if you want to retain your performance data for 3 months, specify 93, which is  $3 * 31$ . The default is 7 days. For more information about retention periods, see [Pricing and data retention for Performance Insights \(p. 465\)](#).

The following example turns on Performance Insights for `sample-db-instance` and specifies that Performance Insights data is retained for 93 days (3 months).

For Linux, macOS, or Unix:

```
aws rds modify-db-instance \
    --db-instance-identifier sample-db-instance \
    --enable-performance-insights \
    --performance-insights-retention-period 93
```

For Windows:

```
aws rds modify-db-instance ^
    --db-instance-identifier sample-db-instance ^
    --enable-performance-insights ^
    --performance-insights-retention-period 93
```

If you specify a retention period such as 94 days, which isn't a valid value, RDS issues an error.

```
An error occurred (InvalidParameterValue) when calling the CreateDBInstance operation:
Invalid Performance Insights retention period. Valid values are: [7, 31, 62, 93, 124, 155,
186, 217,
248, 279, 310, 341, 372, 403, 434, 465, 496, 527, 558, 589, 620, 651, 682, 713, 731]
```

## RDS API

When you create a new DB instance in your DB cluster using the [CreateDBInstance](#) operation Amazon RDS API operation, turn on Performance Insights by setting `EnablePerformanceInsights` to `True`. To turn off Performance Insights, set `EnablePerformanceInsights` to `False`.

You can also specify the `EnablePerformanceInsights` value using the following API operations:

- [ModifyDBInstance](#)
- [CreateDBInstanceReadReplica](#)
- [RestoreDBInstanceFromS3](#)

When you turn on Performance Insights, you can optionally specify the amount of time, in days, to retain Performance Insights data with the `PerformanceInsightsRetentionPeriod` parameter. You can specify  $7, month * 31$  (where *month* is a number from 1–23), or 731. For example, if you want to retain your performance data for 3 months, specify 93, which is  $3 * 31$ . The default is 7 days. For more information about retention periods, see [Pricing and data retention for Performance Insights \(p. 465\)](#).

## Turning on 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 of detail. The Performance Schema is designed to have minimal impact on database performance. Performance Insights is a separate feature that you can use with or without the Performance Schema.

### Topics

- [Overview of the Performance Schema \(p. 470\)](#)
- [Performance Insights and the Performance Schema \(p. 470\)](#)
- [Automatic management of the Performance Schema by Performance Insights \(p. 471\)](#)
- [Effect of a reboot on the Performance Schema \(p. 471\)](#)
- [Determining whether Performance Insights is managing the Performance Schema \(p. 472\)](#)
- [Configuring the Performance Schema for automatic management \(p. 472\)](#)

## Overview of the Performance Schema

The Performance Schema monitors events in Aurora MySQL databases. An *event* is a database server action that consumes time and has been instrumented so that timing information can be collected. Examples of events include the following:

- Function calls
- Waits for the operating system
- Stages of SQL execution
- Groups of SQL statements

The `PERFORMANCE_SCHEMA` storage engine is a mechanism for implementing the Performance Schema feature. This engine collects event data using instrumentation in the database source code. The engine stores events in memory-only 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 the *MySQL Reference Manual*.

## Performance Insights and the Performance Schema

Performance Insights and the Performance Schema are separate features, but they are connected. The behavior of Performance Insights for Aurora MySQL depends on whether the Performance Schema is turned on, and if so, whether Performance Insights manages the Performance Schema automatically. The following table describes the behavior.

Performance Schema turned on	Performance Insights management mode	Performance Insights behavior
Yes	Automatic	<ul style="list-style-type: none"><li>• Collects detailed, low-level monitoring information</li><li>• Collects active session metrics every second</li><li>• Displays DB load categorized by detailed wait events, which you can use to identify bottlenecks</li></ul>
Yes	Manual	<ul style="list-style-type: none"><li>• Doesn't collect wait events, per-SQL metrics, or other detailed, low-level monitoring information</li><li>• Collects active session metrics every five seconds instead of every second</li><li>• Reports user states such as inserting and sending, which don't help you identify bottlenecks</li></ul>
No	N/A	<ul style="list-style-type: none"><li>• Doesn't collect wait events, per-SQL metrics, or other detailed, low-level monitoring information</li><li>• Collects active session metrics every five seconds instead of every second</li></ul>

Performance Schema turned on	Performance Insights management mode	Performance Insights behavior
		<ul style="list-style-type: none"> <li>Reports user states such as inserting and sending, which don't help you identify bottlenecks</li> </ul>

## Automatic management of the Performance Schema by Performance Insights

When you create an Aurora MySQL DB instance with Performance Insights turned on, the Performance Schema is also turned on. In this case, Performance Insights automatically manages your Performance Schema parameters. This is the recommended configuration.

For Performance Insights to manage the Performance Schema automatically, the `performance_schema` must be set to 0. By default, the value of `Source` is `system`.

You can also manage the Performance Schema manually. If you choose this option, set the parameters according to the values in the following table.

Parameter name	Parameter value
<code>performance_schema</code>	1 ( <code>Source</code> column has the value <code>system</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

If you change the `performance_schema` parameter value manually, and then later want to change to automatic management, see [Configuring the Performance Schema for automatic management \(p. 472\)](#).

**Important**

When Performance Insights turns on the Performance Schema, it doesn't change the parameter group values. However, the values are changed on the DB instances that are running. The only way to see the changed values is to run the `SHOW GLOBAL VARIABLES` command.

## Effect of a reboot on the Performance Schema

Performance Insights and the Performance Schema differ in their requirements for DB instance reboots:

### Performance Schema

To turn this feature on or off, you must reboot the DB instance.

### Performance Insights

To turn this feature on or off, you don't need to reboot the DB instance.

If the Performance Schema isn't currently turned on, and you turn on Performance Insights without rebooting the DB instance, the Performance Schema won't be turned on.

## Determining whether Performance Insights is managing the Performance Schema

To find out whether Performance Insights is currently managing the Performance Schema for major engine versions 5.6, 5.7, and 8.0, review the following table.

Setting of performance_schema parameter	Setting of the Source column	Performance Insights is managing the Performance Schema?
0	system	Yes
0 or 1	user	No

### To determine whether Performance Insights is managing 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. Check whether **Source** is the system default and **Values** is **0**. If so, Performance Insights is managing the Performance Schema automatically. If not, Performance Insights isn't managing the Performance Schema automatically.

Name	Values	Allowed values	Modifiable	Source
performance_schema	1	0, 1	true	user

## Configuring the Performance Schema for automatic management

Assume that Performance Insights is turned on for your DB instance but isn't currently managing the Performance Schema. If you want to allow Performance Insights to manage the Performance Schema automatically, complete the following steps.

### To configure the Performance Schema for automatic management

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. Choose **Edit parameters**.
7. Select the **performance\_schema** parameter.

8. In **Values**, choose **0**.
9. Choose **Reset** and then **Reset parameters**.
10. Reboot the DB instance.

**Important**

Whenever you turn the Performance Schema on or off, make sure to reboot the DB instance.

For more information about modifying instance parameters, see [Modifying parameters in a DB parameter group \(p. 231\)](#). For more information about the dashboard, see [Analyzing metrics with the Performance Insights dashboard \(p. 476\)](#). For more information about the MySQL performance schema, see [MySQL 8.0 Reference Manual](#).

## Configuring access policies for Performance Insights

To access Performance Insights, a principal must have the appropriate permissions from AWS Identity and Access Management (IAM). You can grant access in the following ways:

- Attach the `AmazonRDSPerformanceInsightsReadOnly` managed policy to an IAM user or role.
- Create a custom IAM policy and attach it to an IAM user or role.

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 `AmazonRDSPerformanceInsightsReadOnly` policy to an IAM principal

`AmazonRDSPerformanceInsightsReadOnly` is an AWS-managed policy that grants access to all read-only operations of the Amazon RDS Performance Insights API. Currently, all operations in this API are read-only.

If you attach `AmazonRDSPerformanceInsightsReadOnly` 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 the `AmazonRDSPerformanceInsightsReadOnly` 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": "AmazonRDSPerformanceInsights:Describe*",  
            "Resource": "*"  
        }  
    ]  
}
```

```

        "Action": "rds:DescribeDBInstances",
        "Resource": "*"
    },
    {
        "Effect": "Allow",
        "Action": "rds:DescribeDBClusters",
        "Resource": "*"
    },
    {
        "Effect": "Allow",
        "Action": "pi:DescribeDimensionKeys",
        "Resource": "arn:aws:pi:us-east-1:111122223333:metrics/rds/*"
    },
    {
        "Effect": "Allow",
        "Action": "pi:GetDimensionKeyDetails",
        "Resource": "arn:aws:pi:us-east-1:111122223333:metrics/rds/*"
    },
    {
        "Effect": "Allow",
        "Action": "pi:GetResourceMetadata",
        "Resource": "arn:aws:pi:us-east-1:111122223333:metrics/rds/*"
    },
    {
        "Effect": "Allow",
        "Action": "pi:GetResourceMetrics",
        "Resource": "arn:aws:pi:us-east-1:111122223333:metrics/rds/*"
    },
    {
        "Effect": "Allow",
        "Action": "pi>ListAvailableResourceDimensions",
        "Resource": "arn:aws:pi:us-east-1:111122223333:metrics/rds/*"
    },
    {
        "Effect": "Allow",
        "Action": "pi>ListAvailableResourceMetrics",
        "Resource": "arn:aws:pi:us-east-1:111122223333:metrics/rds/*"
    }
]
}

```

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.

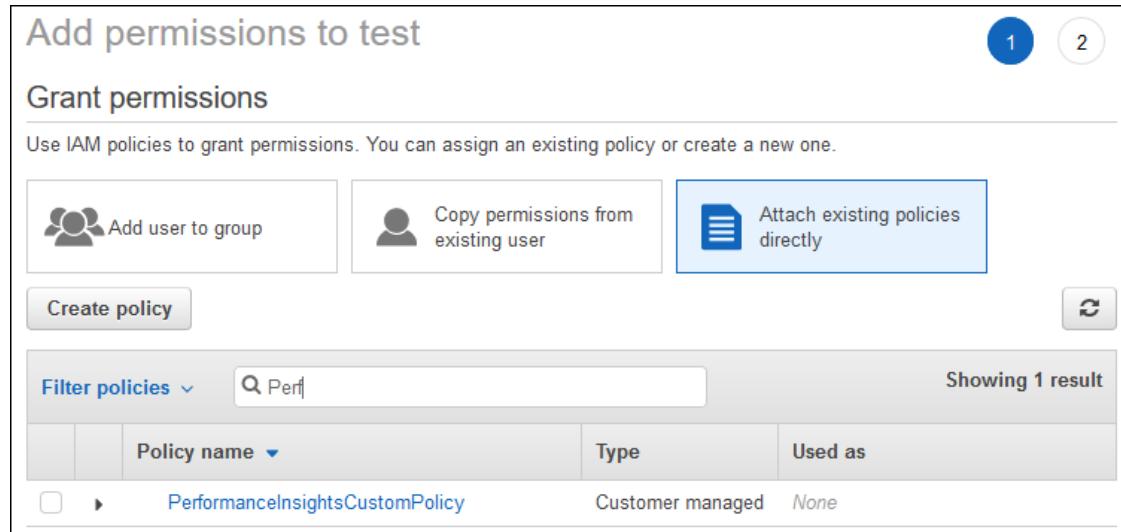
#### To attach the policy to an IAM user

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 `AmazonRDSPerformanceInsightsReadOnly` predefined policy provides read-only access to Amazon RDS. For more information, see [Managing access using policies \(p. 1655\)](#).

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 can do either of the following:

- 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 example 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-AAAAAABBBBCCCCDDDDDEEEEEE"
        }
    }
}
```

## Analyzing metrics with the Performance Insights dashboard

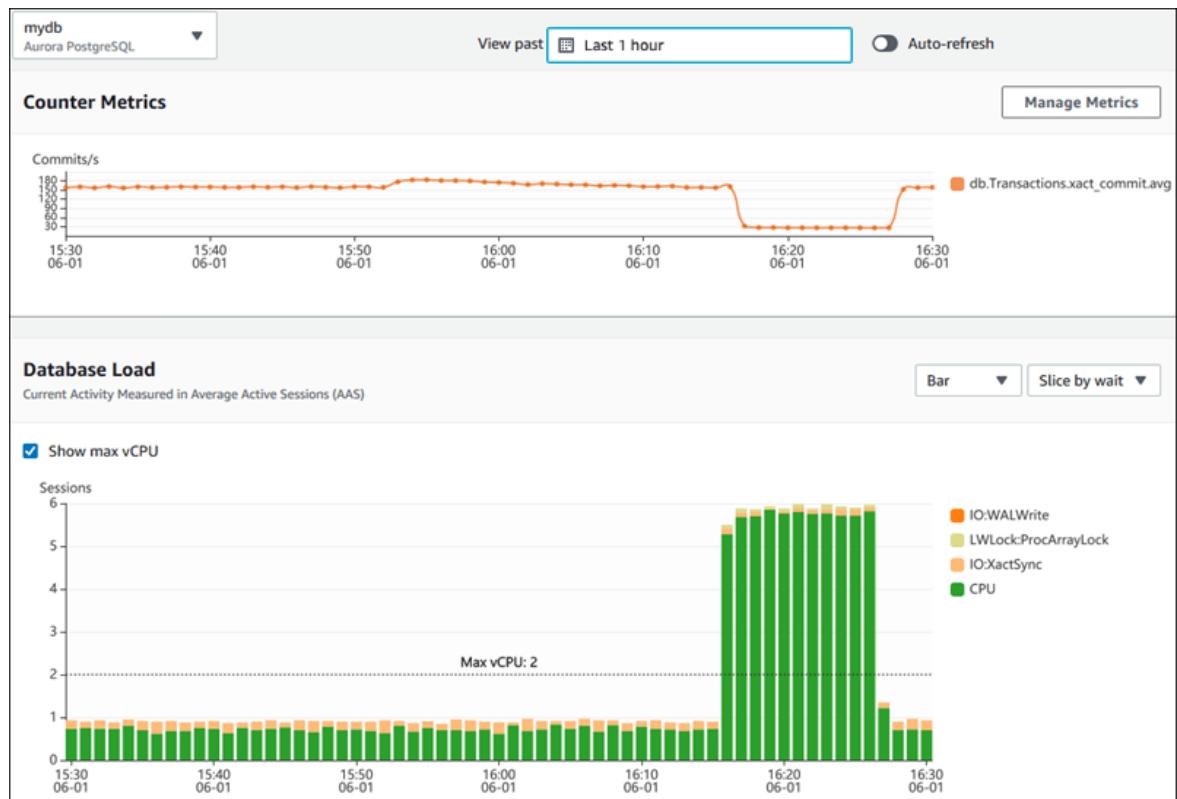
The Performance Insights dashboard contains database performance information to help you analyze and troubleshoot performance issues. On the main dashboard page, you can view information about the database load. You can "slice" DB load by dimensions such as wait events or SQL.

### Performance Insights dashboard

- [Overview of the Performance Insights dashboard \(p. 476\)](#)
- [Accessing the Performance Insights dashboard \(p. 482\)](#)
- [Analyzing DB load by wait events \(p. 484\)](#)
- [Analyzing queries in the Performance Insights dashboard \(p. 485\)](#)

## 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.



## Topics

- [Time range filter \(p. 477\)](#)
- [Counter metrics chart \(p. 477\)](#)
- [Database load chart \(p. 479\)](#)
- [Top dimensions table \(p. 481\)](#)

## Time range filter

By default, the Performance Insights dashboard shows DB load for the last hour. You can adjust this range to be as short as 5 minutes or as long as 2 years. You can also select a custom relative range.

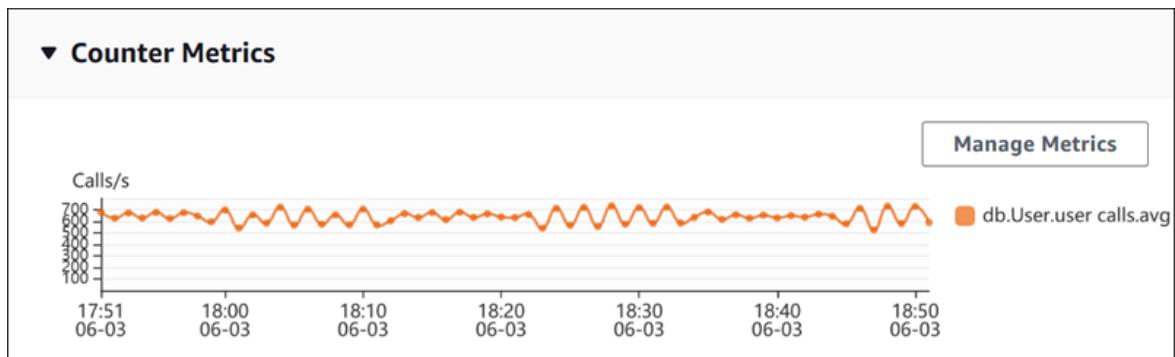
You can select an absolute range with a beginning and ending date and time. The following example shows the time range beginning at midnight on 4/11/22 and ending at 11:59 PM on 4/14/22.

## 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.

This screenshot shows the 'Select metrics shown on the graph' dialog box. It includes a search bar labeled 'Find metrics', a tab for 'Database metrics (1)' which is selected, and a 'Clear all selections' button. The main area lists various metrics categorized by type:

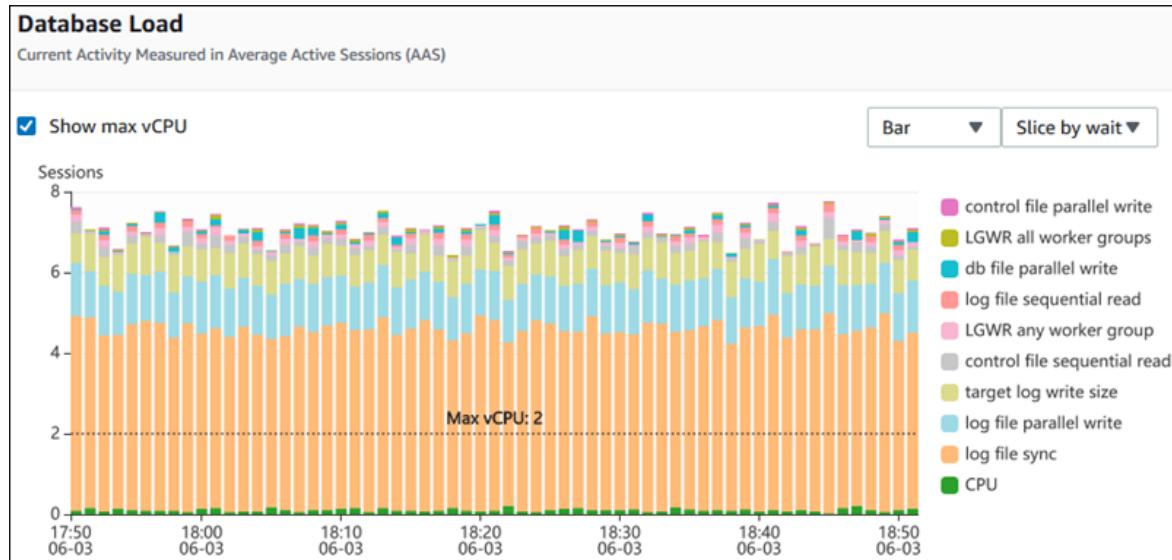
- User:** CPU used by this session, user commits, bytes sent via SQL\*Net to client, SQL\*Net roundtrips to/from client, logons cumulative, user rollbacks, bytes received via SQL\*Net from client, user calls (selected).
- Redo:** redo size
- Cache:** physical read bytes, physical reads, consistent gets, db block gets, consistent gets from cache, DBWR checkpoints, db block gets from cache
- SQL:** parse count (total), sorts (memory), parse count (hard), sorts (disk), table scan rows gotten, sorts (rows)

At the bottom are 'Cancel' and 'Update graph' buttons.

For descriptions of the counter metrics that you can add for each DB engine, see [Performance Insights counter metrics \(p. 546\)](#).

## 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.

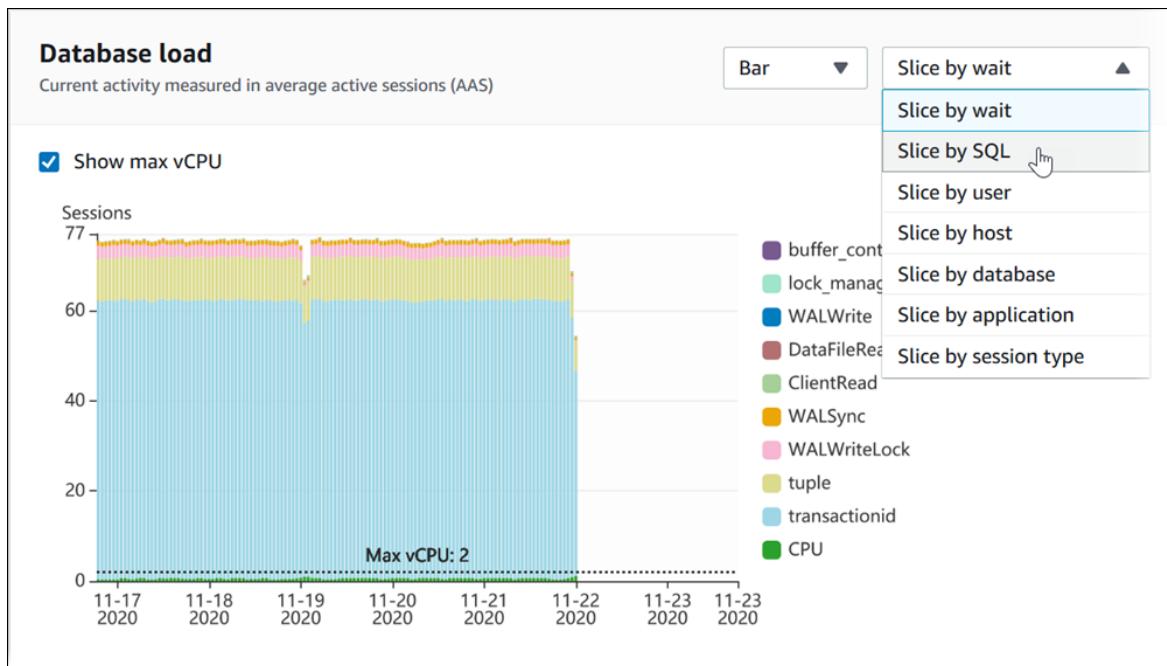


### DB load sliced by dimensions

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.

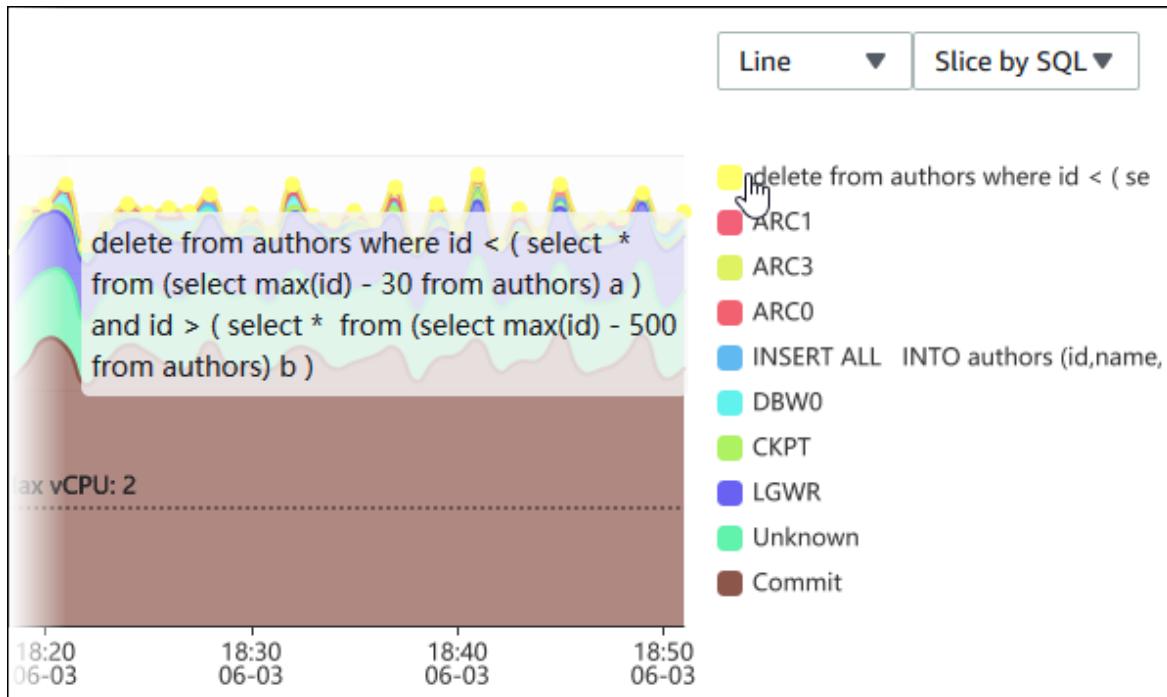
Dimension	Aurora PostgreSQL	Aurora MySQL
Host	Yes	Yes
SQL	Yes	Yes
User	Yes	Yes
Waits	Yes	Yes
Application	Yes	No
Database	Yes	Yes
Session type	Yes	No

The following image shows the dimensions for a PostgreSQL DB instance.

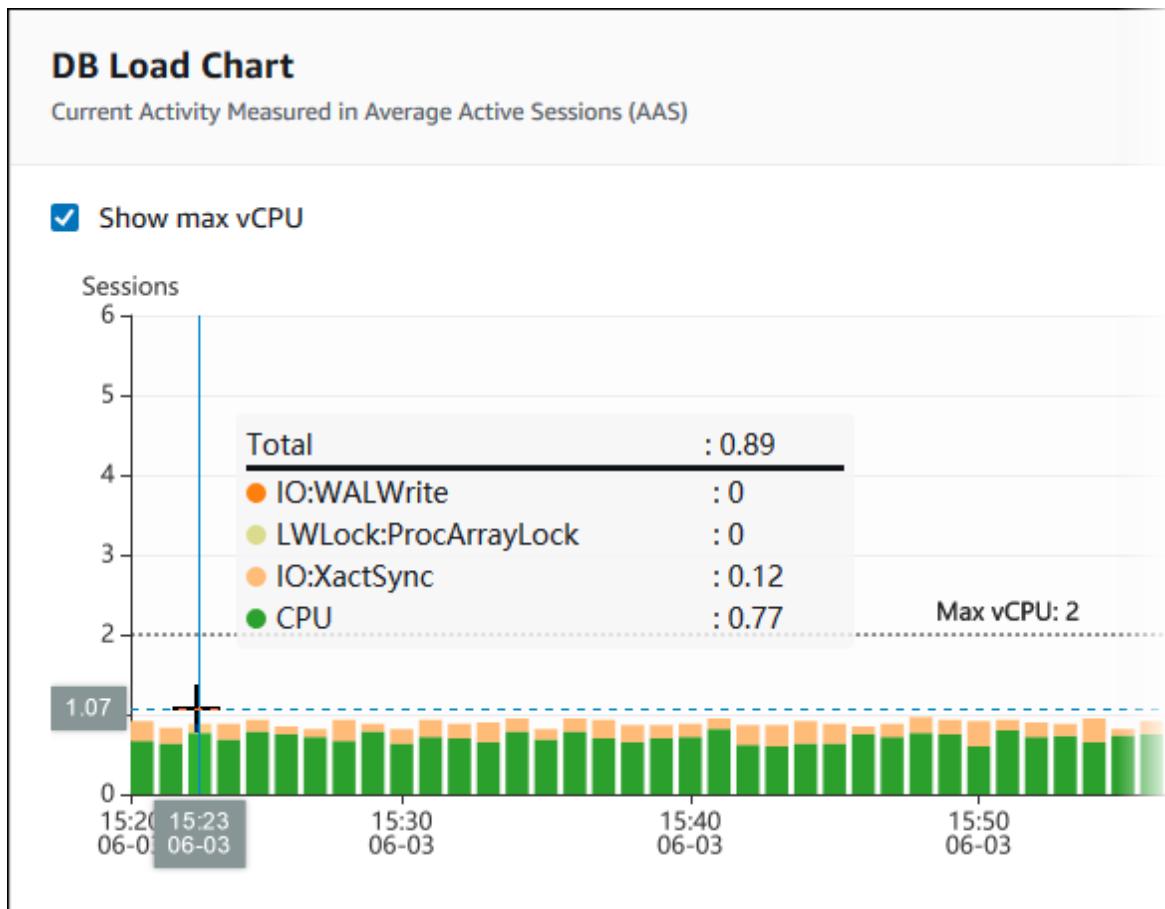


### DB load details for a dimension item

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	<b>Top SQL</b>	Top hosts	Top users	Top connections	Top databases	Top applications	Top session types
<b>Top SQL (0)</b> <a href="#">Learn more</a>							
<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 \(p. 486\)](#).

## Accessing the Performance Insights dashboard

To access 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 left 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 turned on, 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.

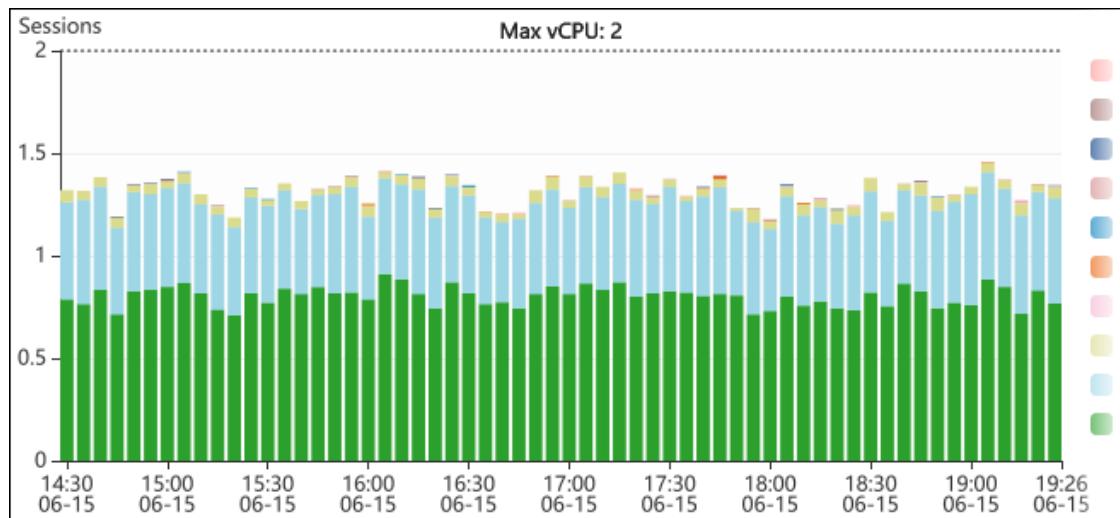
Databases		Group resources	Actions	Restore from S3	Create database
<input type="checkbox"/>	DB identifier	<input checked="" type="radio"/> Filter databases			
<input type="checkbox"/>	database1	MySQL Community	<div style="width: 45.51%;">45.51%</div>	<div style="width: 1.34%;">1.34 Sessions</div>	
<input type="checkbox"/>	database2	Oracle Enterprise Edition	<div style="width: 55.41%;">55.41%</div>	<div style="width: 3.48%;">3.48 Sessions</div>	
<input type="checkbox"/>	database3	Oracle Enterprise Edition	<div style="width: 1.02%;">1.02%</div>	<div style="width: 0%;">0 Connections</div>	

4. (Optional) Choose **View past** in the upper right and specify a different relative or absolute time interval.

In the following example, the DB load is shown for the last 5 hours.

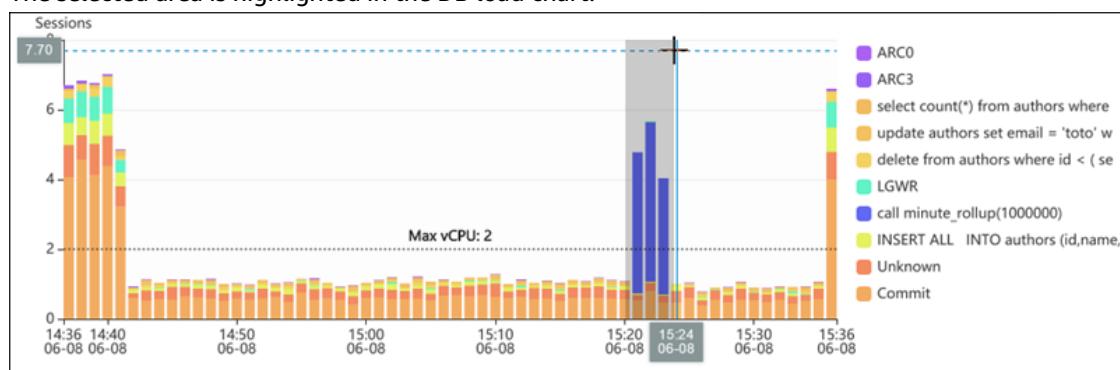


In the following screenshot, the DB load interval is 5 hours.

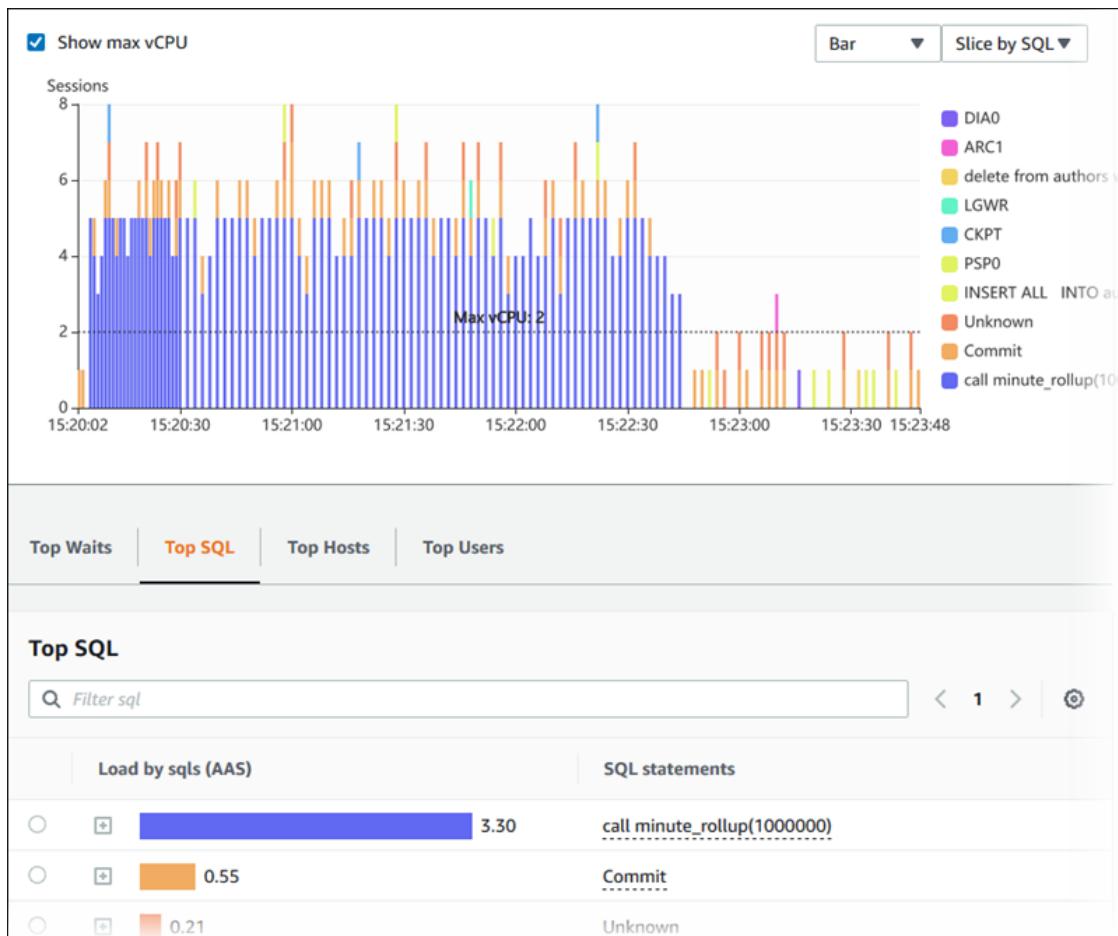


5. (Optional) To zoom in on a portion of the DB load chart, choose the start time and drag to the end of the time period you want.

The selected area is highlighted in the DB load chart.



When you release the mouse, the DB load chart zooms in on the selected AWS Region, and the **Top dimensions** table is recalculated.



6. (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.

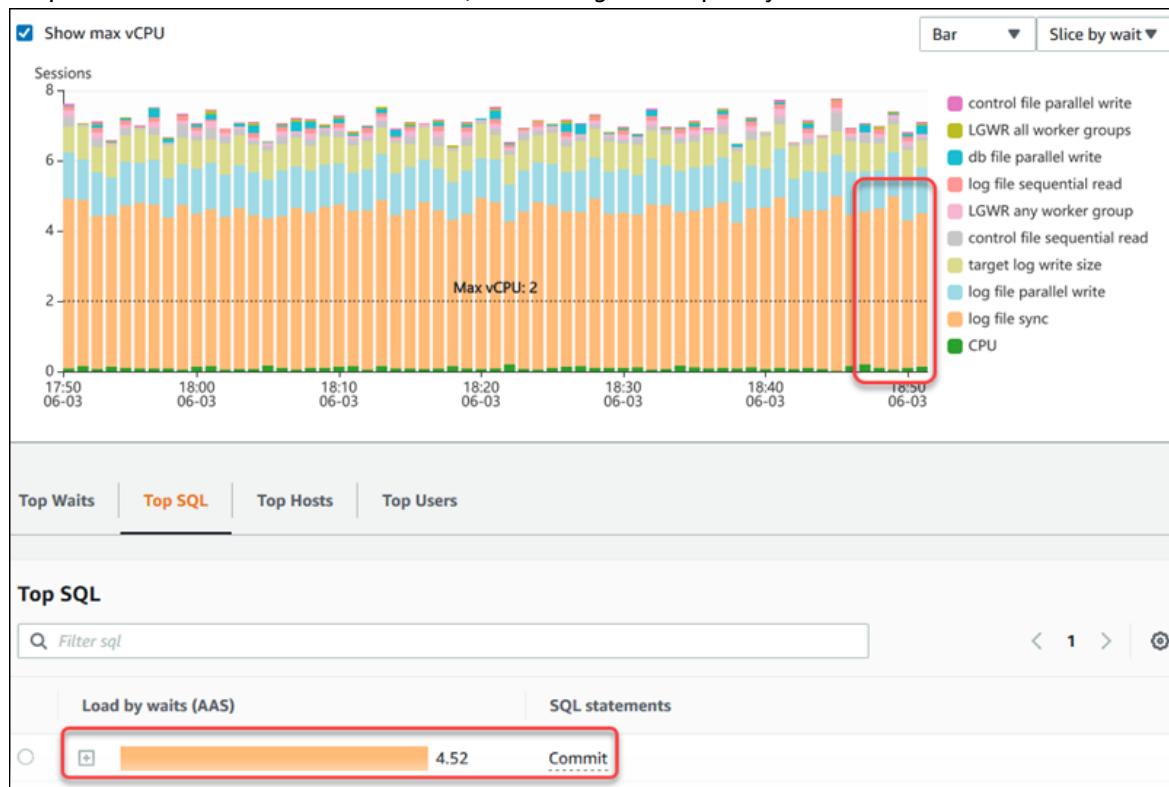
DB load grouped by waits and top SQL queries is the default Performance Insights dashboard view. This combination typically provides the most insight into performance issues. DB load grouped by waits

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 queries in the Performance Insights dashboard

In the Amazon RDS Performance Insights dashboard, you can find information about running and recent queries in the **Top SQL** tab in the **Top dimensions** table. You can use this information to tune your queries.

### Topics

- [Overview of the Top SQL tab \(p. 486\)](#)
- [Accessing more SQL text in the Performance Insights dashboard \(p. 491\)](#)

- [Viewing SQL statistics in the Performance Insights dashboard \(p. 493\)](#)

## 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 text \(p. 486\)](#)
- [SQL statistics \(p. 487\)](#)
- [Load by waits \(AAS\) \(p. 488\)](#)
- [SQL information \(p. 488\)](#)
- [Preferences \(p. 489\)](#)

### SQL text

By default, each row in the **Top SQL** table shows 500 bytes of SQL text for each SQL statement.

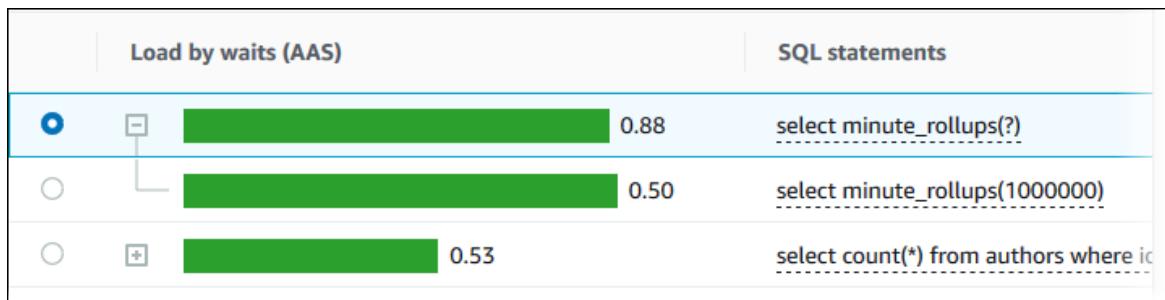
Top SQL (4) <a href="#">Learn more</a>	
<input type="text"/> Find SQL statements	
Load by waits (AAS)	SQL statements
<input type="radio"/> <input checked="" type="checkbox"/> <span style="width: 50%;">&lt; 0.01</span>	autovacuum: ANALYZE public.rds_heartbeat2
<input type="radio"/> <input checked="" type="checkbox"/> <span style="width: 25%;">&lt; 0.01</span>	autovacuum: VACUUM public.rds_heartbeat2
<input type="radio"/> <input checked="" type="checkbox"/> <span style="width: 10%;">&lt; 0.01</span>	autovacuum: VACUUM ANALYZE public.rds_heartbeat2
<input type="radio"/> <input checked="" type="checkbox"/> < 0.01	SELECT name, setting FROM pg_settings WHERE name in (?,?,?,?,?,?,?,?,?,?)

To learn how to see more than the default 500 bytes of SQL text, see [Accessing more SQL text in the Performance Insights dashboard \(p. 491\)](#).

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 = 'Sanchez'  
SELECT * FROM emp WHERE lname = 'Olagappan'  
SELECT * FROM emp WHERE lname = 'Wu'
```

To see the literal SQL statements in a digest, select the query, and then choose the plus symbol (+). In the following example, the selected query is a digest.



#### Note

A SQL digest groups similar SQL statements, but doesn't redact sensitive information.

#### 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.

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			
	Load by waits (AAS)	SQL statements	calls/sec	rows/sec	
0	0.88	select minute_rollups(?)	0.06	0.06	
0	0.53	select count(*) from authors where id < (select max(id) - 31 from authors) and...	33.68	101.04	
0	0.17	WITH cte AS (SELECT id FROM authors LIMIT ?) UPDATE ...	33.68	33.68	
0	0.08	delete from authors where id < (select * from (select max(id) - ? from authors...	33.68	303.13	
0	0.07	INSERT INTO authors (id,name,email) VALUES (nextval(?),? ,? )	33.68	303.13	
0	0.06	select count(*) from authors where id < (select max(id) - 31 from authors) and...	0.00	0.00	

Performance Insights can report 0.00 and – (unknown) for SQL statistics. This situation occurs under the following conditions:

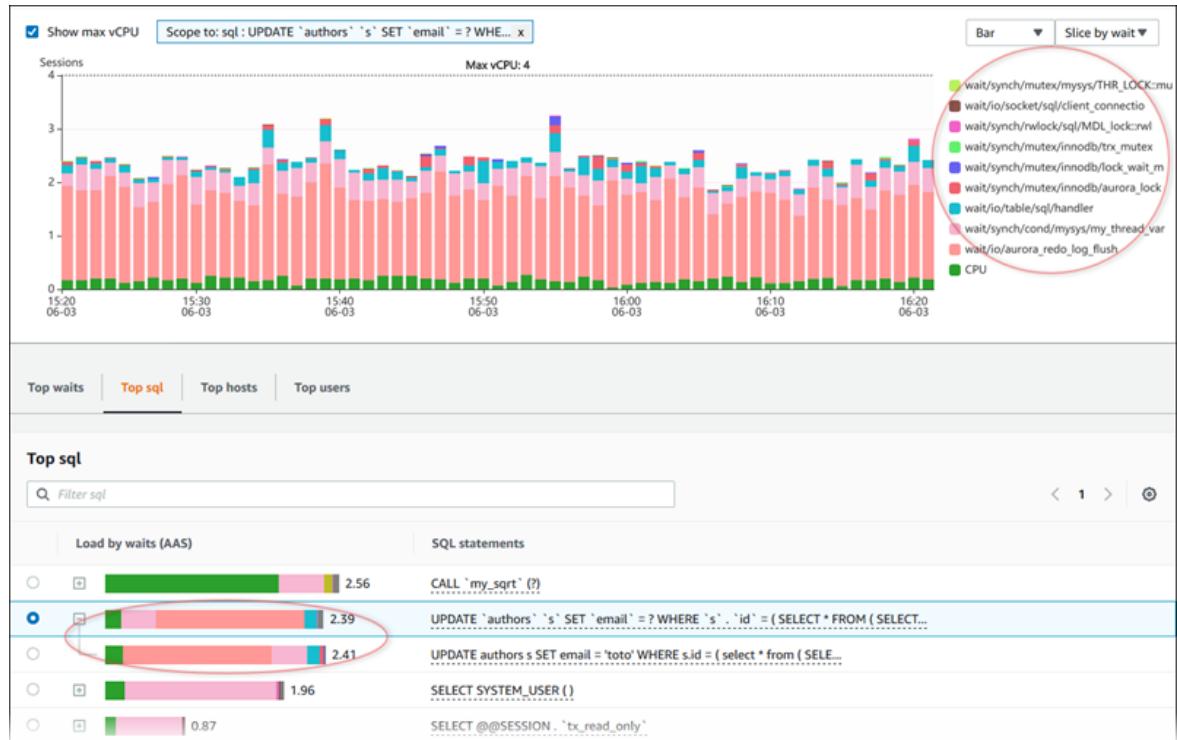
- Only one sample exists. For example, Performance Insights calculates rates of change for Aurora PostgreSQL queries based on multiple samples from the pg\_stats\_statements view. When a workload runs for a short time, Performance Insights might collect only one sample, which means that it can't calculate a rate of change. The unknown value is represented with a dash (-).
- Two samples have the same values. Performance Insights can't calculate a rate of change because no change has occurred, so it reports the rate as 0.00.
- An Aurora PostgreSQL statement lacks a valid identifier. PostgreSQL creates a identifier for a statement only after parsing and analysis. Thus, a statement can exist in the PostgreSQL internal in-memory structures with no identifier. Because Performance Insights samples internal in-memory structures once per second, low-latency queries might appear for only a single sample. If the query identifier isn't available for this sample, Performance Insights can't associate this statement with its statistics. The unknown value is represented with a dash (-).

For a description of the SQL statistics for the Aurora engines, see [SQL statistics for Performance Insights \(p. 554\)](#).

## Load by waits (AAS)

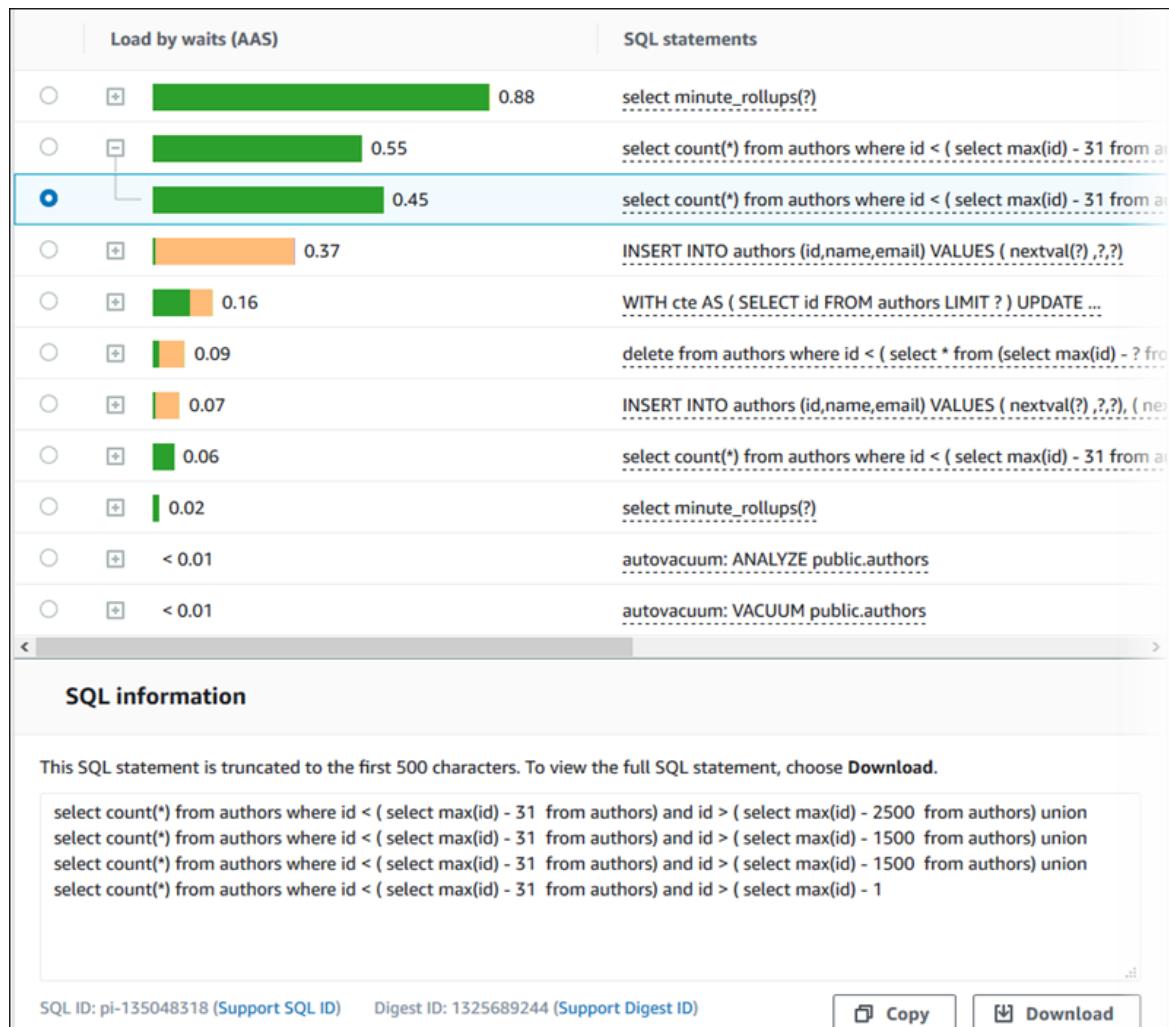
In **Top SQL**, the **Load by waits (AAS)** column illustrates the percentage of the database load associated with each top load item. This column reflects the load for that item by whatever grouping is currently selected in the **DB Load Chart**.

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.

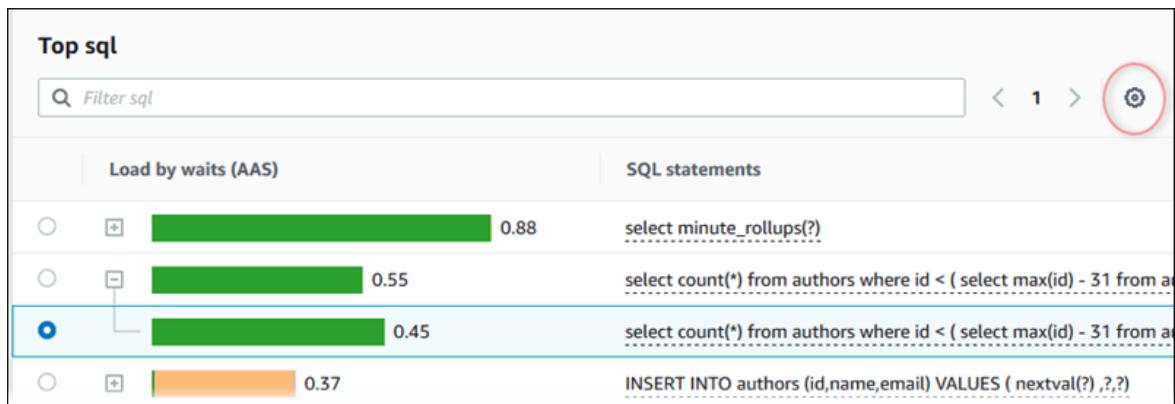


The following types of identifiers (IDs) that are associated with SQL statements:

- **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. The current settings are:

- Page size:** All resources (selected)
- Wrap lines:** Unselected
- Columns:** All metrics listed below are selected (indicated by blue checkboxes).

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)	Unselected
blk hits/sec (shared_blk_hits_per_sec)	Unselected
blk reads/sec (shared_blk_reads_per_sec)	Unselected
blk dirty/sec (shared_blk_dirty_per_sec)	Unselected
blk writes/sec (shared_blk_written_per_sec)	Unselected
local blk hits/sec (local_blk_hits_per_sec)	Unselected
local blk reads/sec (local_blk_reads_per_sec)	Unselected
local blk dirty/sec (local_blk_dirty_per_sec)	Unselected

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**.

## Accessing more SQL text in the Performance Insights dashboard

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 in the **SQL text** section below the **Top SQL** table. In this case, the maximum length for the text displayed in **SQL text** is 4 KB. This limit is introduced by the console and is subject to the limits set by the database engine. To save the text shown in **SQL text**, choose **Download**.

### Topics

- [Text size limits for Aurora MySQL \(p. 491\)](#)
- [Setting the SQL text limit for Aurora PostgreSQL DB instances \(p. 491\)](#)
- [Viewing and downloading SQL text in the Performance Insights dashboard \(p. 492\)](#)

### Text size limits for Aurora MySQL

When you download SQL text, the database engine determines its maximum length. You can download SQL text up to the following per-engine limits.

DB engine	Maximum length of downloaded text
Aurora MySQL 5.7	4,096 bytes
Aurora MySQL 5.6	1,024 bytes

The **SQL text** section of 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 the query in **SQL text** or download it, 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 \(p. 231\)](#).

### Viewing and downloading SQL text in the Performance Insights dashboard

In the Performance Insights dashboard, you can view or download SQL text.

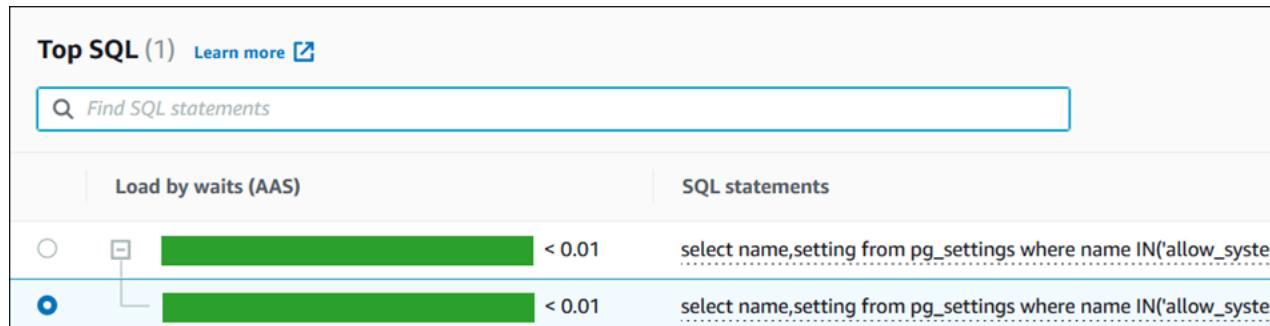
### To view more SQL text in the Performance Insights dashboard

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.

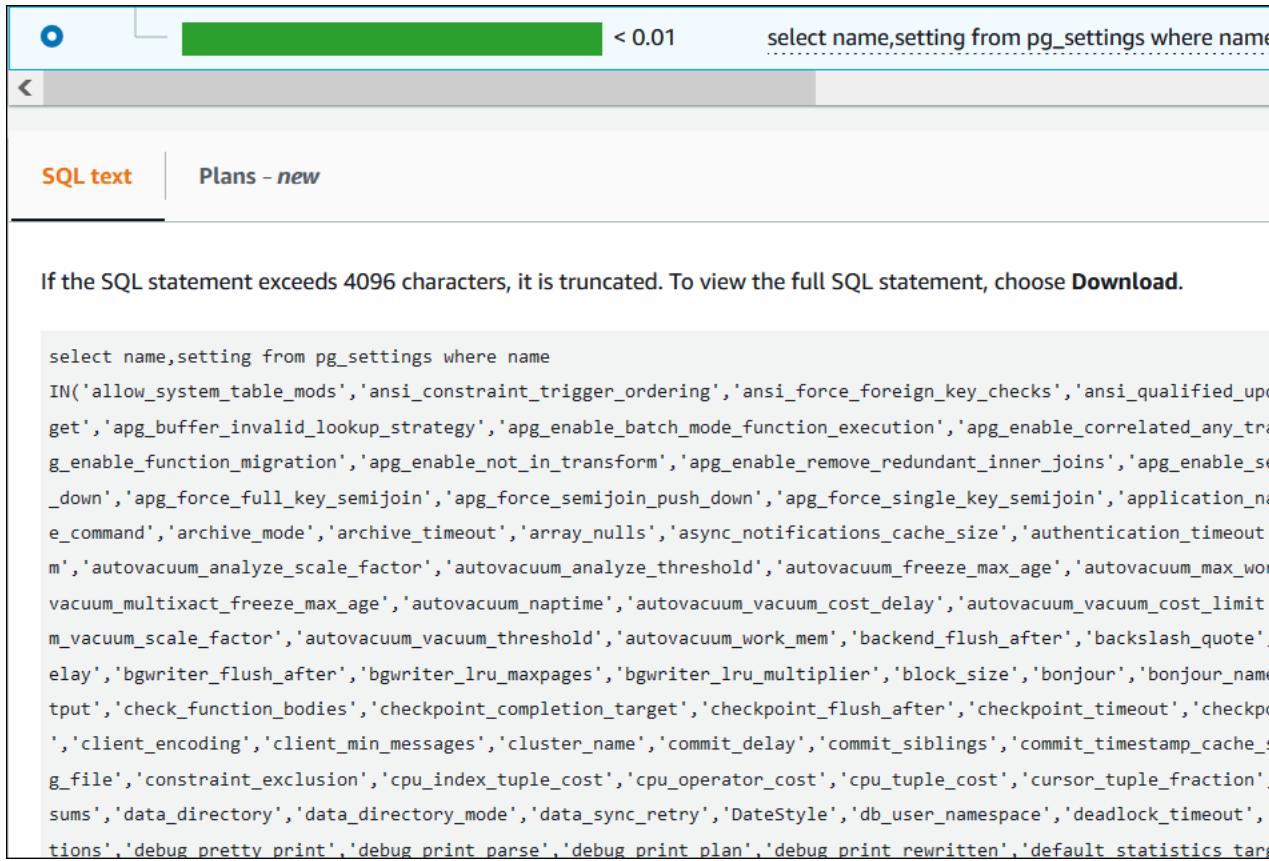
The Performance Insights dashboard is displayed for your DB instance.

4. Scroll down to the **Top SQL** tab.
5. Choose a SQL statement.

SQL statements with text larger than 500 bytes look similar to the following image.



6. Scroll down to the **SQL text** tab.



The screenshot shows a browser window with the Amazon Aurora Performance Insights dashboard. At the top, there's a search bar with the query: "select name,setting from pg\_settings where name". Below the search bar, there are two tabs: "SQL text" (which is selected) and "Plans - new". A note below the tabs states: "If the SQL statement exceeds 4096 characters, it is truncated. To view the full SQL statement, choose Download." The main content area displays a very long SQL query, starting with "select name,setting from pg\_settings where name" and ending with "tions','debug pretty print','debug print parse','debug print plan','debug print rewritten','default statistics tar".

The Performance Insights dashboard can display up to 4,096 bytes for each SQL statement.

7. (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**

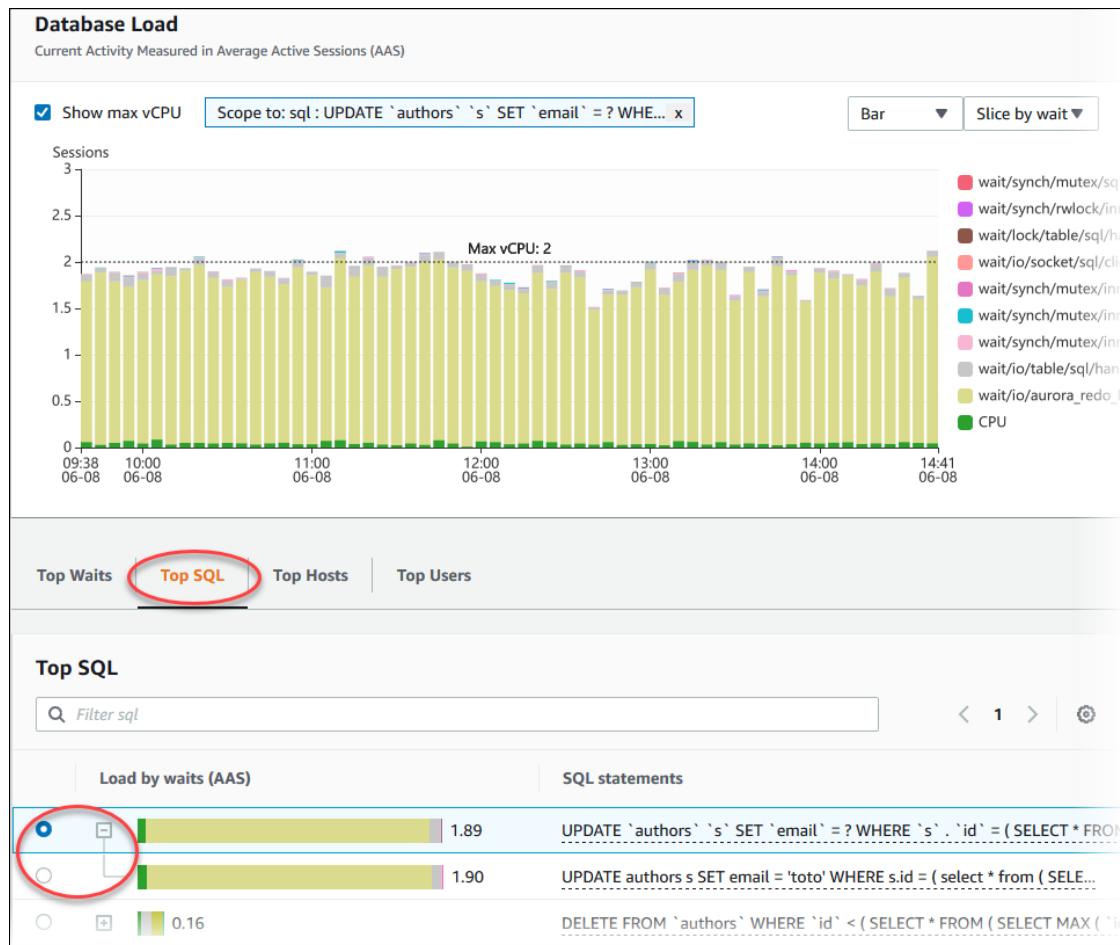
To copy or download the SQL statement, disable pop-up blockers.

## Viewing SQL statistics in the Performance Insights dashboard

In the Performance Insights dashboard, SQL statistics are available in the **Top SQL** tab of the **Database load** chart.

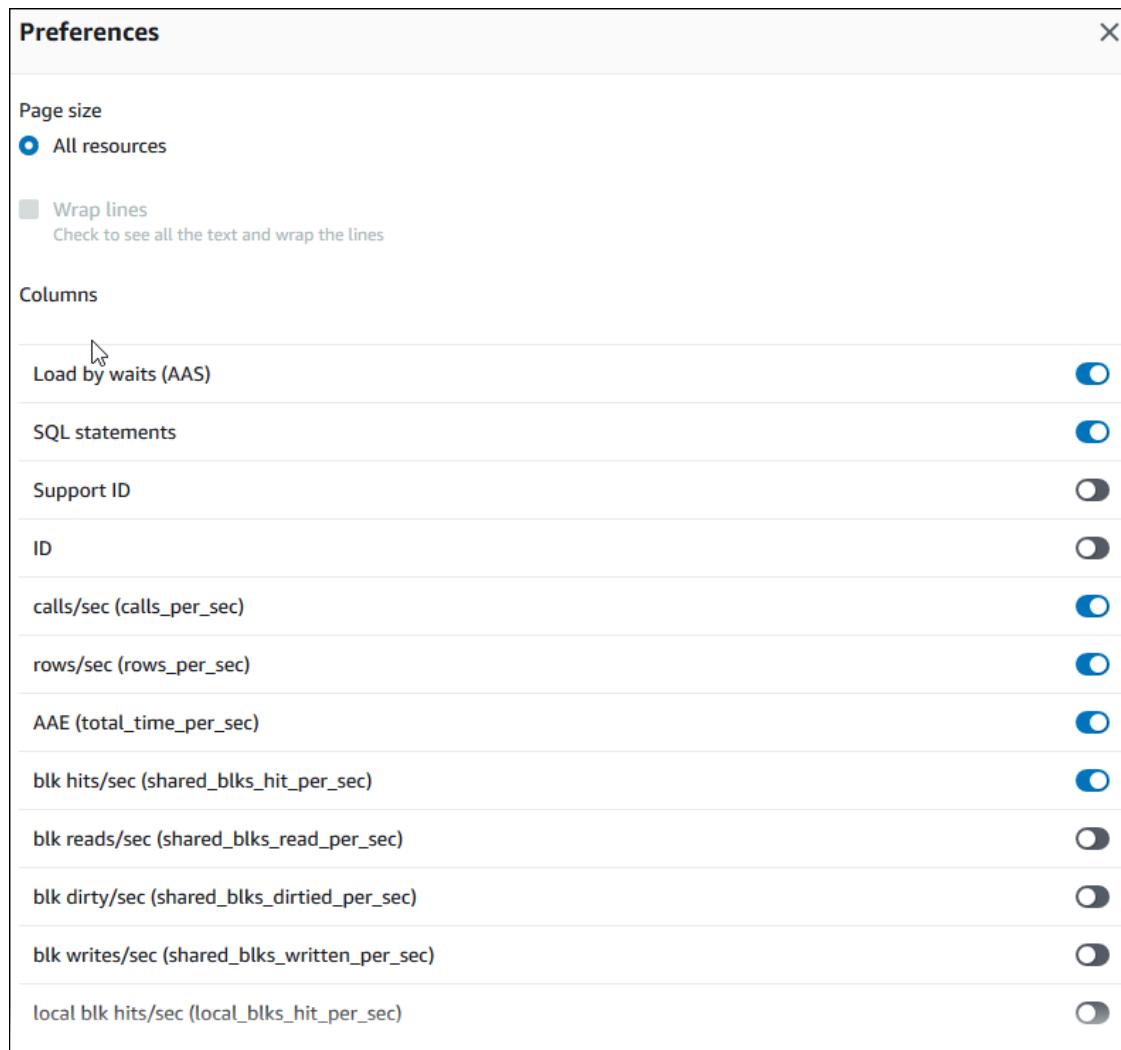
### To view SQL statistics

1. Open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. In the left navigation pane, choose **Performance Insights**.
3. At the top of the page, choose the database whose SQL statistics you want to see.
4. Scroll to the bottom of the page and choose the **Top SQL** tab.
5. Choose an individual statement (Aurora MySQL only) or digest query.

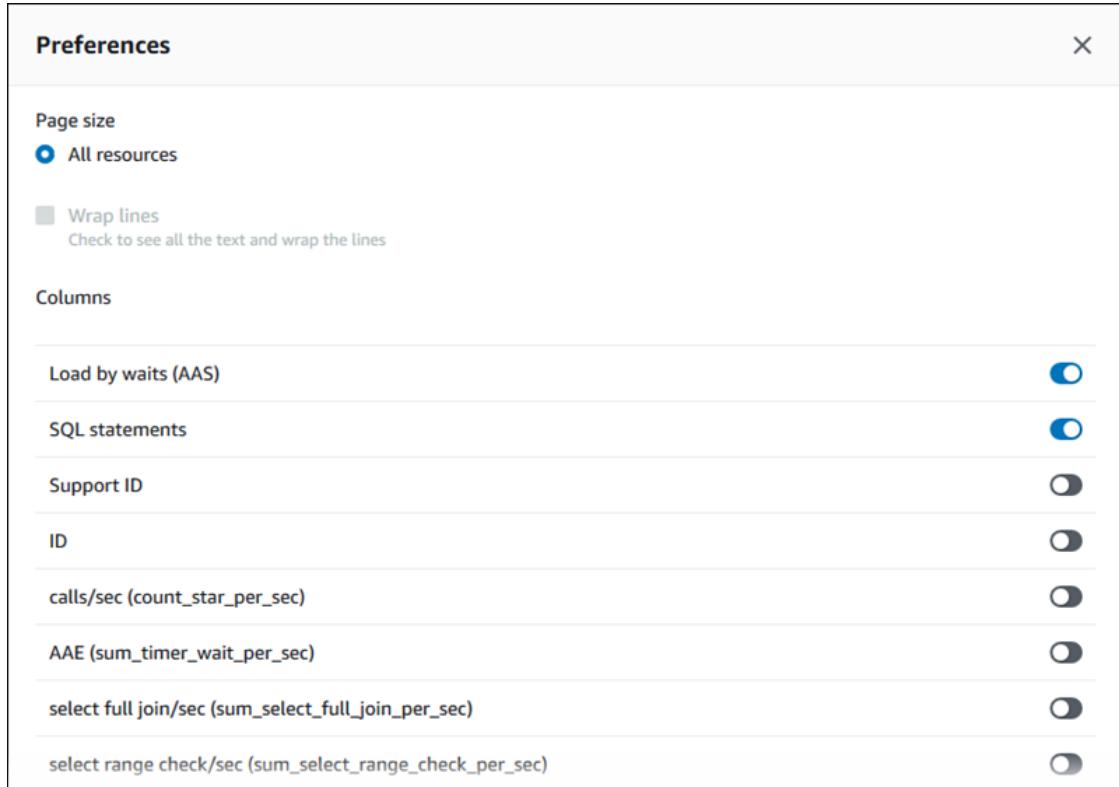


6. Choose which statistics to display by choosing the gear icon in the upper-right corner of the chart. For descriptions of the SQL statistics for the Amazon RDS Aurora engines, see [SQL statistics for Performance Insights \(p. 554\)](#).

The following example shows the preferences for Aurora PostgreSQL.



The following example shows the preferences for Aurora MySQL DB instances.



7. Choose Save to save your preferences.

The **Top SQL** table refreshes.

## 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 [Turning Performance Insights on and off \(p. 466\)](#). 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
<a href="#">DescribeDimensionKeys</a>	<code>aws pi describe-dimension-keys</code>	Retrieves the top N dimension keys for a metric for a specific time period.
<a href="#">GetDimensionKeyDetails</a>	<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.
<a href="#">GetResourceMetadata</a>	<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.
<a href="#">GetResourceMetrics</a>	<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.
<a href="#">ListAvailableResourceDimensions</a>	<code>aws pi list-available-resource-dimensions</code>	Retrieve the dimensions that can be queried for each specified metric type on a specified instance.
<a href="#">ListAvailableResourceMetrics</a>	<code>aws pi list-available-resource-metrics</code>	Retrieve all available metrics of the specified metric types that can be queried for a specified DB instance.

#### Topics

- [AWS CLI for Performance Insights \(p. 497\)](#)
- [Retrieving time-series metrics \(p. 498\)](#)
- [AWS CLI examples for Performance Insights \(p. 499\)](#)

## 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.

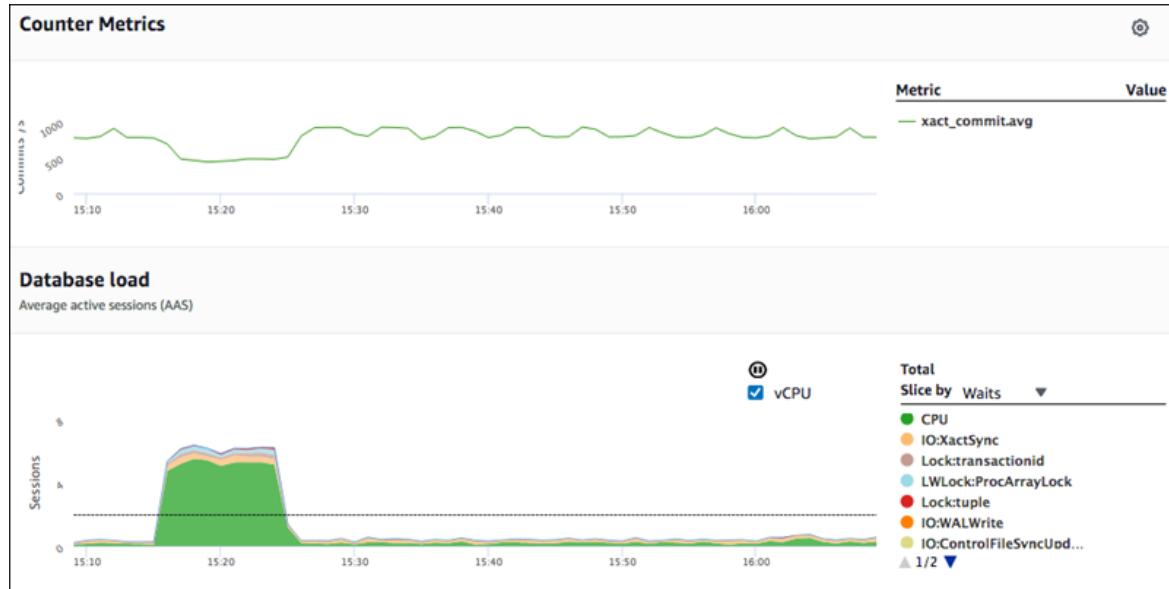
```
aws pi help
```

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 \(p. 546\)](#).

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

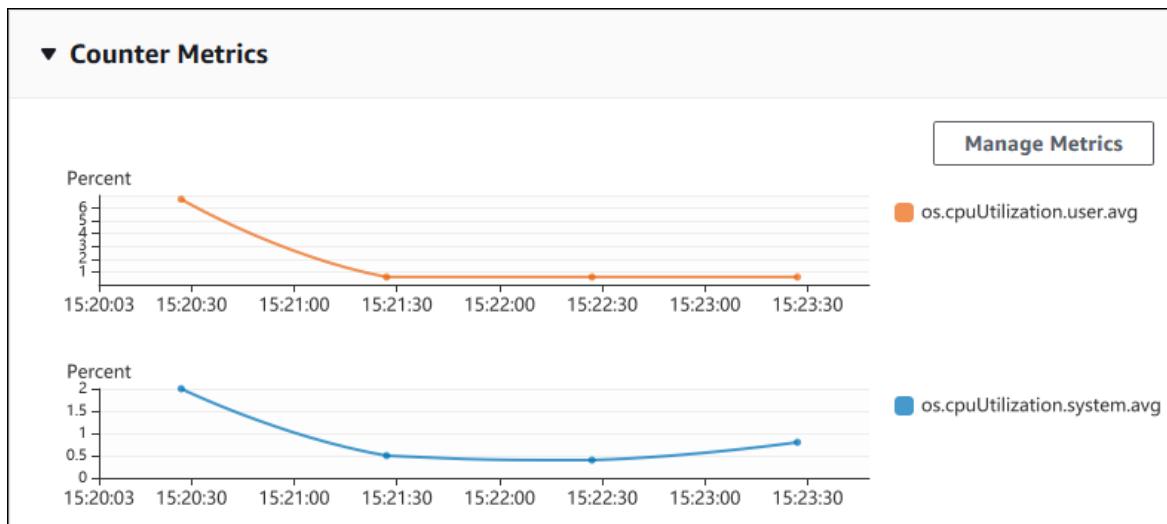
The following examples show how to use the AWS CLI for Performance Insights.

### Topics

- [Retrieving counter metrics \(p. 499\)](#)
- [Retrieving the DB load average for top wait events \(p. 502\)](#)
- [Retrieving the DB load average for top SQL \(p. 504\)](#)
- [Retrieving the DB load average filtered by SQL \(p. 506\)](#)
- [Retrieving the full text of a SQL statement \(p. 509\)](#)

### 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": [
        //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 `DataPoint` 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 \(p. 499\)](#). 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 \(p. 502\)](#). 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 \(p. 504\)](#). 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
                }
            ]
        },
        ...
    ],
    "Identifier": "db-XXX"
}
```

```
{  
    "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": "AKIAI44QH8DHBEEXAMPLE",  
        "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": "9fffbe15c-96b5-4fe6-bed9-9fcfcff1a0525",  
    "eventID": "08908de0-2431-4e2e-ba7b-f5424f908433",  
    "eventType": "AwsApiCall",  
    "recipientAccountId": "123456789012"  
}
```

# Analyzing performance anomalies with Amazon DevOps Guru for Amazon 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*.

The following video is an overview of DevOps Guru for RDS.

For a deep dive on this subject, see [Amazon DevOps Guru for RDS under the hood](#).

## Topics

- [Benefits of DevOps Guru for RDS \(p. 512\)](#)
- [How DevOps Guru for RDS works \(p. 513\)](#)
- [Setting up DevOps Guru for RDS \(p. 513\)](#)

## 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.

##### 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 Amazon RDS to publish insights for an Amazon Aurora database, complete the following tasks.

#### Topics

- [Configuring IAM access policies for DevOps Guru for RDS \(p. 514\)](#)
- [Turning on Performance Insights for your Aurora DB instances \(p. 514\)](#)

- [Turning on DevOps Guru and specifying resource coverage \(p. 514\)](#)

## Configuring IAM access policies for DevOps Guru for RDS

To view alerts from DevOps Guru in the RDS console, your AWS Identity and Access Management (IAM) user or role must have either of the following policies:

- The AWS managed policy `AmazonDevOpsGuruConsoleFullAccess`
- The AWS managed policy `AmazonDevOpsGuruConsoleReadOnlyAccess` and either of the following policies:
  - The AWS managed policy `AmazonRDSFullAccess`
  - A customer managed policy that includes `pi:GetResourceMetrics` and `pi:DescribeDimensionKeys`

For more information, see [Configuring access policies for Performance Insights \(p. 473\)](#).

## Turning on Performance Insights for your 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 an Aurora DB cluster or modify a cluster instance, you can turn on Performance Insights. For more information, see [Turning Performance Insights on and off \(p. 466\)](#).

## Turning on DevOps Guru and specifying resource coverage

You can turn on DevOps Guru to have it monitor your Aurora databases in either of the following ways.

### Topics

- [Turning on DevOps Guru in the RDS console \(p. 514\)](#)
- [Adding Aurora resources on the DevOps Guru console \(p. 517\)](#)

### Turning on DevOps Guru in the RDS console

You can take multiple paths in the Amazon RDS console to turn on DevOps Guru.

### Topics

- [Turning on DevOps Guru when you create an Aurora database \(p. 514\)](#)
- [Turning on DevOps Guru from the notification banner \(p. 515\)](#)
- [Responding to a permissions error when you turn on DevOps Guru \(p. 516\)](#)

### Turning on DevOps Guru when you create an Aurora database

The creation workflow includes a setting that turns on DevOps Guru coverage for your database. This setting is turned on by default when you choose the **Production** template.

#### To turn on DevOps Guru when you create an Aurora database

1. Sign in to the AWS Management Console and open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. Follow the steps in [Creating a DB cluster \(p. 131\)](#), up to but not including the step where you choose monitoring settings.

3. In **Monitoring**, choose **Turn on Performance Insights**. For DevOps Guru for RDS to provide detailed analysis of performance anomalies, Performance Insights must be turned on.
4. Choose **Turn on DevOps Guru**.

**Monitoring**

Turn on Performance Insights [Info](#)

Retention period for Performance Insights [Info](#)

Default (7 days)

AWS KMS Key [Info](#)

(default) aws/rds

Account

123456789012

KMS key ID

1abc2345-de67-8fg9-0123-h45678hij9kl

**⚠ You can't change the KMS key after enabling Performance Insights.**

Turn on DevOps Guru [Info](#)

DevOps Guru for RDS automatically detects performance anomalies for Aurora databases and provides recommendations.

Tag key	Tag value
devops-guru-default	database-1

Cost per resource per hour

\$0.0042 [Amazon DevOps Guru pricing](#)

5. Create a tag for your database so that DevOps Guru can monitor it. Do the following:
  - In the text field for **Tag key**, enter a name that begins with **Devops-Guru-**.
  - In the text field for **Tag value**, enter any value. For example, if you enter **rds-database-1** for the name of your Aurora database, you can also enter **rds-database-1** as the tag value.

For more information about tags, see "[Use tags to identify resources in your DevOps Guru applications](#)" in the *Amazon DevOps Guru User Guide*.

6. Complete the remaining steps in [Creating a DB cluster \(p. 131\)](#).

### Turning on DevOps Guru from the notification banner

If your resources aren't covered by DevOps Guru, Amazon RDS notifies you with a banner in the following locations:

- The **Monitoring** tab of a DB cluster instance
- The Performance Insights dashboard



#### Turn on DevOps Guru - new

DevOps Guru for RDS automatically detects performance anomalies for Aurora databases and provides recommendations. [Learn more](#)

[Turn on DevOps Guru for RDS](#)



### To turn on DevOps Guru for your Aurora database

1. In the banner, choose **Turn on DevOps Guru for RDS**.
2. Enter a tag key name and value. For more information about tags, see "[Use tags to identify resources in your DevOps Guru applications](#)" in the *Amazon DevOps Guru User Guide*.

#### Turn on DevOps Guru for aaclustertest-instance-1



DevOps Guru for RDS automatically detects performance anomalies for Aurora databases and provides recommendations.

To allow DevOps Guru for RDS to monitor a resource, specify a tag.

The tag key must begin with Devops-Guru. [Learn more](#)

Tag key

Devops-Guru-default

Tag value

aaclustertest-instance-1

Cost per resource per hour

\$0.0042 [Amazon DevOps Guru Pricing](#)

By choosing **Turn on DevOps Guru**, you agree to the terms related to use of DevOps Guru in the [AWS Service Terms](#).

Cancel

**Turn on DevOps Guru**

3. Choose **Turn on DevOps Guru**.

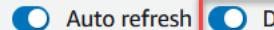
In the Performance Insights dashboard, you can also turn off DevOps Guru.



Last 1 hour



Auto refresh



DevOps Guru for RDS



### Responding to a permissions error when you turn on DevOps Guru

If you turn on DevOps Guru from the RDS console when you create a database, RDS might display the following banner about missing permissions.

Failed to turn on DevOps Guru for database-9-instance-1 because of missing permissions



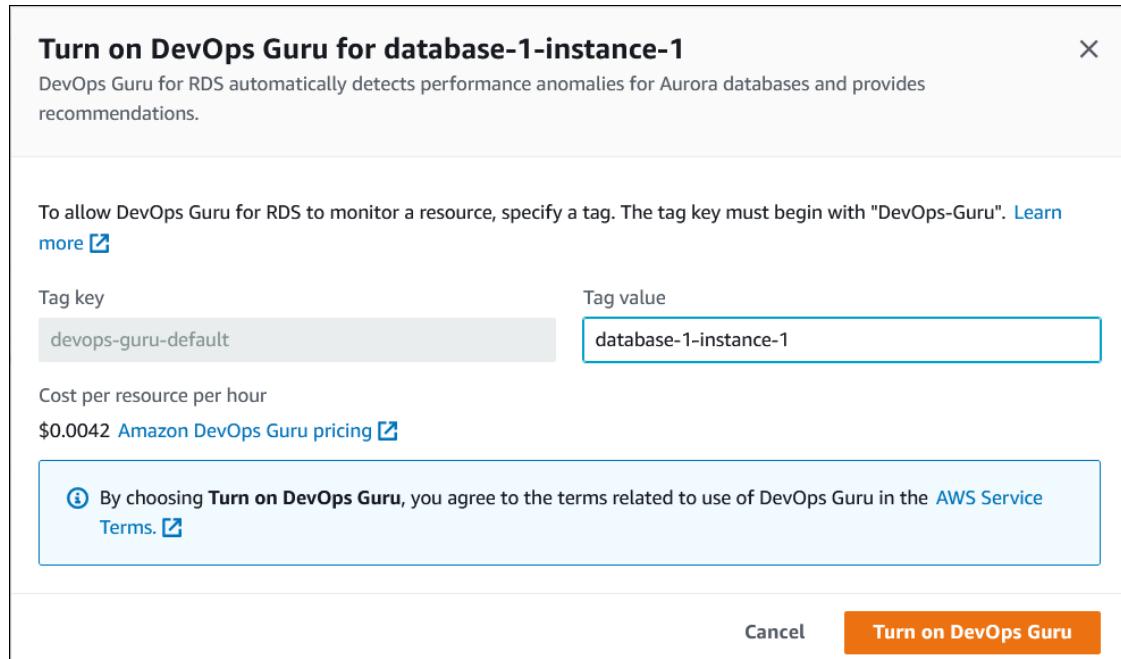
Add permissions to your IAM policy to enable DevOps Guru. [Learn more](#)

### To respond to a permissions error

1. Grant your IAM user or role the user managed role `AmazonDevOpsGuruConsoleFullAccess`. For more information, see [Configuring IAM access policies for DevOps Guru for RDS \(p. 514\)](#).

2. Open the RDS console.
3. In the navigation pane, choose **Performance Insights**.
4. Choose a DB instance in the cluster that you just created.
5. Turn on **DevOps Guru for RDS**.  

6. Choose a tag value. For more information, see "[Use tags to identify resources in your DevOps Guru applications](#)" in the *Amazon DevOps Guru User Guide*.



7. Choose **Turn on DevOps Guru**.

## Adding Aurora resources on the DevOps Guru console

You can specify your DevOps Guru resource coverage on the DevOps Guru console. Follow the step described in [Specify your DevOps Guru resource coverage](#) in the *Amazon DevOps Guru User Guide*. When you edit your analyzed resources, choose one of the following options:

- Choose **All account resources** to analyze all supported resources, including the Aurora databases, in your AWS account and Region.
- Choose **CloudFormation stacks** to analyze the Aurora databases that are in stacks you choose. For more information, see [Use AWS CloudFormation stacks to identify resources in your DevOps Guru applications](#) in the *Amazon DevOps Guru User Guide*.
- Choose **Tags** to analyze the Aurora databases that you have tagged. For more information, see [Use tags to identify resources in your DevOps Guru applications](#) in the *Amazon DevOps Guru User Guide*.

For more information, see [Enable 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 \(p. 518\)](#)
- [Setting up and enabling Enhanced Monitoring \(p. 519\)](#)
- [Viewing OS metrics in the RDS console \(p. 523\)](#)
- [Viewing OS metrics using CloudWatch Logs \(p. 524\)](#)

## 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. For descriptions of the Enhanced Monitoring metrics, see [OS metrics in Enhanced Monitoring \(p. 558\)](#).

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 \(p. 518\)](#)
- [Retention of Enhanced Monitoring metrics \(p. 518\)](#)
- [Cost of Enhanced Monitoring \(p. 519\)](#)

## 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.

For descriptions of the Enhanced Monitoring metrics, see [OS metrics in Enhanced Monitoring \(p. 558\)](#). For more information about CloudWatch metrics, see the [Amazon CloudWatch User Guide](#).

## 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 \(p. 519\)](#)
- [Turning Enhanced Monitoring on and off \(p. 520\)](#)
- [Protecting against the confused deputy problem \(p. 522\)](#)

## 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 \(p. 519\)](#)
- [Creating the IAM role before you enable Enhanced Monitoring \(p. 520\)](#)

### 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, read replica, or Multi-AZ DB cluster.

#### To create the IAM role when enabling Enhanced Monitoring

1. Follow the steps in [Turning Enhanced Monitoring on and off \(p. 520\)](#).
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**.
6. Ensure that the **Permissions policies** shows **AmazonRDSEnhancedMonitoringRole**, and then choose **Next**.
7. 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**.
8. 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 turn on 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 turn on Enhanced Monitoring, you don't need to reboot your DB instance for the change to take effect.

You can turn on 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`.
4. 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.

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 turn on 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 \(p. 519\)](#).

- [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 turns 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
```

## Example

The following example turns on Enhanced Monitoring for a Multi-AZ DB cluster:

For Linux, macOS, or Unix:

```
aws rds modify-db-cluster \
    --db-cluster-identifier mydbcluster \
    --monitoring-interval 30 \
    --monitoring-role-arn arn:aws:iam::123456789012:role/emaccess
```

For Windows:

```
aws rds modify-db-cluster ^
    --db-cluster-identifier mydbcluster ^
    --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 \(p. 519\)](#). 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`.

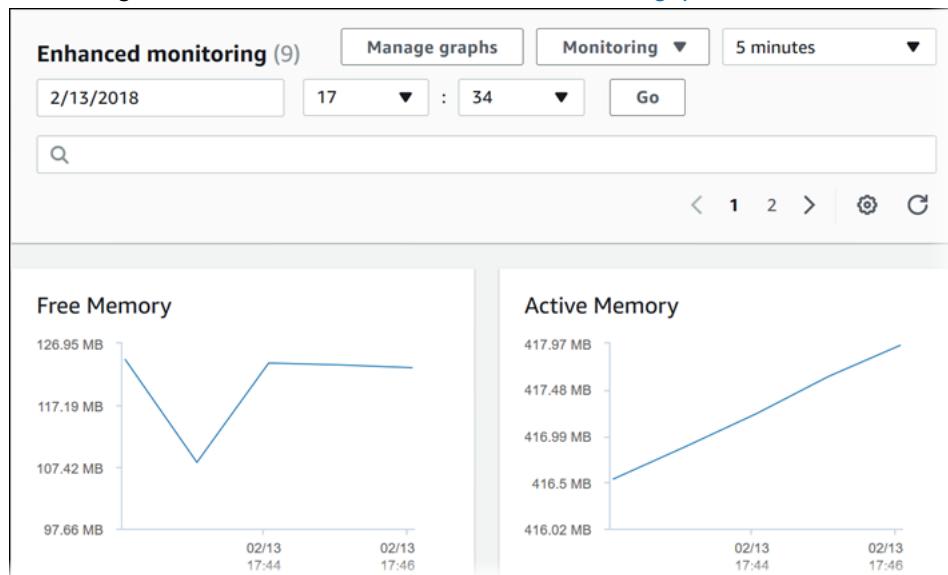
The following example uses the `aws:SourceArn` and `aws:SourceAccount` global condition context keys in a trust policy to prevent the confused deputy problem.

```
{  
    "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 following example shows the Enhanced Monitoring page. For descriptions of the Enhanced Monitoring metrics, see [OS metrics in Enhanced Monitoring \(p. 558\)](#).



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.

The table lists the following processes:

NAME	VIRT	RES	CPU%	MEM%	VMLIMIT
postgres [3181] <sup>t</sup>	283.55 MB	17.11 MB	0.02	1.72	
postgres:					
rdsadmin	384.7 MB	9.51 MB	0.02	0.95	
rdsadmin					
localhost(40156)					
idle [2953] <sup>t</sup>					

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 \(p. 524\)](#).

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 cluster, you can view the metrics for it using CloudWatch Logs, with each log stream representing a single DB instance or DB cluster being monitored. The log stream identifier is the resource identifier (`DbiResourceId`) for the DB instance or DB cluster.

### To view Enhanced Monitoring log data

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. If necessary, choose the AWS Region that your DB cluster 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 \(p. 525\)](#)
- [Amazon CloudWatch dimensions for Aurora \(p. 541\)](#)
- [Availability of Aurora metrics in the Amazon RDS console \(p. 542\)](#)
- [Amazon CloudWatch metrics for Performance Insights \(p. 544\)](#)
- [Performance Insights counter metrics \(p. 546\)](#)
- [SQL statistics for Performance Insights \(p. 554\)](#)
- [OS metrics in Enhanced Monitoring \(p. 558\)](#)

## 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 \(p. 189\)](#). For Aurora parallel query metrics, see [Monitoring parallel query \(p. 806\)](#).

## Topics

- [Cluster-level metrics for Amazon Aurora \(p. 525\)](#)
- [Instance-level metrics for Amazon Aurora \(p. 531\)](#)
- [Amazon CloudWatch usage metrics for Amazon Aurora \(p. 541\)](#)

## 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
AuroraGlobalDBDataTransfer	<a href="#">Aurora Global DB Data Transfer Bytes (Bytes)</a>	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
AuroraGlobalDBReplicatedWrite	<a href="#">Aurora Global DB Replicated Write IO</a>	In an Aurora Global Database, the number	Aurora MySQL	Count

Metric	Console name	Description	Applies to	Units
		<p>of write I/O operations replicated from the 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 <code>VolumeWriteIOPS</code> to account for writes performed within the cluster. The billing calculations for the primary AWS Region in a global database use <code>VolumeWriteIOPS</code> to account for the write activity within that cluster, and <code>AuroraGlobalDBReplicatedWriteIO</code> to account for cross-Region replication within the global database.</p>	and Aurora PostgreSQL	
<code>AuroraGlobalDBReplicationLag</code>	<b>Aurora Global DB Replication Lag (Milliseconds)</b>	For an Aurora Global Database, the amount of lag when replicating updates from the primary AWS Region.	Aurora MySQL and Aurora PostgreSQL	Milliseconds
<code>AuroraGlobalDBRPOLag</code>		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
BacktrackChangeRecordsCreateRate	<b>Backtrack Change Records Creation Rate (Count)</b>	The number of backtrack change records created over 5 minutes for your DB cluster.	Aurora MySQL	Count per 5 minutes
BacktrackChangeRecordsStored	<b>Backtrack Change Records Stored (Count)</b>	The number of backtrack change records used by your DB cluster.	Aurora MySQL	Count

Metric	Console name	Description	Applies to	Units
BackupRetentionPeriodStorageUsed	<b>Backup Retention Period Storage Used (GiB)</b>	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 <a href="#">Understanding Aurora backup storage usage (p. 372)</a> .	Aurora MySQL and Aurora PostgreSQL	Bytes
ServerlessDatabaseCapacity		The current capacity of an Aurora Serverless DB cluster.	Aurora MySQL and Aurora PostgreSQL	Count
SnapshotStorageUsed	<b>Snapshot Storage Used (GiB)</b>	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 <a href="#">Understanding Aurora backup storage usage (p. 372)</a> .	Aurora MySQL and Aurora PostgreSQL	Bytes
TotalBackupStorageBilled	<b>Total Backup Storage Used (GiB)</b>	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 <a href="#">Understanding Aurora backup storage usage (p. 372)</a> .	Aurora MySQL and Aurora PostgreSQL	Bytes

Metric	Console name	Description	Applies to	Units
VolumeBytesUsed	<b>Volume Bytes Used (GiB)</b>	<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 <a href="#">Amazon RDS product page</a>).</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	<b>Volume Read IOPS (Count)</b>	<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 <b>Read operations</b> metric over a 5-minute period. You can determine the amount of billed read operations per second by taking the value of the <b>Billed read operations</b> metric and dividing by 300 seconds. For example, if the <b>Billed read operations</b> returns 13,686, then the billed read operations per second is 45 (<math>13,686 / 300 = 45.62</math>).</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><b>Tip</b> 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	<b>Volume Write IOPS (Count)</b>	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 <a href="#">VolumeReadIOPs</a> .	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	<b>Active Transactions (Count)</b>	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	<b>Aurora Binlog Replica Lag (Seconds)</b>	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"> <li>A high value           <ul style="list-style-type: none"> <li>The replica is lagging the replication source.</li> <li>0 or a value close to 0</li> <li>The replica process is active and current.</li> <li>-1</li> <li>Aurora can't determine the lag, which can happen during replica setup or when the replica is in an error state.</li> </ul> </li> <li>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.</li> <li>For more information about administering replication, see <a href="#">Replicating Amazon Aurora MySQL DB clusters across AWS Regions (p. 831)</a>.</li> </ul>		

Metric	Console Name	Description	Applies to	Units
		For more information about troubleshooting, see <a href="#">Amazon Aurora MySQL replication issues (p. 1766)</a> .		
AuroraReplicaLag	<b>Aurora Replica Lag (Milliseconds)</b>	For an Aurora replica, the amount of lag when replicating updates from the primary instance.	Replica for Aurora MySQL and Aurora PostgreSQL	Milliseconds
AuroraReplicaLagMaximum	<b>Replica Lag Maximum (Milliseconds)</b>	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	<b>Replica Lag Minimum (Milliseconds)</b>	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	<b>Backtrack Window Actual (Minutes)</b>	The difference between the target backtrack window and the actual backtrack window.	Primary for Aurora MySQL	Minutes
BacktrackWindowAlert	<b>Backtrack Window Alert (Count)</b>	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	<b>Blocked Transactions (Count)</b>	The average number of transactions in the database that are blocked per second.	Aurora MySQL	Count per second
BufferCacheHitRatio	<b>Buffer Cache Hit Ratio (Percent)</b>	The percentage of requests that are served by the buffer cache.	Aurora MySQL and Aurora PostgreSQL	Percentage
CommitLatency	<b>Commit Latency (Milliseconds)</b>	The average duration of commit operations.	Aurora MySQL and Aurora PostgreSQL	Milliseconds
CommitThroughput	<b>Commit Throughput (Count/Second)</b>	The average number of commit operations per second.	Aurora MySQL and Aurora PostgreSQL	Count per second

Metric	Console Name	Description	Applies to	Units
ConnectionAttempts	<b>Connection Attempts (Count)</b>	The number of attempts to connect to an instance, whether successful or not.	Aurora MySQL and Aurora PostgreSQL	Count
CPUCreditBalance	<b>CPU Credit Balance (Count)</b>	<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> <p>Launch credits work the same way in Amazon RDS as they do in Amazon EC2. For more information, see <a href="#">Launch credits</a> in the <i>Amazon Elastic Compute Cloud User Guide for Linux Instances</i>.</p>	Aurora MySQL and Aurora PostgreSQL	Count
CPUCreditUsage	<b>CPU Credit Usage (Count)</b>	<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	<b>CPU Utilization (Percent)</b>	The percentage of CPU used by an Aurora DB instance.	Aurora MySQL and Aurora PostgreSQL	Percentage

Metric	Console Name	Description	Applies to	Units
DatabaseConnections	<b>DB Connections (Count)</b>	<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"> <li>• Sessions that no longer have a network connection but which the database hasn't cleaned up</li> <li>• Sessions created by the database engine for its own purposes</li> <li>• Sessions created by the database engine's parallel execution capabilities</li> <li>• Sessions created by the database engine job scheduler</li> <li>• Amazon Aurora connections</li> </ul>	Aurora MySQL and Aurora PostgreSQL	Count
DDLLatency	<b>DDL Latency (Milliseconds)</b>	The average duration of requests such as example, create, alter, and drop requests.	Aurora MySQL	Milliseconds
DDLThroughput	<b>DDL (Count/Second)</b>	The average number of DDL requests per second.	Aurora MySQL	Count per second
Deadlocks	<b>Deadlocks (Count)</b>	The average number of deadlocks in the database per second.	Aurora MySQL and Aurora PostgreSQL	Count per second
DeleteLatency	<b>Delete Latency (Milliseconds)</b>	The average duration of delete operations.	Aurora MySQL	Milliseconds
DeleteThroughput	<b>Delete Throughput (Count/Second)</b>	The average number of delete queries per second.	Aurora MySQL	Count per second
DiskQueueDepth	<b>Queue Depth (Count)</b>	The number of outstanding read/write requests waiting to access the disk.	Aurora PostgreSQL	Count

Metric	Console Name	Description	Applies to	Units
DMLLatency	<b>DML Latency (Milliseconds)</b>	The average duration of inserts, updates, and deletes.	Aurora MySQL	Milliseconds
DMLThroughput	<b>DML Throughput (Count/Second)</b>	The average number of inserts, updates, and deletes per second.	Aurora MySQL	Count per second
EBSByteBalance%	<b>EBS Byte Balance (Percent)</b>	<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 <a href="#">EBS optimized by default</a> 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%	<b>EBS IO Balance (Percent)</b>	<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 <a href="#">EBS optimized by default</a> 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 <a href="#">Improving application performance and reducing costs with Amazon EBS-Optimized Instance burst capability</a>.</p>	Aurora MySQL and Aurora PostgreSQL	Percent
EngineUptime	<b>Engine Uptime (Seconds)</b>	The amount of time that the instance has been running.	Aurora MySQL and Aurora PostgreSQL	Seconds

Metric	Console Name	Description	Applies to	Units
FreeableMemory	<b>Freeable Memory (MB)</b>	The amount of available random access memory.	Aurora MySQL and Aurora PostgreSQL	Bytes
FreeLocalStorage	<b>Free Local Storage (Bytes)</b>	<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 <a href="#">Amazon RDS product page</a>). You can increase the amount of free storage space for an instance by choosing a larger DB instance class for your instance.</p> <p>(This doesn't apply to Aurora Serverless v2.)</p>	Aurora MySQL and Aurora PostgreSQL	Bytes
InsertLatency	<b>Insert Latency (Milliseconds)</b>	The average duration of insert operations.	Aurora MySQL	Milliseconds
InsertThroughput	<b>Insert Throughput (Count/Second)</b>	The average number of insert operations per second.	Aurora MySQL	Count per second
LoginFailures	<b>Login Failures (Count)</b>	The average number of failed login attempts per second.	Aurora MySQL	Count per second
MaximumUsedTransactionID	<b>MaximumUsedTransactionID</b>	<p>The age of the oldest unvacuumed transaction ID, in transactions. If this value reaches 2,146,483,648 (<math>2^{31} - 1,000,000</math>), the database is forced into read-only mode, to avoid transaction ID wraparound. For more information, see <a href="#">Preventing transaction ID wraparound failures</a> in the PostgreSQL documentation.</p>	Aurora PostgreSQL	Count

Metric	Console Name	Description	Applies to	Units
NetworkReceiveThroughput	<b>Network Receive Throughput (MB/Second)</b>	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	<b>Network Throughput (Byte/Second)</b>	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	<b>Network Transmit Throughput (MB/Second)</b>	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	<b>Queries (Count/Second)</b>	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	<b>Read IOPS (Count/Second)</b>	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	<b>Read Latency (Milliseconds)</b>	The average amount of time taken per disk I/O operation.	Aurora MySQL and Aurora PostgreSQL	Seconds
ReadThroughput	<b>Read Throughput (MB/Second)</b>	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	<b>Result Set Cache Hit Ratio (Percent)</b>	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	<b>Select Latency (Milliseconds)</b>	The average amount of time for select operations.	Aurora MySQL	Milliseconds
SelectThroughput	<b>Select Throughput (Count/Second)</b>	The average number of select queries per second.	Aurora MySQL	Count per second
StorageNetworkReceiveThroughput	<b>Network Receive Throughput (MB/Second)</b>	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	<b>Network Throughput (Byte/Second)</b>	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	<b>Network Transmit Throughput (MB/Second)</b>	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	<b>Swap Usage (MB)</b>	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	<b>Transaction Logs Disk Usage (MB)</b>	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	<b>Update Latency (Milliseconds)</b>	The average amount of time taken for update operations.	Aurora MySQL	Milliseconds
UpdateThroughput	<b>Update Throughput (Count/Second)</b>	The average number of updates per second.	Aurora MySQL	Count per second
WriteIOPS	<b>Volume Write IOPS (Count)</b>	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	<b>Write Latency (Milliseconds)</b>	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	<b>Write Throughput (MB/Second)</b>	The average number of bytes written to persistent storage every second.	Aurora PostgreSQL	Bytes per second

## Amazon CloudWatch usage metrics for Amazon Aurora

The AWS/Usage namespace in Amazon CloudWatch includes account-level usage metrics for your Amazon RDS service quotas. CloudWatch collects usage metrics automatically for all AWS Regions except Asia Pacific (Jakarta).

For more information, see [CloudWatch usage metrics](#) in the *Amazon CloudWatch User Guide*. For more information about quotas, see [Quotas and constraints for Amazon Aurora \(p. 1756\)](#) and [Requesting a quota increase](#) in the *Service Quotas User Guide*.

Metric	Description	Units*
DBClusterParameterGroups	The number of DB cluster parameter groups in your AWS account. The count excludes default parameter groups.	Count
DBClusters	The number of Amazon Aurora DB clusters in your AWS account.	Count
DBInstances	The number of DB instances in your AWS account.	Count
DBParameterGroups	The number of DB parameter groups in your AWS account. The count excludes the default DB parameter groups.	Count
DBSubnetGroups	The number of DB subnet groups in your AWS account. The count excludes the default subnet group.	Count
ManualClusterSnapshot	The number of manually created DB cluster snapshots in your AWS account. The count excludes invalid snapshots.	Count
OptionGroups	The number of option groups in your AWS account. The count excludes the default option groups.	Count
ReservedDBInstances	The number of reserved DB instances in your AWS account. The count excludes retired or declined instances.	Count

\* Amazon RDS doesn't publish units for usage metrics to CloudWatch. The units only appear in the documentation.

## 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 Aurora DB cluster.

Dimension	Filters the requested data for . . .
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.
DbClusterIdentifier, EngineName	A specific Aurora DB cluster and engine name combination. For example, you can view the VolumeReadIOPs metric for cluster ams1 and engine aurora.
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 aurora-postgresql.
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 \(p. 542\)](#)
- [Aurora metrics available in specific cases \(p. 543\)](#)
- [Aurora metrics that aren't available in the console \(p. 544\)](#)

### 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 BlockedTransactions BufferCacheHitRatio CommitLatency CommitThroughput DatabaseConnections

Category	Metrics
	<b>DDL</b> DDLLatency DDLThroughput Deadlocks <b>DML</b> DMLLatency DMLThroughput LoginFailures ResultSetCacheHitRatio <b>Select</b> SelectLatency SelectThroughput
<b>System</b>	AuroraReplicaLag AuroraReplicaLagMaximum AuroraReplicaLagMinimum CPUCreditBalance CPUCreditUsage CPUUtilization FreeableMemory FreeLocalStorage (This doesn't apply to Aurora Serverless v2.) NetworkReceiveThroughput
<b>Deployment</b>	AuroraReplicaLag BufferCacheHitRatio ResultSetCacheHitRatio SelectThroughput

## 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

Metric	Display name
AuroraReplicaLagMinimum	Replica lag minimum
DDLThroughput	DDL
NetworkReceiveThroughput	Network throughput
VolumeBytesUsed	[Billed] Volume bytes used
VolumeReadIOPS	[Billed] Volume read IOPS
VolumeWriteIOPS	[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

Metric	Description
	number of active sessions. In Performance Insights, this data is queried as db.load.avg.
DBLoadCPU	The number of active sessions where the wait event type is CPU. In Performance Insights, this data is queried as db.load.avg, 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": 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 \(p. 546\)](#)
- [Performance Insights counters for Aurora MySQL \(p. 548\)](#)
- [Performance Insights counters for Aurora PostgreSQL \(p. 552\)](#)

## Performance Insights operating system counters

The following operating system counters, which are prefixed with os, are available with Performance Insights for Aurora PostgreSQL. You can find definitions for these metrics in [Viewing OS metrics using CloudWatch Logs \(p. 524\)](#).

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
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

Counter	Type	Metric
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
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

Counter	Type	Metric
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
acuUtilization	general	os.general.acuUtilization
maxConfiguredAcu	general	os.general.maxConfiguredAcu
minConfiguredAcu	general	os.general.minConfiguredAcu
numVCpus	general	os.general.numVCpus
serverlessDatabaseCapacity	general	os.general.serverlessDatabaseCapacity

## Performance Insights counters for Aurora MySQL

The following database counters are available with Performance Insights for Aurora MySQL.

### Topics

- [Native counters for Aurora MySQL \(p. 549\)](#)
- [Non-native counters for Aurora MySQL \(p. 550\)](#)

## Native counters for Aurora MySQL

Native metrics are defined by the database engine and not by Amazon Aurora. 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
Queries	SQL	Queries per second	db.SQL.Queries
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

Counter	Type	Unit	Metric
Sort_scan	SQL	Queries per second	db.SQL.Sort_scan
Total_query_time	SQL	Milliseconds	db.SQL.Total_query_time
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
Connections	Users	Connections	db.Users.Connections
External_threads_connected	Users	Connections	db.Users.External_threads_connected
Threads_connected	Users	Connections	db.Users.Threads_connected
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)
innodb_buffer_pool_usage	Cache	db.Cache.innodb_buffer_pool_usage	The percentage of storage InnoDB buffer pool that contains data (pages).	Innodb_buffer_pool_pages_data / Innodb_buffer_pool_pages_total * 100.0
			<b>Note</b> When using compressed tables, this value can vary. For more information, see the information about <a href="#">Innodb_buffer_pool_pages_data</a> and <a href="#">Innodb_buffer_pool_pages_total</a> in <a href="#">Server Status Variables</a> in the MySQL documentation.	
query_cache_hit_rate	Cache	db.Cache.query_cache_hit_rate	The hit rate 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_METRICS 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

Counter	Type	Metric	Description	Definition
				INFORMATION_SCHEMA.INNODB_METRICS WHERE NAME='lock_timeouts'
innodb_row_lock_waits	Locks	db.Locks.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_METRICS 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 \(p. 552\)](#)
- [Non-native counters for Aurora PostgreSQL \(p. 553\)](#)

### Native Counters for Aurora PostgreSQL

Native metrics are defined by the database engine and not by Amazon Aurora. You can find definitions for these native metrics in [Viewing Statistics](#) in the PostgreSQL documentation.

Counter	Type	Unit	Metric
queries_started	SQL	Queries per second	db.SQL.queries
total_query_time	SQL	Milliseconds	db.SQL.total_query_time
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
blks_hit	Cache	Blocks per second	db.Cache.blks_hit
buffers_alloc	Cache	Blocks per second	db.Cache.buffers_alloc
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

Counter	Type	Unit	Metric
maxwritten_clean	Checkpoint	Bgwriter clean stops per minute	db.Checkpoint.maxwritten_clean
deadlocks	Concurrency	Deadlocks per minute	db.Concurrency.deadlocks
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
idle_in_transaction_aborted	State	Sessions	db.State.idle_in_transaction_aborted
idle_in_transaction_count	State	Sessions	db.State.idle_in_transaction_count
idle_in_transaction_max_time	State	Seconds	db.State.idle_in_transaction_max_time
temp_bytes	Temp	Bytes per second	db.Temp.temp_bytes
temp_files	Temp	Files per minute	db.Temp.temp_files
active_transactions	Transactions	Transactions	db.Transactions.active_transactions
blocked_transactions	Transactions	Transactions	db.Transactions.blocked_transactions
duration_commits	Transactions	Milliseconds	db.Transactions.duration_commits
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
numbackends	User	Connections	db.User.numbackends
total_auth_attempts	User	Connections	db.User.total_auth_attempts
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
logical_reads	SQL	db.SQL.logical_reads	The total number of blocks hit and read.	blks_hit + blks_read
checkpoint_sync_time	Checkpoint	db.Checkpoint.checkpoint_sync_time	The time spent in milliseconds that has been spent in the portion of checkpoint	checkpoint_sync_time / (checkpoints_timed + checkpoints_req)

Counter	Type	Metric	Description	Definition
			processing where files are synchronized to disk.	
checkpoint_write_time	Checkpoints	db.Checkpoint.duration	The total amount of time 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
commit_latency	Transactions	db.Transactions.duration	The total duration of commit operations.	db.Transactions.duration_commits / db.Transactions.xact_commit

## SQL statistics for Performance Insights

*SQL statistics* are performance-related metrics about SQL queries that are collected by Performance Insights. Performance Insights gathers statistics for each second that a query is running and for each SQL call. All engines support statistics for digest queries. All engines support statement-level statistics except for Aurora PostgreSQL.

### Topics

- [SQL statistics for Aurora MySQL \(p. 554\)](#)
- [SQL statistics for Aurora PostgreSQL \(p. 556\)](#)

## SQL statistics for Aurora MySQL

Aurora MySQL collect SQL statistics only at the digest level. No statistics are shown at the statement level.

### Topics

- [Digest statistics for Aurora MySQL \(p. 554\)](#)
- [Per-second statistics for Aurora MySQL \(p. 555\)](#)
- [Per-call statistics for Aurora MySQL \(p. 555\)](#)

## Digest statistics for 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 [Turning on the Performance Schema for Performance Insights on Aurora MySQL \(p. 469\)](#).

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
<code>db.sql_tokenized.stats.sum_sort_range_per_sec</code>	Sort ranges per second
<code>db.sql_tokenized.stats.sum_sort_rows_per_sec</code>	Sort rows per second
<code>db.sql_tokenized.stats.sum_rows_affected_per_sec</code>	Rows affected per second
<code>db.sql_tokenized.stats.sum_rows_examined_per_sec</code>	Rows examined per second
<code>db.sql_tokenized.stats.sum_rows_sent_per_sec</code>	Rows sent per second
<code>db.sql_tokenized.stats.sum_created_tmp_disk_tables_per_sec</code>	Created temporary disk tables per second
<code>db.sql_tokenized.stats.sum_created_tmp_tables_per_sec</code>	Created temporary tables per second
<code>db.sql_tokenized.stats.sum_lock_time_per_sec</code>	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
<code>db.sql_tokenized.stats.sum_timer_wait_per_call</code>	Average latency per call (in ms)

Metric	Unit
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_per_call	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)

## SQL statistics for Aurora PostgreSQL

For each SQL call and for each second that a query runs, Performance Insights collects SQL statistics. For other Aurora engines, statistics are collected at the statement-level and the digest-level. However, for Aurora PostgreSQL, Performance Insights collects SQL statistics at the digest-level only.

A *SQL digest* is a composite of all queries having a given pattern but not necessarily having the same literal values. The digest replaces literal values with a question mark. For example, `SELECT * FROM emp WHERE lname = ?` is an example digest. This digest might consist of the following child queries:

```
SELECT * FROM emp WHERE lname = 'Sanchez'
SELECT * FROM emp WHERE lname = 'Olagappan'
SELECT * FROM emp WHERE lname = 'Wu'
```

Following, you can find information about digest-level statistics for Aurora PostgreSQL.

### Topics

- [Digest statistics for Aurora PostgreSQL \(p. 556\)](#)
- [Per-second digest statistics for Aurora PostgreSQL \(p. 557\)](#)
- [Per-call digest statistics for Aurora PostgreSQL \(p. 557\)](#)

## 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 parameter groups \(p. 215\)](#).

### 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
<code>db.sql_tokenized.stats.calls_per_sec</code>	Calls per second
<code>db.sql_tokenized.stats.rows_per_sec</code>	Rows per second
<code>db.sql_tokenized.stats.total_time_per_sec</code>	Average active executions per second (AAE)
<code>db.sql_tokenized.stats.shared_blk_hit_per_sec</code>	Block hits per second
<code>db.sql_tokenized.stats.shared_blk_read_per_sec</code>	Block reads per second
<code>db.sql_tokenized.stats.shared_blk_dirtied_per_sec</code>	Blocks dirtied per second
<code>db.sql_tokenized.stats.shared_blk_written_per_sec</code>	Block writes per second
<code>db.sql_tokenized.stats.local_blk_hit_per_sec</code>	Local block hits per second
<code>db.sql_tokenized.stats.local_blk_read_per_sec</code>	Local block reads per second
<code>db.sql_tokenized.stats.local_blk_dirtied_per_sec</code>	Local block dirty per second
<code>db.sql_tokenized.stats.local_blk_written_per_sec</code>	Local block writes per second
<code>db.sql_tokenized.stats.temp_blk_written_per_sec</code>	Temporary writes per second
<code>db.sql_tokenized.stats.temp_blk_read_per_sec</code>	Temporary reads per second
<code>db.sql_tokenized.stats.blk_read_time_per_sec</code>	Average concurrent reads per second
<code>db.sql_tokenized.stats.blk_write_time_per_sec</code>	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
<code>db.sql_tokenized.stats.rows_per_call</code>	Rows per call
<code>db.sql_tokenized.stats.avg_latency_per_call</code>	Average latency per call (in ms)
<code>db.sql_tokenized.stats.shared_blk_hit_per_call</code>	Block hits per call
<code>db.sql_tokenized.stats.shared_blk_read_per_call</code>	Block reads per call
<code>db.sql_tokenized.stats.shared_blk_written_per_call</code>	Block writes per call
<code>db.sql_tokenized.stats.shared_blk_dirtied_per_call</code>	Blocks dirtied per call

Metric	Unit
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
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.

## 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 \(p. 558\)](#)

## 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.
	instanceReserveID	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.

<b>Group</b>	<b>Metric</b>	<b>Console name</b>	<b>Description</b>
	idle	<b>CPU Idle</b>	The percentage of CPU that is idle.
	irq	<b>CPU IRQ</b>	The percentage of CPU in use by software interrupts.
	nice	<b>CPU Nice</b>	The percentage of CPU in use by programs running at lowest priority.
	steal	<b>CPU Steal</b>	The percentage of CPU in use by other virtual machines.
	system	<b>CPU System</b>	The percentage of CPU in use by the kernel.
	total	<b>CPU Total</b>	The total percentage of the CPU in use. This value includes the nice value.
	user	<b>CPU User</b>	The percentage of CPU in use by user programs.
	wait	<b>CPU Wait</b>	The percentage of CPU unused while waiting for I/O access.
diskIO	avgQueueLen	<b>Avg Queue Size</b>	The number of requests waiting in the I/O device's queue.
	avgReqSz	<b>Ave Request Size</b>	The average request size, in kilobytes.
	await	<b>Disk I/O Await</b>	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	<b>Read IO/s</b>	The number of read operations per second.
	readKb	<b>Read Total</b>	The total number of kilobytes read.
	readKbPS	<b>Read Kb/s</b>	The number of kilobytes read per second.
	readLatency	<b>Read Latency</b>	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	<b>Read Throughput</b>	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	<b>Rrqms</b>	The number of merged read requests queued per second.
	tps	<b>TPS</b>	The number of I/O transactions per second.
	util	<b>Disk I/O Util</b>	The percentage of CPU time during which requests were issued.
	writeIOsPS	<b>Write IO/s</b>	The number of write operations per second.
	writeKb	<b>Write Total</b>	The total number of kilobytes written.
	writeKbPS	<b>Write Kb/s</b>	The number of kilobytes written per second.

Group	Metric	Console name	Description
	writeLatency	<b>Write Latency</b>	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	<b>Write Throughput</b>	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	<b>Wrqms</b>	The number of merged write requests queued per second.
fileSys	maxFiles	<b>Max Inodes</b>	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	<b>Total Filesystem</b>	The total number of disk space available for the file system, in kilobytes.
	used	<b>Used Filesystem</b>	The amount of disk space used by files in the file system, in kilobytes.
	usedFilePercent	<b>Used %</b>	The percentage of available files in use.
	usedFiles	<b>Used Inodes</b>	The number of files in the file system.
	usedPercent	<b>Used Inodes %</b>	The percentage of the file-system disk space in use.
loadAverage	fifteen	<b>Load Avg 15 min</b>	The number of processes requesting CPU time over the last 15 minutes.
	five	<b>Load Avg 5 min</b>	The number of processes requesting CPU time over the last 5 minutes.
	one	<b>Load Avg 1 min</b>	The number of processes requesting CPU time over the last minute.
memory	active	<b>Active Memory</b>	The amount of assigned memory, in kilobytes.
	buffers	<b>Buffered Memory</b>	The amount of memory used for buffering I/O requests prior to writing to the storage device, in kilobytes.
	cached	<b>Cached Memory</b>	The amount of memory used for caching file system-based I/O.
	dirty	<b>Dirty Memory</b>	The amount of memory pages in RAM that have been modified but not written to their related data block in storage, in kilobytes.
	free	<b>Free Memory</b>	The amount of unassigned memory, in kilobytes.

Group	Metric	Console name	Description
memory	hugePagesFree	<b>Huge Pages Free</b>	The number of free huge pages. Huge pages are a feature of the Linux kernel.
	hugePagesRsvd	<b>Huge Pages Rsvd</b>	The number of committed huge pages.
	hugePagesSize	<b>Huge Pages Size</b>	The size for each huge pages unit, in kilobytes.
	hugePagesSurp	<b>Huge Pages Surp</b>	The number of available surplus huge pages over the total.
	hugePagesTotal	<b>Huge Pages Total</b>	The total number of huge pages.
	inactive	<b>Inactive Memory</b>	The amount of least-frequently used memory pages, in kilobytes.
	mapped	<b>Mapped Memory</b>	The total amount of file-system contents that is memory mapped inside a process address space, in kilobytes.
	pageTables	<b>Page Tables</b>	The amount of memory used by page tables, in kilobytes.
	slab	<b>Slab Memory</b>	The amount of reusable kernel data structures, in kilobytes.
	total	<b>Total Memory</b>	The total amount of memory, in kilobytes.
network	writeback	<b>Writeback Memory</b>	The amount of dirty pages in RAM that are still being written to the backing storage, in kilobytes.
	interface	Not applicable	The identifier for the network interface being used for the DB instance.
	rx	<b>RX</b>	The number of bytes received per second.
	tx	<b>TX</b>	The number of bytes uploaded per second.
	cpuUsedPc	<b>CPU %</b>	The percentage of CPU used by the process.
	id	Not applicable	The identifier of the process.
	memoryUsedPc	<b>MEM%</b>	The percentage of memory used by the process.
processList	name	Not applicable	The name of the process.
	parentID	Not applicable	The process identifier for the parent process of the process.
	rss	<b>RES</b>	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.

<b>Group</b>	<b>Metric</b>	<b>Console name</b>	<b>Description</b>
	vss	<b>VIRT</b>	The amount of virtual memory allocated to the process, in kilobytes.
swap	swap	<b>Swap</b>	The amount of swap memory available, in kilobytes.
	swap_in	<b>Swaps in</b>	The amount of memory, in kilobytes, swapped in from disk.
	swap_out	<b>Swaps out</b>	The amount of memory, in kilobytes, swapped out to disk.
	free	<b>Free Swap</b>	The amount of swap memory free, in kilobytes.
	committed	<b>Committed Swap</b>	The amount of swap memory, in kilobytes, used as cache memory.
tasks	blocked	<b>Tasks Blocked</b>	The number of tasks that are blocked.
	running	<b>Tasks Running</b>	The number of tasks that are running.
	sleeping	<b>Tasks Sleeping</b>	The number of tasks that are sleeping.
	stopped	<b>Tasks Stopped</b>	The number of tasks that are stopped.
	total	<b>Tasks Total</b>	The total number of tasks.
	zombie	<b>Tasks Zombie</b>	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 \(p. 427\)](#) 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 \(p. 563\)](#)
- [Monitoring Amazon Aurora events \(p. 568\)](#)
- [Monitoring Amazon Aurora log files \(p. 597\)](#)
- [Monitoring Amazon Aurora API calls in AWS CloudTrail \(p. 615\)](#)
- [Monitoring Amazon Aurora with Database Activity Streams \(p. 619\)](#)

## 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 \(p. 569\)](#).
- **Logs** – Shows database log files generated by a DB instance in your Aurora cluster. For more information, see [Monitoring Amazon Aurora log files \(p. 597\)](#).

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 top navigation bar includes 'RDS > Databases > apga'. On the right, there are 'Modify' and 'Actions ▾' buttons. Below the navigation, the cluster name 'apg' is displayed. A 'Related' section contains a search bar labeled 'Filter by databases'. The main table lists the cluster structure:

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, tabs for 'Connectivity & security', 'Monitoring', 'Logs & events' (which is selected), 'Configuration', 'Maintenance & backups', and 'Tags' are visible.

4. Scroll down and choose **Configuration**.

The following example shows the status of the database activity streams for your cluster.

Configuration	Maintenance & backups	Tags
<b>Availability</b>	<b>Encryption</b>	
IAM DB authentication	Encryption	
Not enabled	Enabled	
Master username	AWS KMS key	
apga_admin	<a href="#">aws/rds</a> 	
Master password	<b>Database activity stream</b>	
*****	Status	
Multi-AZ	Stopped	
3 Zones	<b>Published logs</b>	
	CloudWatch Logs	
	<a href="#">PostgreSQL</a>	

5. Choose **Logs & events**.

The Logs & events section appears.

The screenshot shows the 'Logs & events' tab selected in the top navigation bar. The interface is divided into 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 shows a message: "No auto scaling activities found".
- Recent events (3)**: A table with columns for Time and System notes. It includes a search bar, pagination, and a dropdown menu. The events listed are:
  - February 03, 2022, 5:12:34 PM UTC: Started failover to DB instance: apga-instance-1-us-east-1c

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)**: An empty table with columns 'Name' and 'State'. It includes a 'Create alarm' button.

**Recent events (0)**: An empty table with columns 'Time' and 'System notes'. It displays the message 'No events found.'

**Logs (29)**: A table listing log files. The columns are 'Name', 'Last written', and 'Logs'. The data is as follows:

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. For descriptions of the Aurora events, see [Amazon RDS event categories and event messages \(p. 590\)](#).

## Topics

- [Overview of events for Aurora \(p. 568\)](#)
- [Viewing Amazon RDS events \(p. 569\)](#)
- [Working with Amazon RDS event notification \(p. 572\)](#)
- [Creating a rule that triggers on an Amazon Aurora event \(p. 587\)](#)
- [Amazon RDS event categories and event messages \(p. 590\)](#)

## 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 the following resources:

- DB clusters
  - For a list of cluster events, see [DB cluster events \(p. 590\)](#).
- DB instances
  - For a list of DB instance events, see [DB instance events \(p. 592\)](#).
- DB parameter groups
  - For a list of DB parameter group events, see [DB parameter group events \(p. 595\)](#).
- DB security groups
  - For a list of DB security group events, see [DB security group events \(p. 595\)](#).
- DB cluster snapshots
  - For a list of DB cluster snapshot events, see [DB cluster snapshot events \(p. 595\)](#).
- RDS Proxy events
  - For a list of RDS Proxy events, see [RDS Proxy events \(p. 596\)](#).

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

## Viewing Amazon RDS events

You can retrieve the following event information for your Amazon Aurora resources:

- Resource name
- Resource type
- Time of the event
- Message summary of the event

Access the events through the AWS Management Console, which shows events from the past 24 hours. You can also retrieve events 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 \(p. 587\)](#)

For descriptions of the Amazon Aurora events, see [Amazon RDS event categories and event messages \(p. 590\)](#).

To access detailed information about events using AWS CloudTrail, including request parameters, see

[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 \[Viewing events with CloudTrail event history\]\(#\).](#)

(p. ).

## Console

### To view all Amazon RDS 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. (Optional) Enter a search term to filter your results.

The following example shows a list of events filtered by the characters **apg**.

Events (34)			
Source	Type	Time	Message
apg134a-instance-1-snap-04-20-22	Cluster snapshots	April 20, 2022, 3:30:36 PM UTC	Manual cluster snapshot created
apg134a-instance-1-snap-04-20-22	Cluster snapshots	April 20, 2022, 3:27:01 PM UTC	Creating manual cluster snapshot
apg134a-instance-1-us-east-1d	Instances	April 20, 2022, 3:16:07 PM UTC	Performance Insights has been enabled

## AWS CLI

To view all events generated in the last hour, call [describe-events](#) with no parameters.

```
aws rds describe-events
```

The following sample output shows that a DB cluster instance has started recovery.

```
{
  "Events": [
    {
      "EventCategories": [
        "recovery"
      ],
      "SourceType": "db-instance",
      "SourceArn": "arn:aws:rds:us-east-1:123456789012:db:mycluster-instance-1",
      "Date": "2022-04-20T15:02:38.416Z",
      "Message": "Recovery of the DB instance has started. Recovery time will vary with the amount of data to be recovered.",
      "SourceIdentifier": "mycluster-instance-1"
    },
    ...
  ]
}
```

To view all Amazon RDS events for the past 10080 minutes (7 days), call the [describe-events](#) AWS CLI command and set the --duration parameter to 10080.

```
aws rds describe-events --duration 10080
```

The following example shows the events in the specified time range for DB instance *test-instance*.

```
aws rds describe-events \
--source-identifier test-instance \
--source-type db-instance \
--start-time 2022-03-13T22:00Z \
--end-time 2022-03-13T23:59Z
```

The following sample output shows the status of a backup.

```
{
  "Events": [
    {
      "SourceType": "db-instance",
      "SourceIdentifier": "test-instance",
      "EventCategories": [
        "backup"
      ],
      "Message": "Backing up DB instance",
      "Date": "2022-03-13T23:09:23.983Z",
      "SourceArn": "arn:aws:rds:us-east-1:123456789012:db:test-instance"
    },
    {
      "SourceType": "db-instance",
      "SourceIdentifier": "test-instance",
      "EventCategories": [
        "backup"
      ],
      "Message": "Finished DB Instance backup",
      "Date": "2022-03-13T23:15:13.049Z",
      "SourceArn": "arn:aws:rds:us-east-1:123456789012:db:test-instance"
    }
  ]
}
```

## 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.

## Working with 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 \(p. 572\)](#)
- [Granting permissions to publish notifications to an Amazon SNS topic \(p. 577\)](#)
- [Subscribing to Amazon RDS event notification \(p. 578\)](#)
- [Listing Amazon RDS event notification subscriptions \(p. 581\)](#)
- [Modifying an Amazon RDS event notification subscription \(p. 582\)](#)
- [Adding a source identifier to an Amazon RDS event notification subscription \(p. 583\)](#)
- [Removing a source identifier from an Amazon RDS event notification subscription \(p. 584\)](#)
- [Listing the Amazon RDS event notification categories \(p. 585\)](#)
- [Deleting an Amazon RDS event notification subscription \(p. 586\)](#)

## 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.

### Topics

- [RDS resources eligible for event subscription \(p. 572\)](#)
- [Basic process for subscribing to Amazon RDS event notifications \(p. 573\)](#)
- [Delivery of RDS event notifications \(p. 573\)](#)
- [Billing for Amazon RDS event notifications \(p. 573\)](#)
- [Examples of Aurora events \(p. 573\)](#)

## 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
- Custom engine version

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 that receives all event notifications for all DB instances and another subscription that includes only

critical events for a subset of the DB instances. For the second subscription, specify one or more DB instances in the filter.

## 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.
3. You confirm your subscription by choosing the link in the notification you received.
4. The Amazon RDS console updates the **My Event Subscriptions** section with the status of your subscription.
5. Amazon RDS begins sending the notifications to the addresses that you provided when you created the subscription.

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](#).

## Examples of Aurora events

The following examples illustrate different types of Aurora events in JSON format. For a tutorial that shows you how to capture and view events in JSON format, see [Tutorial: Log DB instance state changes using Amazon EventBridge \(p. 587\)](#).

## Topics

- [Example of a DB cluster event \(p. 574\)](#)
- [Example of a DB instance event \(p. 574\)](#)
- [Example of a DB parameter group event \(p. 575\)](#)
- [Example of a DB cluster snapshot event \(p. 575\)](#)

### 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-b930-0153b71dc42",
    "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-b930-0153b71dc42",
    "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"
    }
}

```

}

## Granting permissions to publish notifications to an Amazon SNS topic

To grant Amazon RDS permissions to publish notifications to an Amazon Simple Notification Service (Amazon SNS) topic, attach an AWS Identity and Access Management (IAM) policy to the destination topic. For more information about permissions, see [Example cases for Amazon Simple Notification Service access control](#) in the *Amazon Simple Notification Service Developer Guide*.

By default, an Amazon SNS topic has a policy allowing all Amazon RDS resources within the same account to publish notifications to it. You can attach a custom policy to allow cross-account notifications, or to restrict access to certain resources.

The following is an example of an IAM policy that you attach to the destination Amazon SNS topic. It restricts the topic to DB instances with names that match the specified prefix. To use this policy, specify the following values:

- **Resource** – The Amazon Resource Name (ARN) for your Amazon SNS topic
- **SourceARN** – Your RDS resource ARN
- **SourceAccount** – Your AWS account ID

To see a list of resource types and their ARNs, see [Resources Defined by Amazon RDS](#) in the *Service Authorization Reference*.

```
{  
    "Version": "2008-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Principal": {  
                "Service": "events.rds.amazonaws.com"  
            },  
            "Action": [  
                "sns:Publish"  
            ],  
            "Resource": "arn:aws:sns:us-east-1:123456789012:topic_name",  
            "Condition": {  
                "ArnLike": {  
                    "aws:SourceArn": "arn:aws:rds:us-east-1:123456789012:db:prefix-*"  
                },  
                "StringEquals": {  
                    "aws:SourceAccount": "123456789012"  
                }  
            }  
        }  
    ]  
}
```

## 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:

### Source type

The type of source. For example, **Source type** might be **Instances**. You must choose a source type.

### Resources to include

The Amazon RDS resources that are generating the events. For example, you might choose **Select specific instances** and then **myDBInstance1**.

The following table explains the result when you specify or don't specify **Resources to include**.

Resources to include	Description	Example
Specified	RDS notifies you about all events for the specified resource only.	If your <b>Source type</b> is <b>Instances</b> and your resource is <b>myDBInstance1</b> , RDS notifies you about all events for myDBInstance1 only.
Not specified	RDS notifies you about the events for the specified source type for all your Amazon RDS resources.	If your <b>Source type</b> is <b>Instances</b> , RDS notifies you about all instance-related events in your 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. Enter your subscription details as follows:
  - a. For **Name**, enter a name for the event notification subscription.
  - b. For **Send notifications to**, do one of the following:
    - Choose **New email topic**. Enter a name for your email topic and a list of recipients.
    - Choose **Amazon Resource Name (ARN)**. Then choose existing Amazon SNS ARN for an Amazon SNS topic.

If you want to use a topic that has been enabled for server-side encryption (SSE), grant Amazon RDS the necessary permissions to access the AWS KMS key. For more information, see

[Enable compatibility between event sources from AWS services and encrypted topics in the Amazon Simple Notification Service Developer Guide.](#)

- c. For **Source type**, choose a source type. For example, choose **Clusters** or **Cluster snapshots**.
- d. Choose the event categories and resources that you want to receive event notifications for.

The following example configures event notifications for the DB instance named testinst.

**Source**

**Source type**  
Source type of resource this subscription will consume events from

Instances

**Instances to include**  
Instances that this subscription will consume events from

All instances  
 Select specific instances

**Specific instances**

Select instances ▾

testinst X

**Event categories to include**  
Event categories that this subscription will consume events from

All event categories  
 Select specific event categories

- e. Choose **Create**.

The Amazon RDS console indicates that the subscription is being created.

**Event subscriptions (2)**

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 myeventsSubscription \
--sns-topic-arn arn:aws:sns:us-east-1:123456789012:myawsuser-RDS \
--enabled
```

For Windows:

```
aws rds create-event-subscription ^
--subscription-name myeventsubscription ^
--sns-topic-arn arn:aws:sns:us-east-1:123456789012: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"/> <a href="#">Edit</a> <a href="#">Delete</a> <a href="#" style="background-color: orange; color: white;">Create event subscription</a>		
<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.

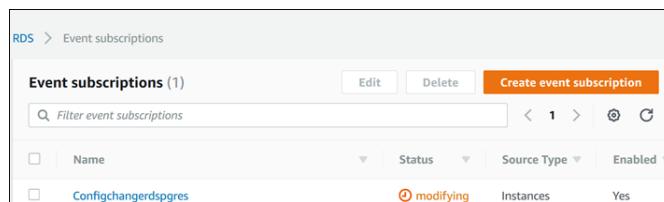
## Modifying an Amazon RDS event notification subscription

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 \(p. 582\)](#).

### 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 \(p. 582\)](#).

### 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:

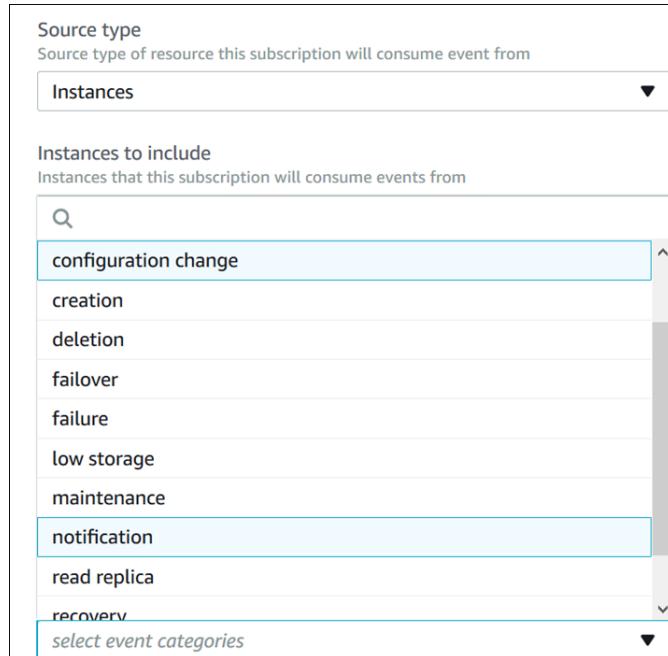
- `SubscriptionName`
- `SourceIdentifier`

## List 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 \(p. 582\)](#).



### 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 DB instance state changes using Amazon EventBridge \(p. 587\)](#)

## Tutorial: Log DB instance state changes using Amazon EventBridge

In this tutorial, you create an AWS Lambda function that logs the state changes for an instance. You then create a rule that runs the function whenever there is a 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 DB instance.

## Topics

- [Step 1: Create an AWS Lambda function \(p. 587\)](#)
- [Step 2: Create a rule \(p. 588\)](#)
- [Step 3: Test the rule \(p. 588\)](#)

## 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 16.x**.
  - c. For **Architecture**, choose **x86\_64**.
  - d. For **Execution role**, do either of the following:
    - Choose **Create a new role with basic Lambda permissions**.
    - For **Existing role**, choose **Use an existing role**. Choose the role that you want to use.
5. On the **RDSInstanceStateChange** page, do the following:
  - a. In **Code source**, select **index.js**.
  - b. In the **index.js** pane, delete the existing code.
  - c. Enter the following code:

```
console.log('Loading function');

exports.handler = async (event, context) => {
    console.log('Received event:', JSON.stringify(event));
};
```

- d. 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. Choose **Rule with an event pattern**, and then choose **Next**.
6. For **Event source**, choose **AWS events or EventBridge partner events**.
7. Scroll down to the **Event pattern** section.
8. For **Event source**, choose **AWS services**.
9. For **AWS service**, choose **Relational Database Service (RDS)**.
10. For **Event type**, choose **RDS DB Instance Event**.
11. Leave the default event pattern. Then choose **Next**.
12. For **Target types**, choose **AWS service**.
13. For **Select a target**, choose **Lambda function**.
14. For **Function**, choose the Lambda function that you created. Then choose **Next**.
15. In **Configure tags**, choose **Next**.
16. Review the steps in your rule. Then choose **Create rule**.

## 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"  
    }  
}
```

For more examples of RDS events in JSON format, see [Overview of events for Aurora \(p. 568\)](#).

10. (Optional) When you're finished, you can open the Amazon RDS console and start the instance that you stopped.

## 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 cluster events \(p. 590\)](#)
- [DB instance events \(p. 592\)](#)
- [DB parameter group events \(p. 595\)](#)
- [DB security group events \(p. 595\)](#)
- [DB cluster snapshot events \(p. 595\)](#)
- [RDS Proxy events \(p. 596\)](#)

## 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 <a href="#">Monitoring Amazon Aurora with Database Activity Streams (p. 619)</a> .
configuration change	RDS-EVENT-0180	Database Activity Streams is stopped on your database cluster. For more information see <a href="#">Monitoring Amazon Aurora with Database Activity Streams (p. 619)</a> .
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.
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 <a href="#">Migrating data from MySQL by using an Amazon S3 bucket (p. 693)</a> .

Category	RDS event ID	Description
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 <a href="#">Maintaining an Amazon Aurora DB cluster (p. 321)</a> .
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.
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.

Category	RDS event ID	Description
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 instance events

The following table shows the event category and a list of events when a DB instance is the source type.

Category	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 <a href="#">Backtracking an Aurora DB cluster (p. 725)</a> .
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.

Category	RDS event ID	Description
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.
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 <a href="#">To create an IAM role for Amazon RDS enhanced monitoring (p. 520)</a> .
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 <a href="#">To create an IAM role for Amazon RDS enhanced monitoring (p. 520)</a> .
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 <a href="#">Migrating data from MySQL by using an Amazon S3 bucket (p. 693)</a> .
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 <a href="#">RDS FAQ</a> . You can monitor the storage space for a DB instance using the <b>Free Storage Space</b> metric.

Category	RDS event ID	Description
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 <b>Free Storage Space</b> metric.
low storage	RDS-EVENT-0227	The Aurora storage subsystem is running low on space.
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	The DB instance was patched.
maintenance	RDS-EVENT-0155	The DB instance has a DB engine minor version upgrade required.
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.
notification	RDS-EVENT-0087	The DB instance has been stopped.
notification	RDS-EVENT-0088	The DB instance has been started.
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 <a href="#">Troubleshooting a MySQL read replica problem</a> .
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.

Category	RDS event ID	Description
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-0011	Updated to use DBParameterGroup <i>name</i> .
configuration change	RDS-EVENT-0092	Finished updating DB parameter group.

## 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 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.

Category	RDS event ID	Description
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).
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-0215	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).

# Monitoring Amazon Aurora log files

Every RDS database engine generates logs that you can access for auditing and troubleshooting. The type of logs depends on your database engine.

You can access database logs using the AWS Management Console, the AWS Command Line Interface (AWS CLI), or the Amazon RDS API. You can't view, watch, or download transaction logs.

## 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 \(p. 597\)](#)
- [Downloading a database log file \(p. 598\)](#)
- [Watching a database log file \(p. 599\)](#)
- [Publishing database logs to Amazon CloudWatch Logs \(p. 601\)](#)
- [Reading log file contents using REST \(p. 602\)](#)
- [Aurora MySQL database log files \(p. 604\)](#)
- [PostgreSQL database log files \(p. 610\)](#)

## Viewing and listing database log files

You can view database log files for your Amazon Aurora 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.

## Note

You can't view the log files for Aurora Serverless v1 DB clusters in the RDS console. However, you can view them in the Amazon CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.

## 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. (Optional) Enter a search term to filter your results.

The following example lists logs filtered by the text **error**.

Logs (3)		<input type="button" value="C"/>	<input type="button" value="View"/>	<input type="button" value="Watch"/>	<input type="button" value="Download"/>
<input type="text"/> error		<input type="button" value="X"/>	< 1 >	<input type="button" value="G"/>	
Name	Last written	Logs			
<input type="radio"/> error/mysql-error-running.log	Fri Dec 10 2021 12:30:00 GMT-0500	72.3 kB			
<input type="radio"/> error/mysql-error.log	Fri Dec 10 2021 12:36:40 GMT-0500	14.9 kB			

7. 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**.

## Download Log: error/postgresql.log.2018-02-10-20 (4 kB)

Please right-click the download link below and choose "Save Link As..."

[Log Name: error/postgresql.log.2018-02-10-20](#)

[Close](#)

## 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

Watching a database log file is equivalent to tailing the file on a UNIX or Linux system. You can watch a log file by using the AWS Management Console. RDS refreshes the tail of the log every 5 seconds.

### 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.

The screenshot shows the AWS RDS 'Databases' section for a database named 'database-1'. At the top right are 'Modify' and 'Actions' buttons. Below is a summary table with the following data:

DB identifier <b>database-1</b>	CPU 2.53%	Status <span style="color: green;">Available</span>	Class <b>db.m5.large</b>
Role Instance	Current activity 0.00 sessions	Engine MariaDB	Region & AZ us-east-1d

Below the summary are tabs: Connectivity & security, Monitoring, **Logs & events**, Configuration, Maintenance & backups, and Tags. The 'Logs & events' tab is circled in red.

5. In the **Logs** section, choose a log file, and then choose **Watch**.

The screenshot shows the 'Logs' section with the heading 'Logs (4)'. It lists four log files:

Name	Last written	Logs
error/mysql-error-running.log	Tue Aug 02 2022 10:00:00 GMT-0400	0 bytes
<b>error/mysql-error-running.log.2022-08-02.14</b>	Tue Aug 02 2022 09:18:13 GMT-0400	2.9 kB
error/mysql-error.log	Tue Aug 02 2022 11:30:00 GMT-0400	0 bytes
mysqlUpgrade	Tue Aug 02 2022 09:18:16 GMT-0400	1 kB

RDS shows the tail of the log, as in the following MySQL example.

#### Watching Log: error/mysql-error-running.log.2022-08-02.14 (2.9 kB)

```
text: █ background: █
2022-08-02T13:18:12.483484Z 0 [Warning] [MY-011068] [Server] The syntax 'skip_slave_start' is deprecated and will be removed in a future release. Please use skip_replica_start instead.
2022-08-02T13:18:12.483491Z 0 [Warning] [MY-011068] [Server] The syntax 'slave_exec_mode' is deprecated and will be removed in a future release. Please use replica_exec_mode instead.
2022-08-02T13:18:12.483498Z 0 [Warning] [MY-011068] [Server] The syntax 'slave_load_tmpdir' is deprecated and will be removed in a future release. Please use replica_load_tmpdir instead.
2022-08-02T13:18:12.485031Z 0 [Warning] [MY-010101] [Server] Insecure configuration for --secure-file-priv: Location is accessible to all OS users. Consider choosing a different directory.
2022-08-02T13:18:12.485063Z 0 [Warning] [MY-010918] [Server] 'default_authentication_plugin' is deprecated and will be removed in a future release. Please use authentication_policy instead.
2022-08-02T13:18:12.485811Z 0 [System] [MY-010116] [Server] /rdsdbbin/mysql/bin/mysqld (mysqld 8.0.28) starting as process 722
2022-08-02T13:18:12.559455Z 0 [Warning] [MY-010075] [Server] No existing UUID has been found, so we assume that this is the first time that this server has been started. Generating a new UUID: 8f6bd551-1265-11ed-840d-0251cdc2d067.
2022-08-02T13:18:12.580292Z 1 [System] [MY-013576] [InnoDB] InnoDB initialization has started.
2022-08-02T13:18:12.592437Z 1 [Warning] [MY-012191] [InnoDB] Scan path '/rdsdbdata/db/innodb' is ignored because it is a sub-directory of '/rdsdbdata/db/'
2022-08-02T13:18:12.856761Z 1 [System] [MY-013577] [InnoDB] InnoDB initialization has ended.
2022-08-02T13:18:13.126041Z 0 [Warning] [MY-013414] [Server] Server SSL certificate doesn't verify: unable to get issuer certificate
2022-08-02T13:18:13.126139Z 0 [System] [MY-013602] [Server] Channel mysql_main configured to support TLS. Encrypted connections are now supported for this channel.
2022-08-02T13:18:13.158424Z 0 [System] [MY-010931] [Server] /rdsdbbin/mysql/bin/mysqld: ready for connections. Version: '8.0.28' socket: '/tmp/mysql.sock' port: 3306 Source distribution.
----- END OF LOG -----
```

Watching error/mysql-error-running.log.2022-08-02.14, updates every 5 seconds.

# Publishing database logs to Amazon CloudWatch Logs

In an on-premises database, the database logs reside on the file system. Amazon RDS doesn't provide host access to the database logs on the file system of your DB cluster. For this reason, Amazon RDS lets you export database 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.

## Topics

- [Overview of RDS integration with CloudWatch Logs \(p. 601\)](#)
- [Deciding which logs to publish to CloudWatch Logs \(p. 601\)](#)
- [Specifying the logs to publish to CloudWatch Logs \(p. 601\)](#)
- [Searching and filtering your logs in CloudWatch Logs \(p. 602\)](#)

## Overview of RDS integration with CloudWatch Logs

In CloudWatch Logs, a *log stream* is a sequence of log events that share the same source. Each separate source of logs in CloudWatch Logs makes up a separate log stream. A *log group* is a group of log streams that share the same retention, monitoring, and access control settings.

Amazon Aurora continuously streams your DB cluster 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.

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](#).

## Deciding which logs to publish to CloudWatch Logs

Each RDS database engine supports its own set of logs. To learn about the options for your database engine, review the following topics:

- [the section called "Publishing Aurora MySQL logs to CloudWatch Logs" \(p. 924\)](#)
- [the section called "Publishing Aurora PostgreSQL logs to CloudWatch Logs" \(p. 1265\)](#)

## Specifying the logs to publish to CloudWatch Logs

You specify which logs to publish in the console. 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 \(p. 1724\)](#).

### To specify the logs to publish

1. Open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. In the navigation pane, choose **Databases**.
3. Do either of the following:
  - Choose **Create database**.
  - Choose a database from the list, and then choose **Modify**.
4. In **Logs exports**, choose which logs to publish.

The following example specifies the audit log, error logs, general log, and slow query log.

**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.

## Searching and filtering your logs in CloudWatch Logs

You can search for log entries that meet a specified criteria using the CloudWatch Logs console. You can access the logs either through the RDS console, which leads you to the CloudWatch Logs console, or from the CloudWatch Logs console directly.

### To search your RDS logs using the RDS console

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 or a DB instance.
4. Choose **Configuration**.
5. Under **Published logs**, choose the database log that you want to view.

### To search your RDS logs using the CloudWatch Logs console

1. Open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. In the navigation pane, choose **Log groups**.
3. In the filter box, enter `/aws/rds`.
4. For **Log Groups**, choose the name of the log group containing the log stream to search.
5. For **Log Streams**, choose the name of the log stream to search.
6. Under **Log events**, enter the filter syntax to use.

For more information, see [Searching and filtering log data](#) in the *Amazon CloudWatch Logs User Guide*. For a blog tutorial explaining how to monitor RDS logs, see [Build proactive database monitoring for Amazon RDS with Amazon CloudWatch Logs, AWS Lambda, and Amazon SNS](#).

## 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*—*DBInstanceIdentifier* 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 \(p. 597\)](#).

### Topics

- [Overview of Aurora MySQL database logs \(p. 604\)](#)
- [Publishing Aurora MySQL logs to Amazon CloudWatch Logs \(p. 606\)](#)
- [Managing table-based Aurora MySQL logs \(p. 606\)](#)
- [Configuring Aurora MySQL binary logging \(p. 607\)](#)
- [Accessing MySQL binary logs \(p. 608\)](#)

## Overview of Aurora MySQL database logs

You can monitor the following types of Aurora MySQL log files:

- Error log
- Slow query log
- General log
- 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 \(p. 604\)](#)
- [Aurora MySQL slow query and general logs \(p. 605\)](#)
- [Aurora MySQL audit log \(p. 605\)](#)
- [Log rotation and retention for Aurora MySQL \(p. 605\)](#)
- [Size limits on BLOBs \(p. 606\)](#)

## 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 didn't encounter an error that would result in a log entry.

By design, the error logs are filtered so that only unexpected events such as errors are shown. However, the error logs also contain some additional database information, for example query progress, that isn't shown. Therefore, even without any actual errors the size of the error logs might increase because of ongoing database activities.

Aurora MySQL writes `mysql-error.log` to disk every 5 minutes. It appends the contents of the log to `mysql-error-running.log`.

Aurora MySQL rotates the `mysql-error-running.log` file every hour. It 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 parameter groups \(p. 215\)](#). 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. Queries that don't use an index are logged even if their run time is less than the value of the `long_query_time` parameter. The default is 0.
- `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.
  - **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**.

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](#)

## Aurora MySQL audit log

Audit logging for Aurora MySQL is called Advanced Auditing. To turn on Advanced Auditing, you set certain DB cluster parameters. For more information, see [Using Advanced Auditing with an Amazon Aurora MySQL DB cluster \(p. 823\)](#).

## Log rotation and retention for Aurora MySQL

When logging is enabled, Amazon Aurora rotates 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. Aurora MySQL handles rotation and deletion as follows:

- The Aurora MySQL error log file sizes are constrained to no more than 15 percent of the local storage for a DB instance. To maintain this threshold, logs are automatically rotated every hour. Aurora MySQL removes logs after 30 days 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.
- When `FILE` logging is enabled, general log and slow query log files are examined every hour and log files more than 24 hours old are deleted. In some cases, the remaining combined log file size after the deletion might exceed the threshold of 15 percent of a DB instance's local 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, log tables aren't rotated or deleted. Log tables are truncated when the size of all logs combined is too large. You can subscribe to the `low_free_storage` event to be notified when log tables should be manually rotated or deleted to free up space. For more information, see [Working with Amazon RDS event notification \(p. 572\)](#).

You can rotate the `mysql.general_log` table manually 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.

When you rotate log tables manually, 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`.

- The Aurora MySQL audit logs are rotated when the file size reaches 100 MB, and removed after 24 hours.

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 Aurora MySQL error log, these log files are rotated hourly. The log files that were generated during the previous 24 hours are retained. Note that the retention period is different between Amazon RDS and Aurora.

## 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 parameter groups \(p. 215\)](#). 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 \(p. 924\)](#).

## 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.

Binary logs are accessible only from the primary DB instance, not from the replicas.

MySQL on Amazon Aurora supports the *row-based*, *statement-based*, and *mixed* binary logging formats. We recommend mixed unless you need a specific binlog format. 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 parameter groups \(p. 215\)](#).

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 \(p. 949\)](#).

## Accessing MySQL binary logs

You can use the `mysqlbinlog` utility to download or stream binary logs from RDS for MySQL DB instances. 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.

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 \(p. 597\)](#).

### Topics

- [Overview of PostgreSQL logs \(p. 610\)](#)
- [Enabling query logging \(p. 613\)](#)

## 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 look for query failures, login failures, deadlocks, and fatal server errors in the log. 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 based on the following categories:

- Connections and disconnections
- Checkpoints
- Schema modification queries
- Queries waiting for locks
- Queries consuming temporary disk storage
- Backend autovacuum process consuming resources

By logging information for various categories such as shown in the list, you can 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](#).

### Parameters that affect logging behavior

Each Aurora PostgreSQL instance has a *parameter group* that specifies its configuration, including various aspects of logging. The default parameter group settings apply to every Aurora PostgreSQL DB cluster in a given AWS Region. You can't change the defaults because they apply to all instances of a given engine, even those that aren't yours. To modify any parameter values, you create a custom parameter group and modify its settings. For example, to set or change logging parameters, you make changes in the custom

parameter group associated with your Aurora PostgreSQL DB cluster. To learn how, see [Working with parameter groups \(p. 215\)](#).

For an Aurora PostgreSQL DB cluster, the parameters that affect logging behavior include the following:

- `rds.log_retention_period` – PostgreSQL logs that are older than the specified number of minutes are deleted. The default value of 4320 minutes deletes log files after 3 days. For more information, see [Setting the log retention period \(p. 611\)](#).
- `log_rotation_age` – Specifies number of minutes after which Amazon RDS automatically rotates the logs. The default is 60 minutes, but you can specify anywhere from 1 to 1440 minutes. For more information, see [Setting log file rotation \(p. 611\)](#).
- `log_rotation_size` – Sets the size, in kilobytes, at which the Amazon RDS should automatically rotate the logs. There is no value by default because the logs are rotated based on age alone, as specified by the `log_rotation_age` parameter. For more information, see [Setting log file rotation \(p. 611\)](#).
- `log_line_prefix` – Specifies the information that gets prefixed in front of each line that gets logged. The default string for this parameter is `%t :%r :%u@%d :[%p] :`, which notes the time (`%t`) and other distinguishing characteristics such as the database name (`%d`) for the log entry. You can't change this parameter. It applies to the `stderr` messages that get logged.
- `log_destination` – Sets the output format for server logs. The default value for this parameter is standard error (`stderr`), but `csvlog` (comma-separated value log files) is also supported. For more information, see [Setting the log destination \(p. 612\)](#).

## 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 Aurora PostgreSQL DB cluster. 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 needs 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 more information, see the [gzip](#) website.

When storage for the DB instance is low and all available logs are compressed, you get a warning such as the following:

Warning: local storage for PostgreSQL log files is critically low for this Aurora PostgreSQL instance, and could lead to a database outage.

If there's not enough storage, Aurora might delete compressed PostgreSQL logs before the end of specified retention period. If that happens, you see a message similar to the following:

The oldest PostgreSQL log files were deleted due to local storage constraints.

We recommend that you have your logs routinely published to Amazon CloudWatch Logs, so you can view and analyze system data long after the logs have been removed from your Aurora PostgreSQL DB cluster. For more information, see [Publishing Aurora PostgreSQL logs to Amazon CloudWatch Logs \(p. 1265\)](#). After you set up CloudWatch publishing, Aurora doesn't delete a log until after it's published to CloudWatch Logs.

## Setting log file rotation

New log files are created by Aurora every hour by default. The timing is controlled by the `log_rotation_age` parameter. This parameter has a default value of 60 (minutes), but you can set to

anywhere from 1 minute to 24 hours (1,440 minutes). When it's time for rotation, a new distinct log file is created. The file is named according to the pattern specified by the `log_filename` parameter.

Log files can also be rotated according to their size, as specified in the `log_rotation_size` parameter. This parameter specifies that the log should be rotated when it reaches the size (in kilobytes). The default `log_rotation_size` is 100000 kB (kilobytes) for an Aurora PostgreSQL DB cluster, but you can set this to anywhere from 50,000 to 1,000,000 kilobytes.

The log file names are based on the file name pattern specified in the `log_filename` parameter. The available settings for this parameter are as follows:

- `postgresql.log.%Y-%m-%d` – Default format for the log file name. Includes the year, month, and date.
- `postgresql.log.%Y-%m-%d-%H` – Hour format for log file name. Sets the granularity of log to hours.
- `postgresql.log.%Y-%m-%d-%H%M` – Minute format for log file name. Sets the granularity of the log to less than an hour.

If you set `log_rotation_age` parameter to less than 60 minutes, be sure to also set the `log_filename` parameter to the minute format.

For more information, see [log\\_rotation\\_age](#) and [log\\_rotation\\_size](#) in the PostgreSQL documentation.

## Setting the log destination

By default, Aurora PostgreSQL generates logs in standard error (`stderr`) format. This is the default setting for the `log_destination` parameter. This format prefixes each log message with the time, database, and other details specified by the `log_line_prefix` parameter. The `log_line_prefix` is set to the following text string, which can't be changed:

```
%t:%r:%u@%d:[%p]:t
```

This parameter specifies the following details for each log entry:

- `%t` – Time of log entry.
- `%r` – Remote host address.
- `%u@%d` – User name @ database name.
- `[%p]` – Process ID if available.

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
```

Aurora PostgreSQL can generate the logs in `csvlog` format in addition to the default `stderr` specified by the `log_destination` parameter. The `csvlog` is useful for analyzing the log data as CSV data. For example, say that you use the `log_fdw` extension to work with your logs as foreign tables. The foreign table created on `stderr` log files contains a single column with log event data. For the CSV formatted log file, the foreign table has multiple columns, so you can sort and analyze your logs much more easily.

You must be using a custom parameter group so that you change the `log_destination` setting. The `log_destination` parameter is dynamic, that is, the change takes effect immediately, without rebooting.

If you do change this parameter, you need to be aware that `csvlog` files are generated in addition to the `stderr` logs. We recommend that you pay attention to the storage consumed by the logs, taking into account the `rds.log_retention_period` and other settings that affect log storage and turnover. Using both `stderr` and `csvlog` more than doubles the storage consumed by the logs.

If you do set the `log_destination` to include `csvlog` and you later decide that you want to revert to the default only (`stderr`), you can open the custom parameter group for your instance using the AWS Management Console, choose the `log_destination` parameter from the list, choose **Edit parameter** and then choose **Reset**. This reverts the `log_destination` parameter to its default setting, `stderr`.

For more information about configuring logging, see [Working with Amazon 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 `ddl`. 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 `ddl`, `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 Auditing (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:

- Set the `log_statement` parameter to `all`. The following example shows the information that is written to the `postgresql.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 `postgresql.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:
```

2. Set the `log_min_duration_statement` parameter. The following example shows the information that is written to the `postgresql.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 `postgresql.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.reltargetspace
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 \(p. 615\)](#)
- [Amazon Aurora log file entries \(p. 615\)](#)

## 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": "8.0.28",  
        "dBInstancePort": 0,  
        "optionGroupMemberships": [  
            {  
                "status": "in-sync",  
                "optionGroupName": "default:mysql-8-0"  
            }  
        ],  
        "dBParameterGroups": [  
            {  
                "dBParameterGroupName": "default.mysql8.0",  
                "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": "available"  
                }  
            ]  
        }  
    }  
}
```

```

        "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 \(p. 619\)](#)
- [Network prerequisites for Aurora MySQL database activity streams \(p. 622\)](#)
- [Starting a database activity stream \(p. 623\)](#)
- [Getting the status of a database activity stream \(p. 624\)](#)
- [Stopping a database activity stream \(p. 625\)](#)
- [Monitoring database activity streams \(p. 626\)](#)
- [Managing access to database activity streams \(p. 648\)](#)

## 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 \(p. 619\)](#)
- [Asynchronous and synchronous mode for database activity streams \(p. 620\)](#)
- [Requirements for database activity streams \(p. 621\)](#)
- [Supported Aurora engine versions for database activity streams \(p. 621\)](#)
- [Supported DB instance classes for database activity streams \(p. 621\)](#)

## How database activity streams work

In Amazon Aurora, you start a database activity stream at the cluster level. All DB instances within your cluster have database activity streams enabled.

Your Aurora DB cluster 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.

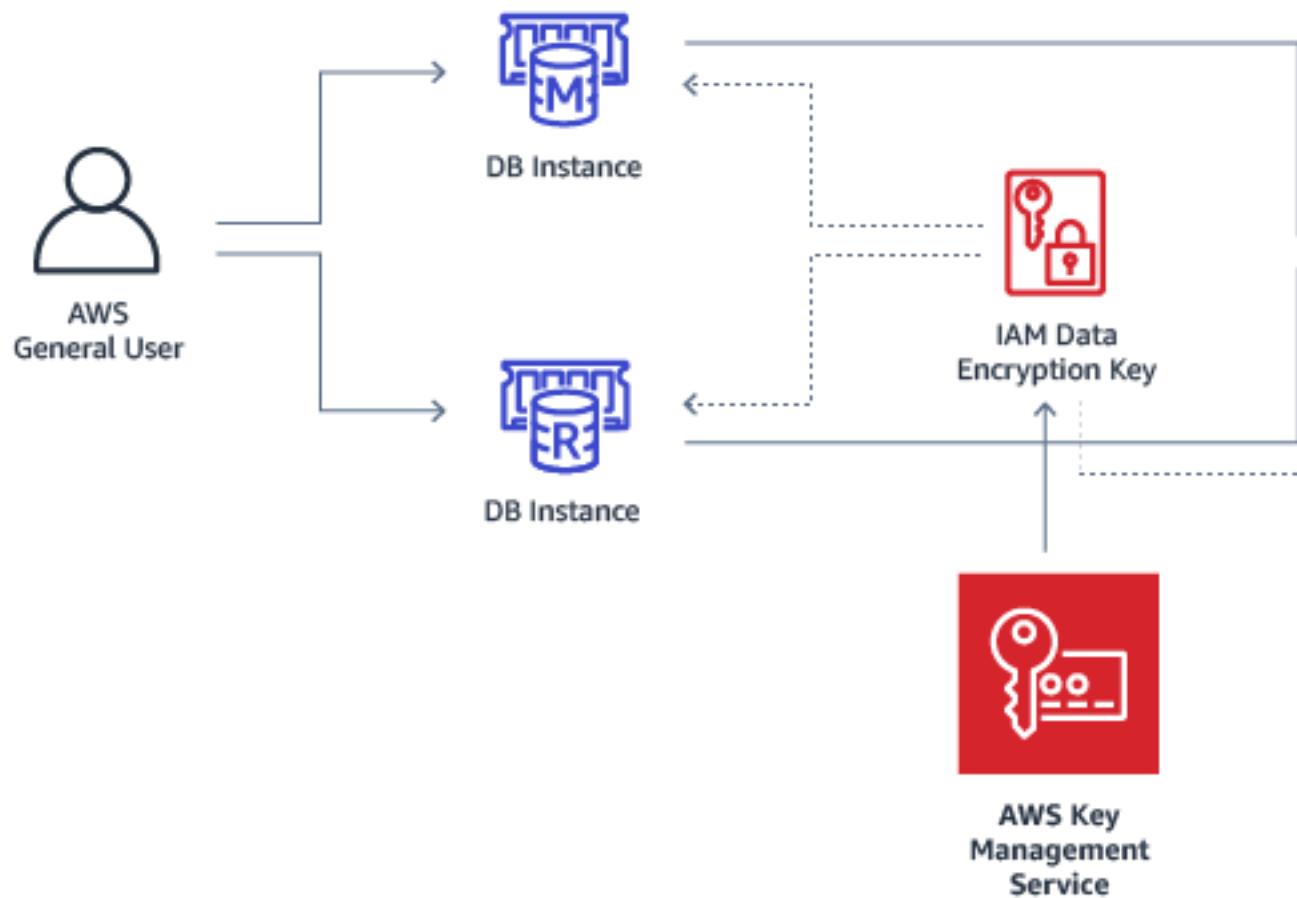
### Important

Use of the Database Activity Streams feature in Amazon Aurora is free, but Amazon Kinesis charges for a data stream. For more information, see [Amazon Kinesis Data Streams pricing](#).

If you use an Aurora global database, start a database activity stream on each DB cluster separately. Each cluster delivers audit data to its own Kinesis stream within its own AWS Region. The activity streams don't operate differently during a failover. They continue to audit your global database as usual.

You can configure applications for compliance management to 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:

- Amazon Kinesis is required for database activity streams.
- AWS Key Management Service (AWS KMS) is required for database activity streams because they are always encrypted.
- Applying additional encryption to your Amazon Kinesis data stream is incompatible with database activity streams, which are already encrypted with your AWS KMS key.
- Start your database activity stream at the DB cluster level. If you add a DB instance to your cluster, you don't need to start an activity stream on the instance: it is audited automatically.
- In an Aurora global database, make sure to start an activity stream on each DB cluster separately. Each cluster delivers audit data to its own Kinesis stream within its own AWS Region.

## Supported Aurora engine versions for database activity streams

**For Aurora PostgreSQL, database activity streams are supported for the following versions:**

- All 14, 13, and 12 versions
- Version 11.6 and higher 11 versions
- Version 10.11 and higher 10 versions

**For Aurora MySQL, database activity streams are supported for the following versions:**

2.08 or higher, which is compatible with MySQL version 5.7

**Database activity streams aren't supported for the following features:**

- Aurora Serverless v1
- Aurora Serverless v2
- Babelfish for Aurora PostgreSQL

For more information about Aurora PostgreSQL versions, see [Amazon Aurora PostgreSQL releases and engine versions..](#)

## 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

## 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 \(p. 622\)](#)
- [Prerequisites for public availability \(p. 622\)](#)
- [Prerequisites for private availability \(p. 622\)](#)

## 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 cluster in a VPC \(p. 1729\)](#). 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 in your Aurora DB cluster, start an activity stream at the cluster level. Any DB instances that you add to the cluster are also automatically monitored. If you use an Aurora global database, start a database activity stream on each DB cluster separately. Each cluster delivers audit data to its own Kinesis stream within its own AWS Region.

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 on which you want to start 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 \(p. 622\)](#) 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.

- `--resource-arn arn` – Specifies the Amazon Resource Name (ARN) of the DB cluster.
- `--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`.
- `--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.

The following example starts a database activity stream for a DB cluster in asynchronous mode.

For Linux, macOS, or Unix:

```
aws rds start-activity-stream \
--mode async \
--kms-key-id my-kms-key-arn \
--resource-arn my-cluster-arn \
--apply-immediately
```

For Windows:

```
aws rds start-activity-stream ^
--mode 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`
- `KmsKeyId`
- `ResourceArn`
- `Mode`

## 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.

The following example describes *my-cluster*.

```
aws rds --region my-region describe-db-clusters --db-cluster-identifier my-cluster
```

The following example shows a JSON response. The following fields are shown:

- `ActivityStreamKinesisStreamName`
- `ActivityStreamKmsKeyId`
- `ActivityStreamStatus`
- `ActivityStreamMode`
- 

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-  
A6TSYXITZCZXJH1RVFUBZ5LTWY",  
            "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 \(p. 627\)](#)
- [Audit log contents and examples \(p. 628\)](#)
- [Processing a database activity stream using the AWS SDK \(p. 642\)](#)

## 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 \(p. 628\)](#)
- [DatabaseActivityMonitoringRecords JSON object \(p. 634\)](#)
- [databaseActivityEvents JSON Object \(p. 634\)](#)
- [databaseActivityEventList JSON array \(p. 635\)](#)

### 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",
      "text": "CREATE TABLE test_table (id integer primary key, name
varchar(32));"
    }
  ]
}
```

```

    "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"> <li>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.</li> <li>Version 1.0 database activity records are generated for Aurora PostgreSQL DB clusters running the engine versions 10.7 and 11.4.</li> </ul> <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 <a href="#">databaseActivityEventList (p. 635)</a> <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> <sup>(String635)</sup>		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"> <li>• ALL</li> <li>• CONNECT – A connect or disconnect event.</li> <li>• DDL – A DDL statement that is not included in the list of statements for the ROLE class.</li> <li>• FUNCTION – A function call or a DO block.</li> <li>• MISC – A miscellaneous command such as DISCARD, FETCH, CHECKPOINT, or VACUUM.</li> <li>• NONE</li> <li>• READ – A SELECT or COPY statement when the source is a relation or a query.</li> <li>• ROLE – A statement related to roles and privileges including GRANT, REVOKE, and CREATE/ALTER/DROP ROLE.</li> <li>• WRITE – An INSERT, UPDATE, DELETE, TRUNCATE, or COPY statement when the destination is a relation.</li> </ul>
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><b>Important</b> 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 <a href="#">PostgreSQL Message Severity Levels</a>. 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 <a href="#">PostgreSQL Error Codes</a>.</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"> <li>• <code>COMPOSITE TYPE</code></li> <li>• <code>FOREIGN TABLE</code></li> <li>• <code>FUNCTION</code></li> <li>• <code>INDEX</code></li> <li>• <code>MATERIALIZED VIEW</code></li> <li>• <code>SEQUENCE</code></li> <li>• <code>TABLE</code></li> <li>• <code>TOAST TABLE</code></li> <li>• <code>VIEW</code></li> <li>• <code>UNKNOWN</code></li> </ul>
<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"> <li>• <code>MAIN</code> – The primary event representing a SQL statement.</li> <li>• <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.</li> </ul>

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"> <li>• <code>CONNECT</code> – When a client session is connected.</li> <li>• <code>QUERY</code> – A SQL statement. Accompanied by one or more events with a <code>class</code> value of <code>AUX</code>.</li> <li>• <code>DISCONNECT</code> – When a client session is disconnected.</li> <li>• <code>FAILED_CONNECT</code> – When a client attempts to connect but isn't able to.</li> <li>• <code>CHANGEUSER</code> – A state change that's part of the MySQL network protocol, not from a statement that you issue.</li> </ul> <p>The values when <code>class</code> is <code>AUX</code> include the following:</p> <ul style="list-style-type: none"> <li>• <code>READ</code> – A <code>SELECT</code> or <code>COPY</code> statement when the source is a relation or a query.</li> <li>• <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.</li> <li>• <code>DROP</code> – Deleting an object.</li> <li>• <code>CREATE</code> – Creating an object.</li> <li>• <code>RENAME</code> – Renaming an object.</li> <li>• <code>ALTER</code> – Changing the properties of an object.</li> </ul>

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><b>Important</b> 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&gt; 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 <a href="#">The Error Log</a> 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"> <li>• INDEX</li> <li>• TABLE</li> <li>• UNKNOWN</li> </ul>
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 for Amazon Aurora \(p. 1653\)](#). For more information about creating IAM policies, see [Creating and using an IAM policy for IAM database access \(p. 1686\)](#).

### 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 \(p. 651\)](#)
- [Security with Amazon Aurora MySQL \(p. 681\)](#)
- [Updating applications to connect to Aurora MySQL DB clusters using new SSL/TLS certificates \(p. 687\)](#)
- [Migrating data to an Amazon Aurora MySQL DB cluster \(p. 690\)](#)
- [Managing Amazon Aurora MySQL \(p. 721\)](#)
- [Tuning Aurora MySQL with wait events and thread states \(p. 746\)](#)
- [Working with parallel query for Amazon Aurora MySQL \(p. 790\)](#)
- [Using Advanced Auditing with an Amazon Aurora MySQL DB cluster \(p. 823\)](#)
- [Single-master replication with Amazon Aurora MySQL \(p. 826\)](#)
- [Working with Aurora multi-master clusters \(p. 867\)](#)
- [Integrating Amazon Aurora MySQL with other AWS services \(p. 890\)](#)
- [Amazon Aurora MySQL lab mode \(p. 939\)](#)
- [Best practices with Amazon Aurora MySQL \(p. 940\)](#)
- [Amazon Aurora MySQL reference \(p. 949\)](#)
- [Database engine updates for Amazon Aurora MySQL \(p. 990\)](#)

## Overview of Amazon Aurora MySQL

The following sections provide an overview of Amazon Aurora MySQL.

## Topics

- [Amazon Aurora MySQL performance enhancements \(p. 651\)](#)
- [Amazon Aurora MySQL and spatial data \(p. 652\)](#)
- [Aurora MySQL version 3 compatible with MySQL 8.0 \(p. 653\)](#)
- [Aurora MySQL version 2 compatible with MySQL 5.7 \(p. 680\)](#)

## 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.

Some Aurora features, such as Aurora Serverless v2, require Aurora MySQL version 3.

### Topics

- [Features from MySQL 8.0 Community Edition \(p. 653\)](#)
- [Aurora MySQL version 3 prerequisite for Aurora MySQL Serverless v2 \(p. 654\)](#)
- [Release notes for Aurora MySQL version 3 \(p. 654\)](#)
- [New parallel query optimizations \(p. 654\)](#)
- [New temporary table behavior in Aurora MySQL version 3 \(p. 655\)](#)
- [Comparison of Aurora MySQL version 2 and Aurora MySQL version 3 \(p. 658\)](#)
- [Comparison of Aurora MySQL version 3 and MySQL 8.0 Community Edition \(p. 663\)](#)
- [Upgrading to Aurora MySQL version 3 \(p. 666\)](#)

## Features from MySQL 8.0 Community Edition

The initial release of Aurora MySQL version 3 is compatible with MySQL 8.0.23 Community Edition. 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 \(Aurora MySQL version 3\) \(p. 741\)](#).
- 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 \(p. 664\)](#).
- `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 \(p. 662\)](#). 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 \(p. 982\)](#). 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 \(p. 660\)](#).

## Aurora MySQL version 3 prerequisite for Aurora MySQL Serverless v2

Aurora MySQL version 3 is a prerequisite for all DB instances in an Aurora MySQL Serverless v2 cluster. Aurora MySQL Serverless v2 includes support for reader instances in a DB cluster, and other Aurora features that aren't available for Aurora MySQL Serverless v1. It also has faster and more granular scaling than Aurora MySQL Serverless v1.

## 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](#) in the *Release Notes for Aurora MySQL*.

## 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 \(p. 802\)](#). For general information about Aurora parallel query, see [Working with parallel query for Amazon Aurora MySQL \(p. 790\)](#).

## New temporary table behavior in Aurora MySQL version 3

Aurora MySQL version 3 handles temporary tables differently from earlier Aurora MySQL versions. This new behavior is inherited from MySQL 8.0 Community Edition. There are two types of temporary tables that can be created with Aurora MySQL version 3:

- Internal (or *implicit*) temporary tables – Created by the Aurora MySQL engine to handle operations such as sorting aggregation, derived tables, or common table expressions (CTEs).
- User-created (or *explicit*) temporary tables – Created by the Aurora MySQL engine when you use the `CREATE TEMPORARY TABLE` statement.

There are additional considerations for both internal and user-created temporary tables on Aurora reader DB instances. We discuss these changes in the following sections.

### Topics

- [Storage engine for internal \(implicit\) temporary tables \(p. 655\)](#)
- [Mitigating fullness issues for internal temporary tables on Aurora Replicas \(p. 656\)](#)
- [User-created \(explicit\) temporary tables on reader DB instances \(p. 657\)](#)
- [Temporary table creation errors and mitigation \(p. 657\)](#)

### Storage engine for internal (implicit) temporary tables

When generating intermediate result sets, Aurora MySQL initially attempts to write to in-memory temporary tables. If this is unsuccessful, because of either incompatible data types or configured limits, the temporary table is converted to an on-disk temporary table rather than being held in memory. More information on this can be found in the [Internal Temporary Table Use in MySQL](#) in the MySQL documentation.

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. This setting is the default.

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 GiB on DB instances with 16 or more GiB of memory, and 16 MB on DB instances with less than 16 GiB 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 `temptable_max_mmap` parameter 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 GiB, 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.

## Mitigating fullness issues for internal temporary tables on Aurora Replicas

To avoid size limitation issues for temporary tables, set the `temptable_max_ram` and `temptable_max_mmap` parameters to a combined value that can fit the requirements of your workload.

Be careful when setting the value of the `temptable_max_ram` parameter. Setting the value too high reduces the available memory on the database instance, which can cause an out-of-memory condition. Monitor the average freeable memory on the DB instance, and then determine an appropriate value for `temptable_max_ram` so that you will still have a reasonable amount of free memory left on the instance. For more information, see [Freeable memory issues in Amazon Aurora \(p. 1764\)](#).

It is also important to monitor the size of the local storage and the temporary table space consumption. For more information on monitoring local storage on an instance, see the AWS Knowledge Center article [What is stored in Aurora MySQL-compatible local storage, and how can I troubleshoot local storage issues?](#).

### Example 1

You know that your temporary tables grow to a cumulative size of 20 GiB. You want to set in-memory temporary tables to 2 GiB and to grow to a maximum of 20 GiB on disk.

Set `temptable_max_ram` to **2,147,483,648** and `temptable_max_mmap` to **21,474,836,480**. These values are in bytes.

These parameter settings make sure that your temporary tables can grow to a cumulative total of 22 GiB.

### Example 2

Your current instance size is 16xlarge or larger. You don't know the total size of the temporary tables that you might need. You want to be able to use up to 4 GiB in memory and up to the maximum available storage size on disk.

Set `temptable_max_ram` to **4,294,967,296** and `temptable_max_mmap` to **1,099,511,627,776**. These values are in bytes.

Here you're setting `temptable_max_mmap` to 1 TiB, which is less than the maximum local storage of 1.2 TiB on a 16xlarge Aurora DB instance.

On a smaller instance size, adjust the value of `temptable_max_mmap` so that it doesn't fill up the available local storage. For example, a 2xlarge instance has only 160 GiB of local storage available. Hence, we recommend setting the value to less than 160 GiB. For more information on the available local storage for DB instance sizes, see [Temporary storage limits for Aurora MySQL \(p. 724\)](#).

## User-created (explicit) temporary tables on reader DB instances

You can create explicit temporary tables using the `TEMPORARY` keyword in your `CREATE TABLE` statement. Explicit temporary tables are supported on the writer DB instance in an Aurora DB cluster. You can also use explicit temporary tables on reader DB instances, but the tables can't enforce the use of the InnoDB storage engine.

To avoid errors while creating explicit temporary tables on Aurora reader DB instances, make sure that all `CREATE TEMPORARY TABLE` statements on reader DB instances are run in either of the following ways:

- Without specifying the `ENGINE=InnoDB` clause
- With the SQL mode `NO_ENGINE_SUBSTITUTION=OFF`

## Temporary table creation errors and mitigation

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. When `NO_ENGINE_SUBSTITUTION` is turned off (default), the appropriate engine substitution is made, and the temporary table creation succeeds.

```
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.
```

For more information about the storage aspects and performance implications of temporary tables in Aurora MySQL version 3, see the blog post [Use the TempTable storage engine on Amazon RDS for MySQL and Amazon Aurora MySQL](#).

## 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 \(p. 658\)](#)
- [Instance class support \(p. 659\)](#)
- [Parameter changes for Aurora MySQL version 3 \(p. 659\)](#)
- [Status variables \(p. 660\)](#)
- [Inclusive language changes for Aurora MySQL version 3 \(p. 660\)](#)
- [AUTO\\_INCREMENT values \(p. 662\)](#)
- [Binary log replication \(p. 662\)](#)

### 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 \(p. 666\)](#).

- You can't use Aurora MySQL version 3 for Aurora Serverless v1 clusters. Aurora MySQL version 3 works with Aurora Serverless v2.
- 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 \(Aurora MySQL version 3\) \(p. 741\)](#).
- 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. 800) and [Aurora MySQL hints \(p. 982\)](#). 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 \(p. 56\)](#).

## 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 \(p. 950\)](#). 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 \(p. 956\)](#). 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 \(p. 660\)](#).

## 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 \(p. 970\)](#).

## 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 REPLICAS 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>
<code>ForwardingMasterDMLThroughput</code>	<code>ForwardingWriterDMLThroughput</code>

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
<code>Aurora_fwd_master_dml_stmt</code>	<code>Aurora_fwd_writer_dml_stmt_duration</code>
<code>Aurora_fwd_master_dml_stmt</code>	<code>Aurora_fwd_writer_dml_stmt_count</code>
<code>Aurora_fwd_master_select_stmt</code>	<code>Aurora_fwd_writer_select_stmt_duration</code>
<code>Aurora_fwd_master_select_stmt</code>	<code>Aurora_fwd_writer_select_stmt_count</code>
<code>Aurora_fwd_master_errors_session_timeout</code>	<code>Aurora_writer_errors_session_timeout</code>

Name to be removed	New or preferred name
Aurora_fwd_master_open_sessions	aurora_fwd_writer_open_sessions
Aurora_fwd_master_errors_sessions	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
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

Name to be removed	New or preferred name
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.

## 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 \(binary log replication\) \(p. 841\)](#).

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 \(Aurora MySQL version 3 and higher\) \(p. 857\)](#).

#### 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 MySQL 8.0 Community Edition

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](#) in the *Release Notes for Aurora MySQL*.

#### Topics

- [MySQL 8.0 features not available in Aurora MySQL version 3 \(p. 663\)](#)
- [Role-based privilege model \(p. 664\)](#)
- [Authentication \(p. 666\)](#)

## 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 \(p. 660\)](#).
- 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 also can't create other kinds of objects such as stored procedures in the `mysql` database. 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`. If your application creates stored procedures or other objects in the `mysql` database, use a different database instead.

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 \(p. 890\)](#). Aurora MySQL version 3 includes the following roles related to accessing other AWS services:

- AWS\_LAMBDA\_ACCESS role, as an alternative to the `INVOKER LAMBDA` privilege. For usage information, see [Invoking a Lambda function from an Amazon Aurora MySQL DB cluster \(p. 917\)](#).
- 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 \(p. 903\)](#).
- 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 \(p. 911\)](#).
- AWS\_SAGEMAKER\_ACCESS role, as an alternative to the `INVOKER SAGEMAKER` privilege. For usage information, see [Using machine learning \(ML\) with Aurora MySQL \(p. 927\)](#).
- AWS\_COMPREHEND\_ACCESS role, as an alternative to the `INVOKER COMPREHEND` privilege. For usage information, see [Using machine learning \(ML\) with Aurora MySQL \(p. 927\)](#).

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. 375).** 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 \(p. 996\)](#). 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 \(p. 705\)](#).

### 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 \(p. 667\)](#)
- [Example of upgrading from Aurora MySQL version 2 to version 3 \(p. 668\)](#)
- [Example of upgrading from Aurora MySQL version 1 to version 3 \(p. 670\)](#)
- [Troubleshooting upgrade issues with Aurora MySQL version 3 \(p. 672\)](#)
- [Post-upgrade cleanup for Aurora MySQL version 3 \(p. 680\)](#)

## 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 MySQL 8.0 Community Edition, see [Comparison of Aurora MySQL version 3 and MySQL 8.0 Community Edition \(p. 663\)](#).
- 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 \(p. 658\)](#).
- 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 \(p. 659\)](#) 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.

We recommend that you use the same setting for `lower_case_table_names` when you upgrade from Aurora MySQL version 2 to version 3.

- Review your Aurora MySQL version 2 database schema and object definitions for the usage of new reserved keywords introduced in MySQL 8.0 Community Edition, before you upgrade. For more information, see [MySQL 8.0 New Keywords and Reserved Words](#) in the MySQL documentation.

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 \(p. 666\)](#). For general information about upgrading by restoring a snapshot, see [Upgrading Amazon Aurora MySQL DB clusters \(p. 996\)](#).

**Note**

We recommend using larger DB instance classes, such as db.r5.24xlarge, when upgrading to Aurora MySQL version 3 using the snapshot restore technique. This helps the upgrade process to complete faster by using the majority of available CPU capacity on the DB instance. You can change to the DB instance class that you want after the major version upgrade is complete.

After you finish the upgrade itself, you can follow the post-upgrade procedures in [Post-upgrade cleanup for Aurora MySQL version 3 \(p. 680\)](#). 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 \(p. 690\)](#). 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.2.09.3",
```

```
"5.7.mysql_aurora.2.10.0",
"5.7.mysql_aurora.2.10.1",
"5.7.mysql_aurora.2.10.2",
"8.0.mysql_aurora.3.01.1",
"8.0.mysql_aurora.3.02.0"
]
```

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.3.02.0 \
--enable-cloudwatch-logs-exports '[{"error", "general", "slowquery", "audit"}'
{
  "DBClusterIdentifier": "cluster-80-restored",
  "Engine": "aurora-mysql",
  "EngineVersion": "8.0.mysql_aurora.3.02.0",
  "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 \(p. 659\)](#).

```
$ 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.3.02.0",
  "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.3.02.0
```

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.3.02.0` version 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.3.02.0'`) == `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 3.02.0.

```
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.3.02.0 \
--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.3.02.0",  
    "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.3.02.0 \
--enable-cloudwatch-logs-exports '["error","general","slowquery","audit"]'
{
    "DBClusterIdentifier": "cluster-80-attempt-1",
    "Engine": "aurora-mysql",
    "EngineVersion": "8.0.mysql_aurora.3.02.0",
    "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.3.02.0",
    "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": "3.02.0",
    "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. Then we run `OPTIMIZE TABLE tbl_name [, tbl_name] ...` on the tables causing the following error:

Table `tbl\_name` contains dangling FULLTEXT index. Kindly recreate the table before 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> OPTIMIZE TABLE dangling_fulltext_index;
Query OK, 0 rows affected (0.05 sec)

```

For more information, see [Optimizing InnoDB Full-Text Indexes](#) and [OPTIMIZE TABLE Statement](#) in the MySQL documentation.

As an alternative solution you could 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.3.02.0 \
--enable-cloudwatch-logs-exports '[{"error", "general", "slowquery", "audit"}]'
{
    "DBClusterIdentifier": "cluster-80-attempt-2",
    "Engine": "aurora-mysql",
    "EngineVersion": "8.0.mysql_aurora.3.02.0",
    "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.3.02.0",
    "DBInstanceStatus": "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.3.02.0
$ aws rds describe-db-instances \
--db-instance-identifier instance-attempt-2 \
--query '*[].[EngineVersion]' --output text
8.0.mysql_aurora.3.02.0

$ 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 \(p. 660\)](#).
- 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 \(p. 660\)](#).

### 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 version 2:

- `CREATE TABLESPACE` SQL statement
- 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 – You can install the plugin, but it isn't supported. You can't customize the plugin.
- Query rewrite plugins
- Replication filtering
- X Protocol

For more information about these features, see the [MySQL 5.7 documentation](#).

## Comparison of Aurora MySQL version 1 and Aurora MySQL version 2

The following Amazon Aurora MySQL features are supported in Aurora MySQL version 1, but currently aren't supported in Aurora MySQL version 2.

- Asynchronous key prefetch (AKP). For more information, see [Optimizing Amazon Aurora indexed join queries with asynchronous key prefetch \(p. 943\)](#).
- Scan batching. For more information, see [Aurora MySQL database engine updates 2017-12-11](#) in the [Release Notes for Aurora MySQL](#).

Currently, Aurora MySQL version 2 doesn't 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](#) in the [Release Notes for Aurora MySQL](#).

The performance schema isn't available for early releases of Aurora MySQL version 2. 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 for Amazon Aurora \(p. 1653\)](#).

If you are using IAM to access the Amazon RDS console, make sure to first sign in to the AWS Management Console with your IAM user credentials. Then go to the Amazon RDS console at <https://console.aws.amazon.com/rds/>.

- Make sure to create Aurora MySQL DB clusters in a virtual public 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 MySQL DB clusters in a VPC, use a VPC security group. You can make these endpoint and port connections by 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 VPC VPCs and Amazon Aurora \(p. 1729\)](#).

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 \(p. 56\)](#). 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 \(p. 1683\)](#).

**Note**

For more information, see [Security in Amazon Aurora \(p. 1634\)](#).

## 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 \(p. 1642\)](#).

We recommend the AWS JDBC Driver for MySQL as a client that supports SAN with SSL/TLS. 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](#).

### Topics

- [Requiring an SSL/TLS connection to an Aurora MySQL DB cluster \(p. 683\)](#)
- [TLS versions for Aurora MySQL \(p. 684\)](#)
- [Encrypting connections to an Aurora MySQL DB cluster \(p. 685\)](#)
- [Configuring cipher suites for connections to Aurora MySQL DB clusters \(p. 685\)](#)

## 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 parameter groups \(p. 215\)](#).

**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 \(p. 221\)](#). 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 \(p. 1642\)](#).

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 \(p. 1430\)](#).

**Note**

For more information on SSL/TLS connections with MySQL, see the [MySQL documentation](#).

## Configuring cipher suites for connections to Aurora MySQL DB clusters

By using configurable cipher suites, you can have more control over the security of your database connections. You can specify a list of cipher suites that you want to allow to secure client SSL/TLS connections to your database. With configurable cipher suites, you can control the connection encryption that your database server accepts. Doing this prevents the use of insecure or deprecated ciphers.

Configurable cipher suites is supported in Aurora MySQL version 3 and Aurora MySQL version 2.

To specify the list of permissible ciphers for encrypting connections, modify the `ssl_cipher` cluster parameter. Set the `ssl_cipher` parameter in a cluster parameter group using the AWS Management Console, the AWS CLI, or the RDS API.

For information about modifying parameters in a DB cluster parameter group, see [Modifying parameters in a DB cluster parameter group \(p. 221\)](#). If you use the CLI to modify the `ssl_cipher` DB cluster parameter, make sure to set the `ApplyMethod` 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.

For the client application, you can specify the ciphers to use for encrypted connections by using the `--ssl-cipher` option when connecting to the database. For more about connecting to your database, see [Connecting to an Amazon Aurora MySQL DB cluster \(p. 207\)](#).

Set the `ssl_cipher` parameter to a string of comma-separated cipher values. The following table shows the supported ciphers along with the TLS encryption protocol and valid Aurora MySQL engine versions for each cipher.

Cipher	Encryption protocol	Supported Aurora MySQL versions
DHE-RSA-AES128-SHA	TLS 1.0	3.01.0 and higher, 2.10.2, 2.10.1, 2.09.3, 2.08.4, 2.07.7, 2.04.9
DHE-RSA-AES128-SHA256	TLS 1.2	3.01.0 and higher, 2.10.2, 2.10.1, 2.09.3, 2.08.4, 2.07.7, 2.04.9
DHE-RSA-AES128-GCM-SHA256	TLS 1.2	3.01.0 and higher, 2.10.2, 2.10.1, 2.09.3, 2.08.4, 2.07.7, 2.04.9
DHE-RSA-AES256-SHA	TLS 1.0	3.01.0 and higher, 2.10.2, 2.10.1, 2.09.3, 2.08.4, 2.07.7, 2.04.9
DHE-RSA-AES256-SHA256	TLS 1.2	3.01.0 and higher, 2.10.2, 2.10.1, 2.09.3, 2.08.4, 2.07.7, 2.04.9
DHE-RSA-AES256-GCM-SHA384	TLS 1.2	3.01.0 and higher, 2.10.2, 2.10.1, 2.09.3, 2.08.4, 2.07.7, 2.04.9
ECDHE-RSA-AES128-SHA	TLS 1.0	3.01.0 and higher, 2.10.2, 2.09.3
ECDHE-RSA-AES128-SHA256	TLS 1.2	3.01.0 and higher, 2.10.2, 2.09.3
ECDHE-RSA-AES128-GCM-SHA256	TLS 1.2	3.01.0 and higher, 2.10.2, 2.09.3
ECDHE-RSA-AES256-SHA	TLS 1.0	3.01.0 and higher, 2.10.2, 2.09.3
ECDHE-RSA-AES256-SHA384	TLS 1.2	3.01.0 and higher, 2.10.2, 2.09.3
ECDHE-RSA-AES256-GCM-SHA384	TLS 1.2	3.01.0 and higher, 2.10.2, 2.09.3

You can also use the [describe-engine-default-cluster-parameters](#) CLI command to determine which cipher suites are currently supported for a specific parameter group family. The following example shows how to get the allowed values for the `ssl_cipher` cluster parameter for Aurora MySQL 5.7.

```
aws rds describe-engine-default-cluster-parameters --db-parameter-group-family aurora-mysql5.7
```

```
...some output truncated...
{
  "ParameterName": "ssl_cipher",
  "ParameterValue": "DHE-RSA-AES128-SHA,DHE-RSA-AES128-SHA256,DHE-RSA-AES128-GCM-
SHA256,DHE-RSA-AES256-SHA,DHE-RSA-AES256-SHA256,DHE-RSA-AES256-GCM-SHA384,ECDHE-RSA-AES128-
SHA,ECDHE-RSA-AES128-SHA256,ECDHE-RSA-AES128-GCM-SHA256,ECDHE-RSA-AES256-SHA,ECDHE-RSA-
AES256-SHA384,ECDHE-RSA-AES256-GCM-SHA384",
  "Description": "The list of permissible ciphers for connection encryption.",
  "Source": "system",
  "ApplyType": "static",
  "DataType": "list",
  "AllowedValues": "DHE-RSA-AES128-SHA,DHE-RSA-AES128-SHA256,DHE-RSA-AES128-GCM-SHA256,DHE-
RSA-AES256-SHA,DHE-RSA-AES256-SHA256,DHE-RSA-AES256-GCM-SHA384,ECDHE-RSA-AES128-SHA,ECDHE-
RSA-AES128-SHA256,ECDHE-RSA-AES128-GCM-SHA256,ECDHE-RSA-AES256-SHA,ECDHE-RSA-AES256-
SHA384,ECDHE-RSA-AES256-GCM-SHA384",
  "IsModifiable": true,
  "SupportedEngineModes": [
    "provisioned"
  ],
  ...
}
...some output truncated...
```

For more information about ciphers, see the [ssl\\_cipher](#) variable in the MySQL documentation. For more information about cipher suite formats, see the [openssl-ciphers list format](#) and [openssl-ciphers string format](#) documentation on the OpenSSL website.

## 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 \(p. 1644\)](#). For more information about downloading certificates, see [Using SSL/TLS to encrypt a connection to a DB cluster \(p. 1642\)](#). For information about using SSL/TLS with Aurora MySQL DB clusters, see [Using SSL/TLS with Aurora MySQL DB clusters \(p. 683\)](#).

### Topics

- [Determining whether any applications are connecting to your Aurora MySQL DB cluster using SSL \(p. 688\)](#)

- [Determining whether a client requires certificate verification to connect \(p. 688\)](#)
- [Updating your application trust store \(p. 689\)](#)
- [Example Java code for establishing SSL connections \(p. 690\)](#)

## 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.

For information about downloading the root certificate, see [Using SSL/TLS to encrypt a connection to a DB cluster \(p. 1642\)](#).

For sample scripts that import certificates, see [Sample script for importing certificates into your trust store \(p. 1651\)](#).

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.cn1e62e2e7kwh.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 \(p. 1644\)](#).

## 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 <a href="#">Migrating data from a MySQL DB instance to an Amazon Aurora MySQL DB cluster by using an Aurora read replica (p. 712)</a> .
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 <a href="#">Migrating data from a MySQL DB instance</a>

Migrating from	Migration type	Solution
		<a href="#">to an Amazon Aurora MySQL DB cluster by using a DB snapshot (p. 706).</a>
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 <a href="#">Migrating from MySQL to Amazon Aurora by using mysqldump (p. 705)</a> .
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 <a href="#">Migrating data from MySQL by using an Amazon S3 bucket (p. 693)</a> .
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 <a href="#">Loading data into an Amazon Aurora MySQL DB cluster from text files in an Amazon S3 bucket (p. 903)</a> .
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 <a href="#">What is AWS database migration service?</a>

#### 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 \(p. 705\)](#).

- 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 \(p. 693\)](#).

## 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 migrate data from a DB snapshot export to Amazon S3. You use Percona XtraBackup to back up your data to S3. For more information, see [Installing Percona XtraBackup \(p. 694\)](#).

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 restore to an Aurora Serverless DB cluster.

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 \(p. 694\)](#)
- [Backing up files to be restored as an Amazon Aurora MySQL DB cluster \(p. 695\)](#)
- [Restoring an Amazon Aurora MySQL DB cluster from an Amazon S3 bucket \(p. 697\)](#)

- [Synchronizing the Amazon Aurora MySQL DB cluster with the MySQL database using replication \(p. 700\)](#)

## 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 \(p. 896\)](#).

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",  
                "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 \(p. 1655\)](#).

### 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 \(p. 892\)](#).
2. Complete the steps in [Creating an IAM role to allow Amazon Aurora to access AWS services \(p. 897\)](#).
3. Complete the steps in [Associating an IAM role with an Amazon Aurora MySQL DB cluster \(p. 898\)](#).

### 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

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](#).

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.

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.

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*.

## 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](#)

698

#### 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 \(p. 433\)](#).

The screenshot shows the AWS RDS console interface for managing a MySQL DB cluster named 'database-1'. The top navigation bar indicates the path: RDS > Databases > database-1. The main title 'database-1' is displayed above a 'Related' section containing a search bar labeled 'Filter databases'. Below this is a table listing cluster components:

DB identifier	Role	Engine
database-1	Regional cluster	Aurora MySQL
database-1-instance-1	Writer instance	Aurora MySQL

Below the table are tabs for 'Connectivity & security' (which is selected), 'Monitoring', 'Logs & events', 'Configuration', and 'Maintenance & backup'. The 'Endpoints (2)' section displays two entries:

Endpoint name	Status
database-1.cluster-ro-... .us-east-2.rds.amazonaws.com	Available
database-1.cluster-... .us-east-2.rds.amazonaws.com	Available

The second endpoint entry is circled in red.

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 \(p. 701\)](#)
- [Synchronizing the Amazon Aurora MySQL DB cluster with the external MySQL database \(p. 702\)](#)

### 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 \(p. 702\)](#).

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 \(p. 683\)](#).

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 \(p. 683\)](#).

**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 \(p. 706\)](#)
- [Migrating data from a MySQL DB instance to an Amazon Aurora MySQL DB cluster by using an Aurora read replica \(p. 712\)](#)

### 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 \(p. 73\)](#).

## 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 \(p. 712\)](#).

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? \(p. 707\)](#)
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 &lt;schema&gt;.〈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 <a href="https://dev.mysql.com/doc/refman/5.6/en/innodb-row-format.html">https://dev.mysql.com/doc/refman/5.6/en/innodb-row-format.html</a>.</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_type'
    )
  )
)
```

```
'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)
```

## Migrating an RDS for MySQL DB snapshot to an Aurora MySQL DB cluster

You can migrate a DB snapshot of an RDS for MySQL DB instance to create an Aurora MySQL DB cluster using the AWS Management Console or the AWS CLI. 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, copy the DB snapshot to that AWS Region. For information about copying a DB snapshot, see [Copying a DB snapshot](#).

### Console

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 \(p. 67\)](#).
- **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 creating a VPC, see [Tutorial: Create a VPC for use with a DB cluster \(IPv4 only\) \(p. 1744\)](#).

Otherwise, you can choose to have Aurora create a VPC for you by selecting **Create a new VPC**.

- **DB 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 \(p. 990\)](#).

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 \(p. 207\)](#).

## AWS CLI

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 \(p. 706\)](#).

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, the AWS CLI, or the RDS API.

### 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
<b>DB instance class</b>	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 <a href="#">Aurora DB instance classes (p. 56)</a> .
<b>Multi-AZ deployment</b>	Choose <b>Create Replica in Different Zone</b> 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 <a href="#">Regions and Availability Zones (p. 11)</a> .
<b>DB instance identifier</b>	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"><li>• It must contain from 1 to 63 alphanumeric characters or hyphens.</li><li>• Its first character must be a letter.</li><li>• It cannot end with a hyphen or contain two consecutive hyphens.</li><li>• It must be unique for all DB instances for each AWS account, for each AWS Region.</li></ul> 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

Option	Description
	the same as the master user name and master password for the source DB instance.
<b>Virtual Private Cloud (VPC)</b>	Select the VPC to host the DB cluster. Select <b>Create new VPC</b> to have Aurora create a VPC for you. For more information, see <a href="#">DB cluster prerequisites (p. 127)</a> .
<b>DB subnet group</b>	Select the DB subnet group to use for the DB cluster. Select <b>Create new DB subnet group</b> to have Aurora create a DB subnet group for you. For more information, see <a href="#">DB cluster prerequisites (p. 127)</a> .
<b>Public accessibility</b>	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 <a href="#">Hiding a DB cluster in a VPC from the internet (p. 1735)</a> .
<b>Availability zone</b>	Determine if you want to specify a particular Availability Zone. For more information about Availability Zones, see <a href="#">Regions and Availability Zones (p. 11)</a> .
<b>VPC security group (firewall)</b>	Select <b>Create new VPC security group</b> to have Aurora create a VPC security group for you. Select <b>Select existing VPC security groups</b> to specify one or more VPC security groups to secure network access to the DB cluster. For more information, see <a href="#">DB cluster prerequisites (p. 127)</a> .
<b>Database port</b>	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.
<b>DB parameter group</b>	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 <a href="#">Working with parameter groups (p. 215)</a> .
<b>DB cluster parameter group</b>	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 <a href="#">Working with parameter groups (p. 215)</a> .

Option	Description
<b>Encryption</b>	<p>Choose <b>Disable encryption</b> if you don't want your new Aurora DB cluster to be encrypted. Choose <b>Enable encryption</b> for your new Aurora DB cluster to be encrypted at rest. If you choose <b>Enable encryption</b>, you must choose a KMS key as the <b>AWS KMS key</b> 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>
<b>Priority</b>	<p>Choose a failover priority for the DB cluster. If you don't select a value, the default is <b>tier-1</b>. This priority determines the order in which Aurora Replicas are promoted when recovering from a primary instance failure. For more information, see <a href="#">Fault tolerance for an Aurora DB cluster (p. 72)</a>.</p>
<b>Backup retention period</b>	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.
<b>Enhanced Monitoring</b>	Choose <b>Enable enhanced monitoring</b> to enable gathering metrics in real time for the operating system that your DB cluster runs on. For more information, see <a href="#">Monitoring OS metrics with Enhanced Monitoring (p. 518)</a> .
<b>Monitoring Role</b>	Only available if <b>Enhanced Monitoring</b> is set to <b>Enable enhanced monitoring</b> . Choose the IAM role that you created to permit Aurora to communicate with Amazon CloudWatch Logs for you, or choose <b>Default</b> to have Aurora create a role for you named <code>rds-monitoring-role</code> . For more information, see <a href="#">Monitoring OS metrics with Enhanced Monitoring (p. 518)</a> .
<b>Granularity</b>	Only available if <b>Enhanced Monitoring</b> is set to <b>Enable enhanced monitoring</b> . Set the interval, in seconds, between when metrics are collected for your DB cluster.
<b>Auto minor version upgrade</b>	<p>This setting doesn't apply to Aurora MySQL DB clusters.</p> <p>For more information about engine updates for Aurora MySQL, see <a href="#">Database engine updates for Amazon Aurora MySQL (p. 990)</a>.</p>
<b>Maintenance window</b>	Select <b>Select window</b> and specify the weekly time range during which system maintenance can occur. Or, select <b>No preference</b> 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=bee4aab750bf7dad0cd9e22b952bd6089d91e2a16592c2293e532eeaab8bc77
```

## 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.

**aurora-mysql-db-cluster**

**Details**

ARN  
arn:aws:rds: [REDACTED] :aurora-mysql-db-cluster

DB cluster  
aurora-mysql-db-cluster ( available )

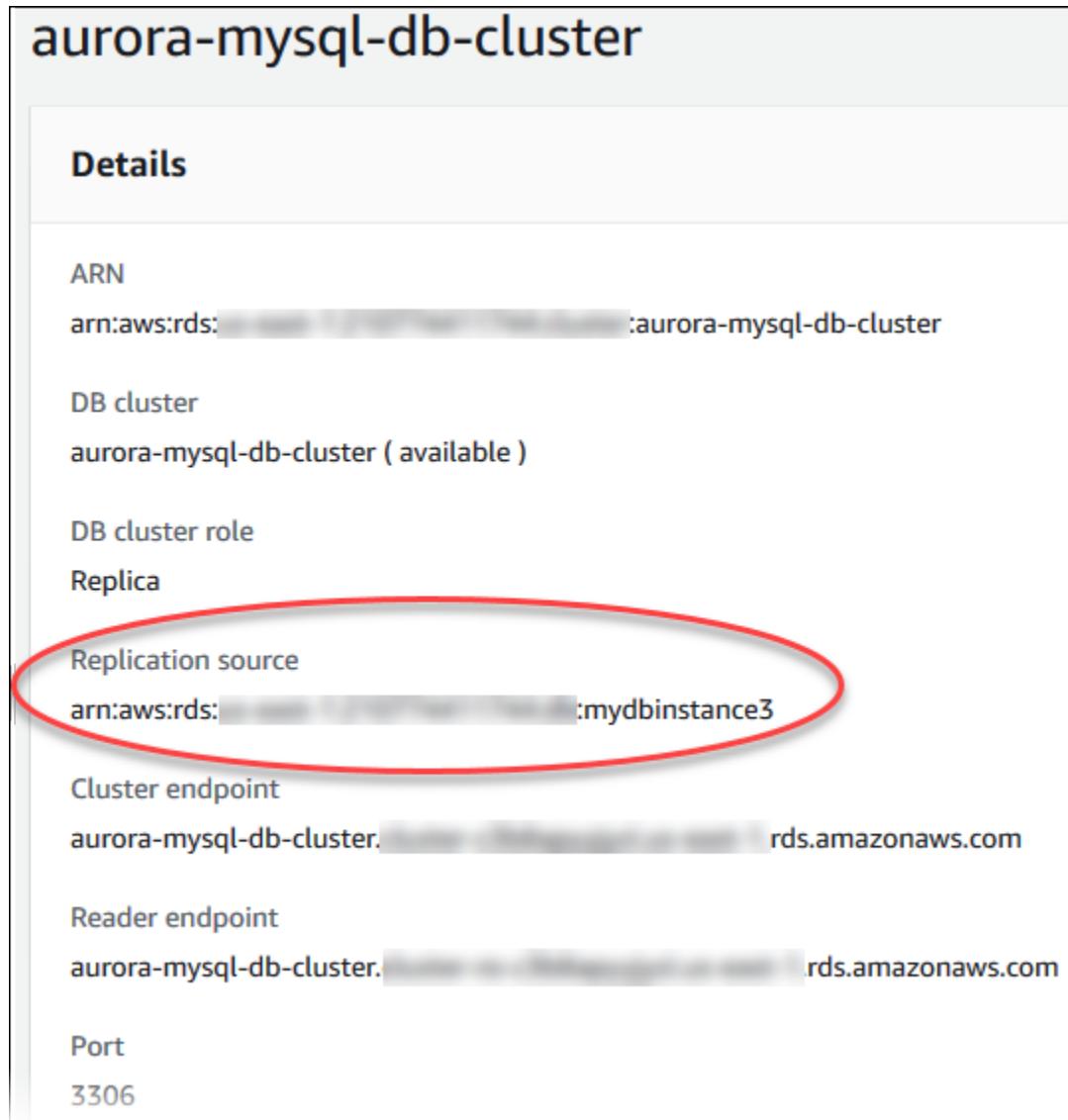
DB cluster role  
Replica

Replication source  
arn:aws:rds: [REDACTED] :mydbinstance3

Cluster endpoint  
aurora-mysql-db-cluster. [REDACTED] rds.amazonaws.com

Reader endpoint  
aurora-mysql-db-cluster. [REDACTED] 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 using the AWS Management Console or AWS CLI.

Then 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 \(p. 35\)](#). 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 REPLICA 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 \(p. 721\)](#)
- [Backtracking an Aurora DB cluster \(p. 725\)](#)
- [Testing Amazon Aurora using fault injection queries \(p. 738\)](#)
- [Altering tables in Amazon Aurora using fast DDL \(p. 741\)](#)
- [Displaying volume status for an Aurora MySQL DB cluster \(p. 746\)](#)

## 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 \(p. 278\)](#).

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 \(p. 56\)](#).

## 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.

For details about how Aurora Serverless v2 instances handle this parameter, see [Parameters that Aurora computes based on Aurora Serverless v2 maximum capacity \(p. 1537\)](#).

Instance class	max_connections 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

Instance class	max_connections default value		
db.r5.4xlarge	4000		
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.

See [Specifying DB parameters \(p. 238\)](#) for more details on how `DBInstanceClassMemory` is calculated.

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. These local storage volumes are backed by Amazon Elastic Block Store and can be extended. For more information about storage, see [Amazon Aurora storage and reliability \(p. 67\)](#).

**Note**

You might see `storage-optimization` events when scaling DB instances, for example, from `db.r5.2xlarge` to `db.r5.4xlarge`.

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)
<code>db.x2g.16xlarge</code>	1280
<code>db.x2g.12xlarge</code>	960
<code>db.x2g.8xlarge</code>	640
<code>db.x2g.4xlarge</code>	320
<code>db.x2g.2xlarge</code>	160
<code>db.x2g.xlarge</code>	80
<code>db.x2g.large</code>	40
<code>db.r6g.16xlarge</code>	1280
<code>db.r6g.12xlarge</code>	960
<code>db.r6g.8xlarge</code>	640
<code>db.r6g.4xlarge</code>	320
<code>db.r6g.2xlarge</code>	160
<code>db.r6g.xlarge</code>	80
<code>db.r6g.large</code>	32
<code>db.r5.24xlarge</code>	1920
<code>db.r5.16xlarge</code>	1280
<code>db.r5.12xlarge</code>	960
<code>db.r5.8xlarge</code>	640
<code>db.r5.4xlarge</code>	320
<code>db.r5.2xlarge</code>	160
<code>db.r5.xlarge</code>	80
<code>db.r5.large</code>	32
<code>db.r4.16xlarge</code>	1280

DB instance class	Maximum temporary storage available (GiB)
db.r4.8xlarge	640
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 \(p. 525\)](#). 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.

(This doesn't apply to Aurora Serverless v2.)

## 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 \(p. 369\)](#).

## 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 \(p. 737\)](#).

## 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 Amazon Aurora DB cluster \(p. 280\)](#).
- 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 in all AWS Regions. For information on where it is supported, see [Backtracking in Aurora \(p. 19\)](#).
- 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.

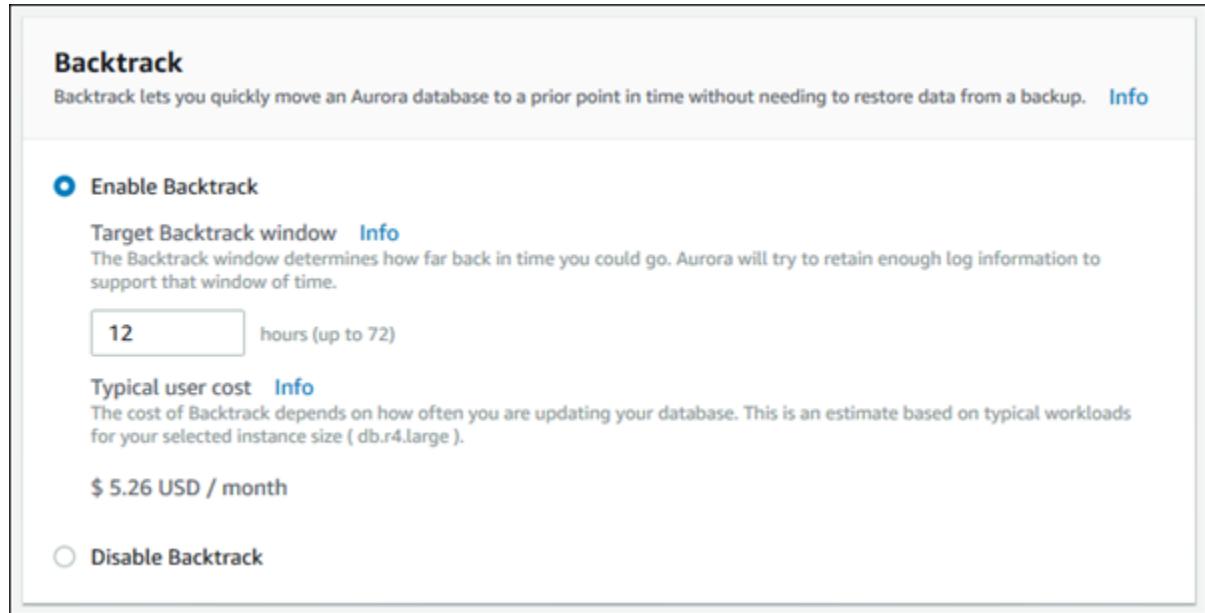
## Topics

- [Configuring backtracking with the console when creating a DB cluster \(p. 728\)](#)
- [Configuring backtrack with the console when modifying a DB cluster \(p. 728\)](#)

### 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 \(p. 127\)](#). 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.

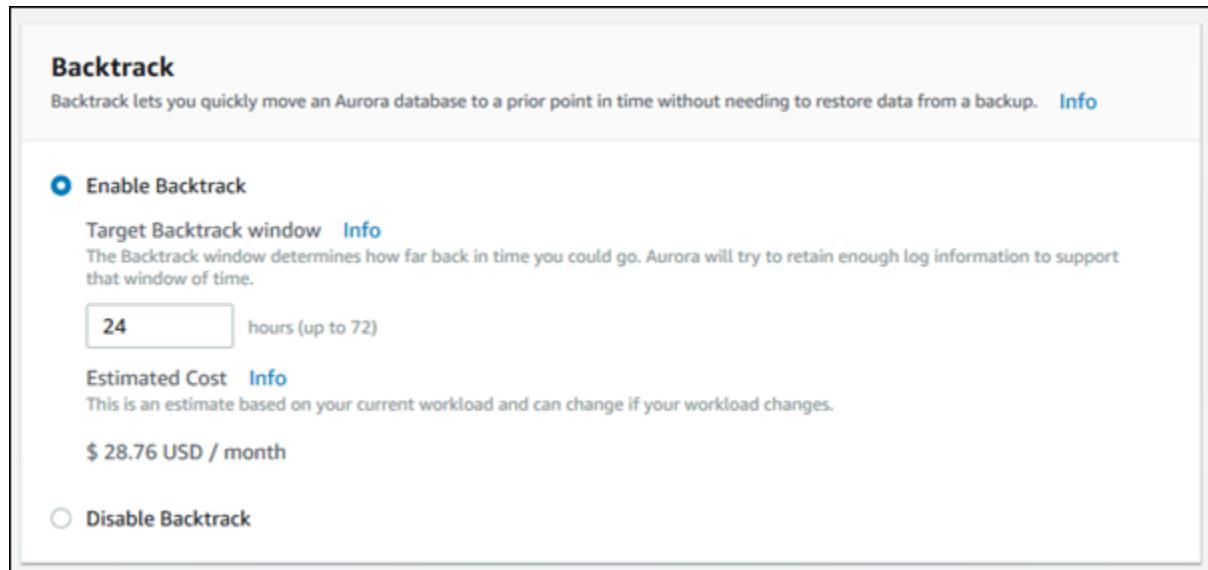
#### Note

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 \(p. 127\)](#).

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 \(p. 127\)](#).

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**

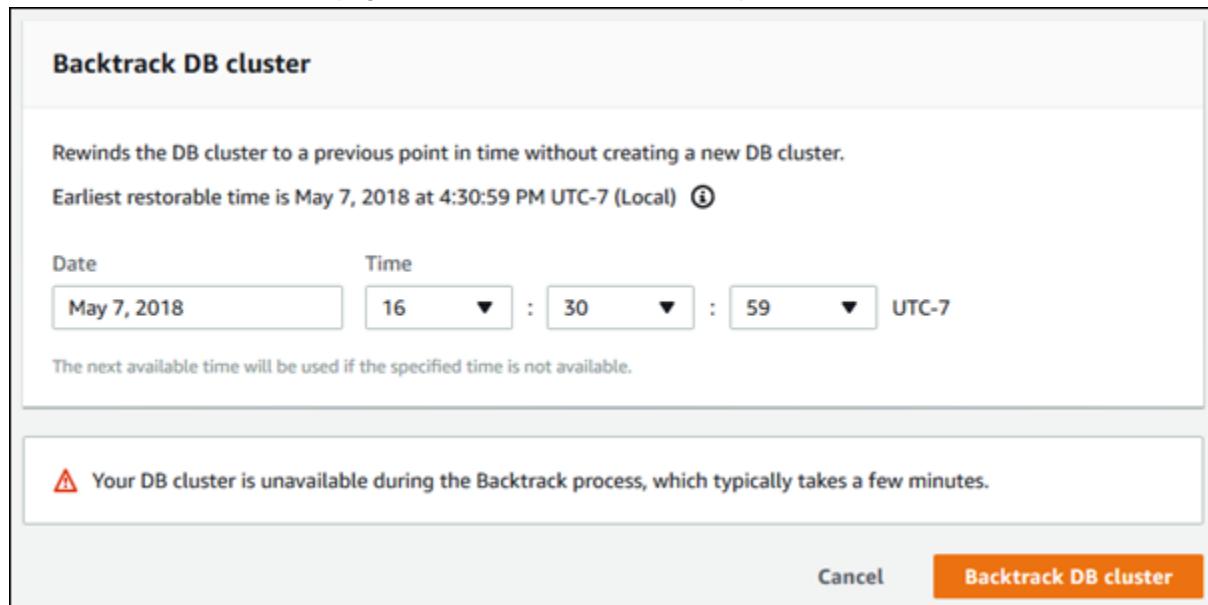
For database clones, you can't backtrack the DB cluster earlier than the date and time when the clone was created. For more information about database cloning, see [Cloning a volume for an Amazon Aurora DB cluster \(p. 280\)](#).

[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

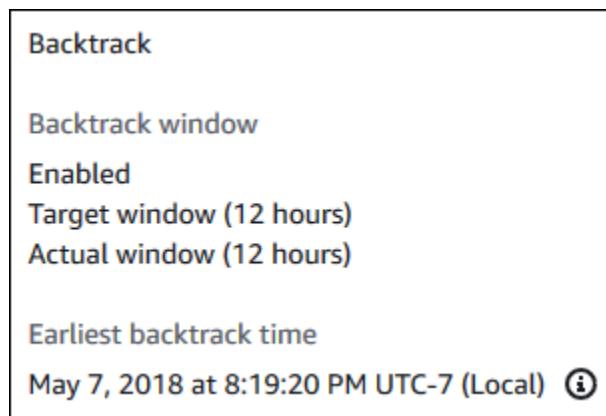
You can view backtracking information and monitor backtracking metrics for a DB cluster.

### Console

#### To view backtracking information and monitor backtracking metrics using the console

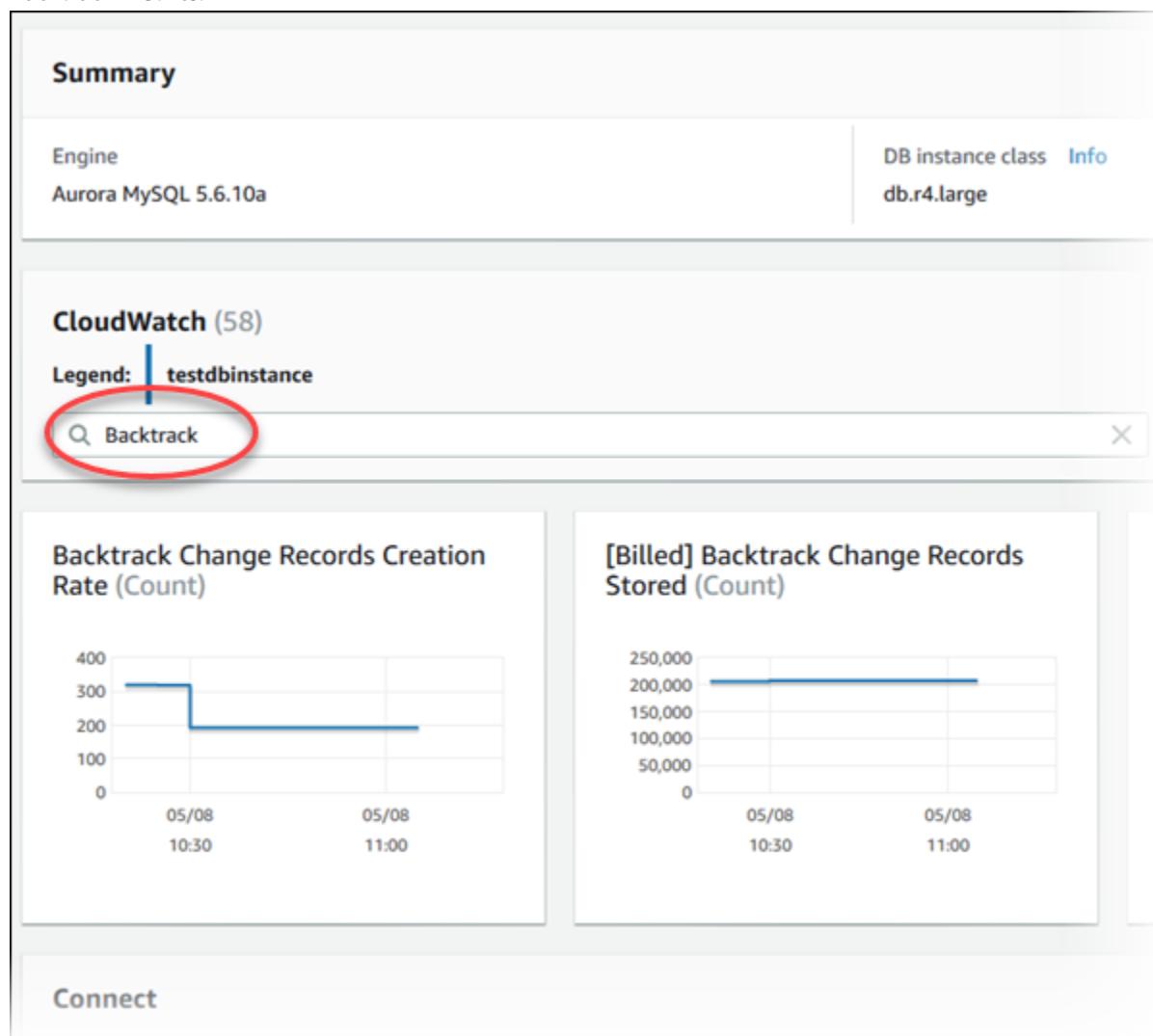
1. Sign in to the AWS Management Console and open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. Choose **Databases**.
3. Choose the DB cluster name to open information about it.

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

736

#### 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 \(p. 35\)](#).

## 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 blocks all requests from the writer instance 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 \(p. 746\)](#).
- **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 \(p. 746\)](#).
- **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 \(Aurora MySQL version 3\) \(p. 741\)](#)
- [Fast DDL \(Aurora MySQL version 1 and 2\) \(p. 743\)](#)

## 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 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 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 (0.05 sec)
```

```
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 \(p. 939\)](#).

## 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 \(p. 738\)](#).

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 \(p. 990\)](#).

## 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 \(p. 747\)](#)
- [Tuning Aurora MySQL with wait events \(p. 749\)](#)
- [Tuning Aurora MySQL with thread states \(p. 785\)](#)

## 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 \(p. 747\)](#)
- [Aurora MySQL thread states \(p. 748\)](#)
- [Aurora MySQL memory \(p. 748\)](#)
- [Aurora MySQL processes \(p. 748\)](#)

## 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 [Turning on the Performance Schema for Performance Insights on Aurora MySQL \(p. 469\)](#).

## Aurora MySQL memory

In Aurora MySQL, the most important memory areas are the buffer pool and log buffer.

### Topics

- [Buffer pool \(p. 748\)](#)

### 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 \(mysqld\) \(p. 748\)](#)
- [Threads \(p. 749\)](#)
- [Thread pool \(p. 749\)](#)

### 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 \(p. 971\)](#).

Wait event	Description
<a href="#">cpu (p. 750)</a>	This event occurs when a thread is active in CPU or is waiting for CPU.
<a href="#">io/aurora_redo_log_flush (p. 753)</a>	This event occurs when a session is writing persistent data to Aurora storage.
<a href="#">io/aurora_respond_to_client (p. 756)</a>	This event occurs when a thread is waiting to return a result set to a client.
<a href="#">io/file/innodb/innodb_data_file (p. 758)</a>	This event occurs when there are threads waiting on I/O operations from storage.
<a href="#">io/socket/sql/client_connection (p. 760)</a>	This event occurs when a thread is in the process of handling a new connection.
<a href="#">io/table/sql/handler (p. 762)</a>	This event occurs when work has been delegated to a storage engine.
<a href="#">synch/cond/mysys/my_thread_var::suspend (p. 765)</a>	This event occurs when threads are suspended because they are waiting on a condition.
<a href="#">synch/cond/sql/MDL_context::COND_wait_status (p. 766)</a>	This event occurs when there are threads waiting on a table metadata lock.
<a href="#">synch/mutex/innodb/aurora_lock_thread_slot_futex (p. 773)</a>	This event occurs when one session has locked a row for an update, and another session tries to update the same row.
<a href="#">synch/mutex/innodb/buf_pool_mutex (p. 775)</a>	This event occurs when a thread has acquired a lock on the InnoDB buffer pool to access a page in memory.
<a href="#">synch/mutex/innodb/fil_system_mutex (p. 777)</a>	This event occurs when a session is waiting to access the tablespace memory cache.

Wait event	Description
<a href="#">synch/mutex/innodb/trx_sys_mutex (p. 780)</a>	This event occurs when there is high database activity with a large number of transactions.
<a href="#">synch/rwlock/innodb/hash_table_locks (p. 781)</a>	This event occurs when there is contention on modifying the hash table that maps the buffer cache.
<a href="#">synch/sxlock/innodb/hash_table_locks (p. 783)</a>	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 \(p. 750\)](#)
- [Context \(p. 750\)](#)
- [Likely causes of increased waits \(p. 751\)](#)
- [Actions \(p. 751\)](#)

### 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 \(p. 751\)](#)
- [Analyze and optimize the high CPU workload \(p. 752\)](#)

### 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 \(p. 476\)](#).

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 \(p. 753\)](#)
- [Context \(p. 753\)](#)
- [Likely causes of increased waits \(p. 753\)](#)
- [Actions \(p. 754\)](#)

### 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

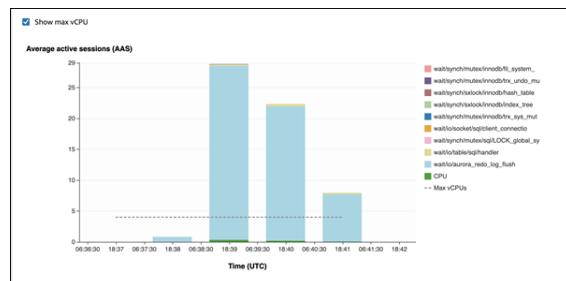
The `io/aurora_redo_log_flush` event is for a write input/output (I/O) operation in Aurora MySQL.

### Likely causes of increased waits

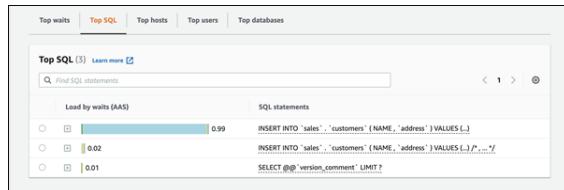
For data persistence, commits require a durable write to stable storage. If the database is doing too many commits, there is a wait event on the write I/O operation, the `io/aurora_redo_log_flush` wait event.

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.

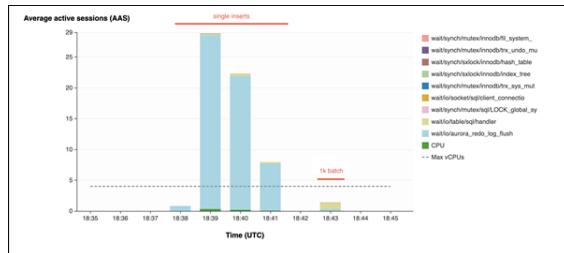


This is caused by the simple insert statements used.



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 \(p. 756\)](#)
- [Context \(p. 756\)](#)
- [Likely causes of increased waits \(p. 756\)](#)
- [Actions \(p. 757\)](#)

### 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 \(p. 757\)](#)
- [Scale the DB instance class \(p. 757\)](#)
- [Check workload for unexpected results \(p. 757\)](#)
- [Distribute workload with reader instances \(p. 758\)](#)
- [Use the SQL\\_BUFFER\\_RESULT modifier \(p. 758\)](#)

### 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 \(p. 449\)](#). For information about DB instance classes, see [Aurora DB instance classes \(p. 56\)](#). For information about modifying a DB cluster, see [Modifying an Amazon Aurora DB cluster \(p. 248\)](#).

### 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 \(p. 546\)](#).

### 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 \(p. 758\)](#)
- [Context \(p. 758\)](#)
- [Likely causes of increased waits \(p. 758\)](#)
- [Actions \(p. 759\)](#)

### 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 \(p. 64\)](#).
- 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 \(p. 759\)](#)
- [Scale up your instance \(p. 759\)](#)
- [Make your buffer scan resistant \(p. 760\)](#)

### 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 \(p. 64\)](#).
- 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 \(p. 760\)](#)
- [Context \(p. 760\)](#)
- [Likely causes of increased waits \(p. 760\)](#)
- [Actions \(p. 760\)](#)

### 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 \(mysqld\) \(p. 748\)](#).

### 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 \(p. 761\)](#)
- [Follow best practices for connection management \(p. 761\)](#)

- [Scale up your instance if resources are being throttled \(p. 761\)](#)
- [Check the top hosts and top users \(p. 762\)](#)
- [Query the performance\\_schema tables \(p. 762\)](#)
- [Check the thread states of your queries \(p. 762\)](#)
- [Audit your requests and queries \(p. 762\)](#)
- [Pool your database connections \(p. 762\)](#)

## 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 \(p. 73\)](#) and [Amazon Aurora connection management \(p. 35\)](#).

## 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 \(p. 56\)](#).

- 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 \(p. 518\)](#).

## 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 \(p. 476\)](#).

## 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 \(p. 823\)](#).

## 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 \(p. 1430\)](#).

# io/table/sql/handler

The `io/table/sql/handler` event occurs when work has been delegated to a storage engine.

## Topics

- [Supported engine versions \(p. 762\)](#)
- [Context \(p. 763\)](#)
- [Likely causes of increased waits \(p. 763\)](#)
- [Actions \(p. 763\)](#)

## 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 \(p. 763\)](#)
- [Check for a correlation with Performance Insights counter metrics \(p. 764\)](#)
- [Check for other correlated wait events \(p. 764\)](#)

### [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 \(p. 765\)](#)
- [Context \(p. 765\)](#)
- [Likely causes of increased waits \(p. 765\)](#)
- [Actions \(p. 766\)](#)

### [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 \(p. 766\)](#)
- [Make sure that backup tools don't lock tables \(p. 766\)](#)
- [Long-running sessions that lock tables \(p. 766\)](#)
- [Non-InnoDB temporary table \(p. 766\)](#)

### 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 \(p. 823\)](#).

### 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 \(p. 767\)](#)
- [Context \(p. 767\)](#)
- [Likely causes of increased waits \(p. 767\)](#)
- [Actions \(p. 768\)](#)

## 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 \(p. 768\)](#)
- [Check for past events \(p. 768\)](#)
- [Run queries on Aurora MySQL version 1 \(p. 769\)](#)
- [Run queries on Aurora MySQL version 2 \(p. 771\)](#)
- [Respond to the blocking session \(p. 773\)](#)

### 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                   | NULL          |          |          |        |       |
+-----+-----+-----+-----+-----+-----+-----+
+
| 14 | rdsadmin     | localhost     | NULL    | Sleep   |    1 | 
| cleaned up                   | NULL          |          |          |        |       |
+-----+-----+-----+-----+-----+-----+-----+
```

15   rdsadmin	localhost	NULL	Sleep	14
cleaned up	NULL			
16   rdsadmin	localhost	NULL	Sleep	1
cleaned up	NULL			
17   rdsadmin	localhost	NULL	Sleep	214
cleaned up	NULL			
40   auroramysql56123	172.31.21.51:44876	sbtest123	Query	1843   User
sleep	select sleep(10000)			
41   auroramysql56123	172.31.21.51:44878	performance_schema	Query	0   init
show processlist				
48   auroramysql56123	172.31.21.51:44894	sbtest123	Execute	0
delayed commit ok initiated	COMMIT			
49   auroramysql56123	172.31.21.51:44899	sbtest123	Execute	0
delayed commit ok initiated	COMMIT			
50   auroramysql56123	172.31.21.51:44896	sbtest123	Execute	0
delayed commit ok initiated	COMMIT			
51   auroramysql56123	172.31.21.51:44892	sbtest123	Execute	0
delayed commit ok initiated	COMMIT			
52   auroramysql56123	172.31.21.51:44898	sbtest123	Execute	0
delayed commit ok initiated	COMMIT			
53   auroramysql56123	172.31.21.51:44902	sbtest	Query	33
Waiting for table metadata lock   INSERT INTO sbtest1 (id, k, c, pad) VALUES (0, 5021, '91560616281-61537173720-56678788409-8805377477				
56   auroramysql56123	172.31.21.51:44908	NULL	Query	118   User
sleep	select sleep(10000)			
58   auroramysql56123	172.31.21.51:44912	NULL	Sleep	41
cleaned up	NULL			
59   auroramysql56123	172.31.21.51:44914	NULL	Query	33
Waiting for table metadata lock   truncate table sbtest.sbtest1				

+-----+-----+-----+-----  
+-----+-----+-----  
+-----+-----+-----  
+-----+-----+-----  
+-----+-----+-----  
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 |           108
+-----+-----+-----+
+-----+-----+-----+
+-----+-----+-----+
+-----+-----+-----+
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 \(p. 774\)](#)
- [Find and respond to the blocking session \(p. 774\)](#)

### [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 0xae915060700, 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 \(p. 775\)](#)
- [Context \(p. 776\)](#)
- [Likely causes of increased waits \(p. 776\)](#)
- [Actions \(p. 776\)](#)

### 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 \(p. 305\)](#).

## 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 \(p. 777\)](#)
- [Context \(p. 777\)](#)
- [Likely causes of increased waits \(p. 778\)](#)
- [Actions \(p. 778\)](#)

## 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 \(p. 778\)](#)
- [Reorganize large tables during off-peak hours \(p. 779\)](#)

### 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: fil0fil.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: fil0fil.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: fil0fil.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 \(p. 780\)](#)
- [Context \(p. 780\)](#)
- [Likely causes of increased waits \(p. 780\)](#)
- [Actions \(p. 781\)](#)

### 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 \(p. 781\)](#)
- [Context \(p. 782\)](#)
- [Likely causes of increased waits \(p. 782\)](#)
- [Actions \(p. 782\)](#)

### [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 \(p. 782\)](#)
- [Improve data access patterns \(p. 782\)](#)
- [Find SQL queries responsible for high load \(p. 782\)](#)
- [Reduce or avoid full-table scans \(p. 783\)](#)

### 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 \(p. 783\)](#)
- [Context \(p. 783\)](#)
- [Likely causes of increased waits \(p. 784\)](#)
- [Actions \(p. 784\)](#)

### [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 \(p. 784\)](#)
- [Improve data access patterns \(p. 784\)](#)
- [Reduce or avoid full-table scans \(p. 785\)](#)
- [Check the error logs for page corruption \(p. 785\)](#)

### 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**.
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.

### 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
<a href="#">???</a> (p. 785)	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.
<a href="#">???</a> (p. 788)	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 \(p. 785\)](#)
- [Context \(p. 785\)](#)
- [Likely causes of increased waits \(p. 786\)](#)
- [Actions \(p. 786\)](#)

### 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 \(p. 786\)](#)
- [Identify the problem queries \(p. 786\)](#)
- [Examine the explain plans for filesort usage \(p. 786\)](#)
- [Increase the sort buffer size \(p. 787\)](#)

### 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 [Turning on the Performance Schema for Performance Insights on Aurora MySQL \(p. 469\)](#).

### 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 \(p. 786\)](#).

### 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 \(p. 788\)](#)
- [Context \(p. 788\)](#)
- [Likely causes of increased waits \(p. 788\)](#)
- [Actions \(p. 789\)](#)

### 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 \(p. 788\)](#)
- [Suboptimal server configuration \(p. 789\)](#)

### 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 \(p. 789\)](#)
- [Examine memory settings \(p. 789\)](#)
- [Examine the explain plans for index usage \(p. 790\)](#)
- [Check the volume of data returned \(p. 790\)](#)
- [Check for concurrency issues \(p. 790\)](#)
- [Check the structure of your queries \(p. 790\)](#)

### [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 [Turning on the Performance Schema for Performance Insights on Aurora MySQL \(p. 469\)](#).

### [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 \(p. 27\)](#).

### Contents

- [Overview of parallel query for Aurora MySQL \(p. 791\)](#)
  - [Benefits \(p. 792\)](#)
  - [Architecture \(p. 792\)](#)
  - [Prerequisites \(p. 793\)](#)

- Limitations (p. 793)
- Planning for a parallel query cluster (p. 794)
  - Checking Aurora MySQL version compatibility for parallel query (p. 794)
- Creating a DB cluster that works with parallel query (p. 795)
  - Creating a parallel query cluster using the console (p. 796)
  - Creating a parallel query cluster using the CLI (p. 797)
- Turning parallel query on and off (p. 799)
  - Aurora MySQL 1.23 and 2.09 or higher (p. 799)
  - Before Aurora MySQL 1.23 (p. 800)
  - Turning on hash join for parallel query clusters (p. 800)
  - Turning on and turning off parallel query using the console (p. 801)
  - Turning on and turning off parallel query using the CLI (p. 801)
- Upgrade considerations for parallel query (p. 802)
  - Upgrading parallel query clusters to Aurora MySQL version 3 (p. 802)
  - Upgrading to Aurora MySQL 1.23 or 2.09 and higher (p. 802)
- Performance tuning for parallel query (p. 803)
- Creating schema objects to take advantage of parallel query (p. 803)
- Verifying which statements use parallel query (p. 803)
- Monitoring parallel query (p. 806)
- How parallel query works with SQL constructs (p. 810)
  - EXPLAIN statement (p. 811)
  - WHERE clause (p. 812)
  - Data definition language (DDL) (p. 813)
  - Column data types (p. 813)
  - Partitioned tables (p. 814)
  - Aggregate functions, GROUP BY clauses, and HAVING clauses (p. 815)
  - Function calls in WHERE clause (p. 815)
  - LIMIT clause (p. 816)
  - Comparison operators (p. 816)
  - Joins (p. 817)
  - Subqueries (p. 818)
  - UNION (p. 818)
  - Views (p. 819)
  - Data manipulation language (DML) statements (p. 819)
  - Transactions and locking (p. 820)
  - B-tree indexes (p. 822)
  - Full-text search (FTS) indexes (p. 822)
  - Virtual columns (p. 822)
  - Built-in caching mechanisms (p. 822)
  - MyISAM temporary tables (p. 823)

## 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 \(p. 803\)](#).

**Topics**

- [Benefits \(p. 792\)](#)
- [Architecture \(p. 792\)](#)
- [Prerequisites \(p. 793\)](#)
- [Limitations \(p. 793\)](#)

## 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 \(p. 802\)](#) 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 \(p. 800\)](#).

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_pg_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 \(p. 810\)](#).

- 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 \(p. 806\)](#). 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 \(p. 27\)](#).
- 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 \(p. 803\)](#).

## 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 \(p. 803\)](#).

## 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 \(p. 127\)](#).

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 \(p. 127\)](#).
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 \(p. 803\)](#).

## 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 \(p. 794\)](#).
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 \(p. 799\)](#).

3. Follow the general AWS CLI procedure in [Creating an Amazon Aurora DB cluster \(p. 127\)](#).
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 \(p. 794\)](#) 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
```

- 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

- 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 \(p. 794\)](#). 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.
- Locate an Aurora MySQL-compatible cluster snapshot.
- Follow the general AWS CLI procedure in [Restoring from a DB cluster snapshot \(p. 375\)](#).
- 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 \(p. 797\)](#).

## 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 \(p. 803\)](#) and [How parallel query works with SQL constructs \(p. 810\)](#).

## 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 parameter groups \(p. 215\)](#). 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, it doesn't require a cluster restart after changing this setting. However, any connections that were using parallel query before toggling the option will continue to do so until the connection is closed, or the instance is rebooted.

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 parameter groups \(p. 215\)](#). 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 instances 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 \(p. 803\)](#) and [How parallel query works with SQL constructs \(p. 810\)](#).

## 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. For information about how to use hash joins effectively, see [Optimizing large Aurora MySQL join queries with hash joins \(p. 945\)](#).

## 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 parameter groups \(p. 215\)](#).
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 DVB 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 \(p. 813\)](#), [Partitioned tables \(p. 814\)](#), and [Aggregate functions, GROUP BY clauses, and HAVING clauses \(p. 815\)](#).

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 \(p. 802\)](#).

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 \(p. 996\)](#) 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 \(p. 799\)](#) and [Turning on hash join for parallel query clusters \(p. 800\)](#).

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 \(p. 461\)](#).
- Backtracking. For more information, see [Backtracking an Aurora DB cluster \(p. 725\)](#).
- Stopping and starting the cluster. For more information, see [Stopping and starting an Amazon Aurora DB cluster \(p. 244\)](#).

## 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 \(p. 803\)](#).
- Monitor which queries use parallel query. To learn how, see [Monitoring parallel query \(p. 806\)](#).
- 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 \(p. 803\)](#).
- 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 \(p. 810\)](#).

## 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 \(p. 793\)](#) and [Limitations \(p. 793\)](#).

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 \(p. 810\)](#).

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_shippriority
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	filtered	Extra
1	SIMPLE	customer	NULL	ALL	NULL	NULL	NULL	NULL	1480234	10.00	Using where; Using temporary; Using filesort
1	SIMPLE	orders	NULL	ALL	NULL	NULL	NULL	NULL	14875240	3.33	Using where; Using join buffer (Block Nested Loop)
1	SIMPLE	lineitem	NULL	ALL	NULL	NULL	NULL	NULL	59270573	3.33	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 \(p. 945\)](#).

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;
|     |             |            |
| 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.

```
+---+...
+-----+-----+
+-----+...| Extra
| id | ...| Extra
|     |     |
+-----+...
+-----+...| 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 \(p. 810\)](#).

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_shipppriority |
+-----+-----+-----+
```

```
| 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_shipppriority |
+-----+-----+-----+
| 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
<code>p_partkey</code>	<code>int(11)</code>	NO	PRI	<code>NULL</code>	
<code>p_name</code>	<code>varchar(55)</code>	NO		<code>NULL</code>	
<code>p_mfgr</code>	<code>char(25)</code>	NO		<code>NULL</code>	
<code>p_brand</code>	<code>char(10)</code>	NO		<code>NULL</code>	
<code>p_type</code>	<code>varchar(25)</code>	NO		<code>NULL</code>	
<code>p_size</code>	<code>int(11)</code>	NO		<code>NULL</code>	
<code>p_container</code>	<code>char(10)</code>	NO		<code>NULL</code>	
<code>p_retailprice</code>	<code>decimal(15,2)</code>	NO		<code>NULL</code>	
<code>p_comment</code>	<code>varchar(23)</code>	NO		<code>NULL</code>	

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 \(p. 449\)](#), 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.
Aurora_pq_request_not_chosen_few_pages_outside_buffer_pool	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.
Aurora_pq_max_concurrent_requests	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.
Aurora_pq_request_in_progress	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 Aurora_pq_max_concurrent_requests.
Aurora_pq_request_throttled	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.
Aurora_pq_request_not_chosen_long trx	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.
Aurora_pq_request_not_chosen_unsupported_access	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.
Aurora_pq_request_not_chosen_column_bit	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.
Aurora_pq_request_not_chosen_column_geom	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 <a href="#">Upgrading parallel query clusters to Aurora MySQL version 3 (p. 802)</a> .

<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 <a href="#">Upgrading parallel query clusters to Aurora MySQL version 3 (p. 802)</a> .
<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_too_long</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.

Aurora_pq_request_not_chosen_tx_isolation	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.
Aurora_pq_request_not_chosen_update_delete	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 \(p. 811\)](#)
- [WHERE clause \(p. 812\)](#)
- [Data definition language \(DDL\) \(p. 813\)](#)
- [Column data types \(p. 813\)](#)
- [Partitioned tables \(p. 814\)](#)
- [Aggregate functions, GROUP BY clauses, and HAVING clauses \(p. 815\)](#)
- [Function calls in WHERE clause \(p. 815\)](#)
- [LIMIT clause \(p. 816\)](#)
- [Comparison operators \(p. 816\)](#)
- [Joins \(p. 817\)](#)
- [Subqueries \(p. 818\)](#)
- [UNION \(p. 818\)](#)
- [Views \(p. 819\)](#)
- [Data manipulation language \(DML\) statements \(p. 819\)](#)
- [Transactions and locking \(p. 820\)](#)
- [B-tree indexes \(p. 822\)](#)
- [Full-text search \(FTS\) indexes \(p. 822\)](#)
- [Virtual columns \(p. 822\)](#)
- [Built-in caching mechanisms \(p. 822\)](#)
- [MyISAM temporary tables \(p. 823\)](#)

## 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 |
+-----+-----+-----+-----+-----+-----+
| 1 | SIMPLE | part | 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 \(Aurora MySQL version 3\) \(p. 741\)](#).

## 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)
+----+....+
+-----+
| 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 \(p. 800\)](#). For information about how to use hash joins effectively, see [Optimizing large Aurora MySQL join queries with hash joins \(p. 945\)](#).

```
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 \(p. 978\)](#).

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_pq_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 \(p. 597\)](#).

**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 \(p. 924\)](#).

## 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 \(p. 825\)](#) and [server\\_audit\\_excl\\_users \(p. 825\)](#).

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 \(p. 231\)](#) 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 \(p. 824\)](#)
- [server\\_audit\\_events \(p. 824\)](#)
- [server\\_audit\\_incl\\_users \(p. 825\)](#)
- [server\\_audit\\_excl\\_users \(p. 825\)](#)

## 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 \(p. 597\)](#).

## 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 \(p. 597\)](#) and [Downloading a database log file \(p. 598\)](#).

## 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 \(p. 619\)](#).

The audit log files include the following comma-delimited information in rows, in the specified order:

Field	Description
timestamp	The Unix time stamp for the logged event with microsecond precision.
serverhost	The name of the instance that the event is logged for.
username	The connected user name of the user.
host	The host that the user connected from.
connectionid	The connection ID number for the logged operation.
queryid	The query ID number, which can be used for finding the relational table events and related queries. For TABLE events, multiple lines are added.
operation	The recorded action type. Possible values are: CONNECT, QUERY, READ, WRITE, CREATE, ALTER, RENAME, and DROP.
database	The active database, as set by the USE command.
object	For QUERY events, this value indicates the query that the database performed. For TABLE 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 \(p. 867\)](#).

**Topics**

- [Using Aurora Replicas \(p. 827\)](#)
- [Replication options for Amazon Aurora MySQL \(p. 828\)](#)
- [Performance considerations for Amazon Aurora MySQL replication \(p. 828\)](#)
- [Zero-downtime restart \(ZDR\) for Amazon Aurora MySQL \(p. 829\)](#)
- [Monitoring Amazon Aurora MySQL replication \(p. 830\)](#)
- [Replicating Amazon Aurora MySQL DB clusters across AWS Regions \(p. 831\)](#)
- [Replication between Aurora and MySQL or between Aurora and another Aurora DB cluster \(binary log replication\) \(p. 841\)](#)
- [Using GTID-based replication for Amazon Aurora MySQL \(p. 863\)](#)

## 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 \(p. 73\)](#).

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 \(p. 831\)](#).

- 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 \(binary log replication\) \(p. 841\)](#).

- 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 \(p. 706\)](#).

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 \(p. 854\)](#).

- 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 \(p. 151\)](#).

**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	No	No	ZDR isn't available for these versions.

Aurora MySQL version	Does ZDR apply to the writer?	Does ZDR apply to readers?	Notes
version 1.*, 1.17.3 and lower			
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"> <li>• Aurora doesn't use the ZDR mechanism if binary logging is turned on for the DB instance.</li> <li>• Aurora rolls back any transactions that are in progress on active connections. Your application must retry the transactions.</li> <li>• Aurora cancels any connections that use TLS/SSL, temporary tables, table locks, or user locks.</li> </ul>
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"> <li>• Aurora rolls back any transactions that are in progress on active connections. Your application must retry the transactions.</li> <li>• Aurora cancels any connections that use TLS/SSL, temporary tables, table locks, or user locks.</li> </ul>
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 \(p. 427\)](#).

## 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 \(p. 151\)](#).

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 \(p. 832\)](#)
- [Creating an Amazon Aurora MySQL DB cluster that is a cross-Region read replica \(p. 832\)](#)
- [Viewing Amazon Aurora MySQL cross-Region replicas \(p. 839\)](#)
- [Promoting a read replica to be a DB cluster \(p. 839\)](#)
- [Troubleshooting Amazon Aurora MySQL cross Region replicas \(p. 840\)](#)

## 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 \(binary log replication\) \(p. 841\)](#). For more information about modifying Aurora MySQL configuration parameters, see [Amazon Aurora DB cluster and DB instance parameters \(p. 217\)](#) and [Working with parameter groups \(p. 215\)](#).

## 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 **Databases**.

4. Choose the DB cluster for which you want to create a cross-Region read replica.
5. For **Actions**, choose **Create cross-Region read replica**.
6. 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
<b>Destination region</b>	Choose the AWS Region to host the new cross-Region read replica DB cluster.
<b>Destination DB subnet group</b>	Choose the DB subnet group to use for the cross-Region read replica DB cluster.
<b>Publicly accessible</b>	Choose <b>Yes</b> to give the cross-Region read replica DB cluster a public IP address; otherwise, select <b>No</b> .
<b>Encryption</b>	Select <b>Enable Encryption</b> to turn on encryption at rest for this DB cluster. For more information, see <a href="#">Encrypting Amazon Aurora resources (p. 1638)</a> .
<b>AWS KMS key</b>	Only available if <b>Encryption</b> is set to <b>Enable Encryption</b> . Select the AWS KMS key to use for encrypting this DB cluster. For more information, see <a href="#">Encrypting Amazon Aurora resources (p. 1638)</a> .
<b>DB instance class</b>	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 <a href="#">Aurora DB instance classes (p. 56)</a> .
<b>Multi-AZ deployment</b>	Choose <b>Yes</b> 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 <a href="#">Regions and Availability Zones (p. 11)</a> .
<b>Read replica source</b>	Choose the source DB cluster to create a cross-Region read replica for.
<b>DB instance identifier</b>	<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"> <li>• It must contain from 1 to 63 alphanumeric characters or hyphens.</li> <li>• Its first character must be a letter.</li> <li>• It cannot end with a hyphen or contain two consecutive hyphens.</li> <li>• It must be unique for all DB instances for each AWS account, for each AWS Region.</li> </ul> <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</p>

Option	Description
	replica are the same as the master user name and master password for the source DB cluster.
<b>DB cluster identifier</b>	<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 <a href="#">Amazon Aurora connection management (p. 35)</a>.</p> <p>The DB cluster identifier has the following constraints:</p> <ul style="list-style-type: none"> <li>• It must contain from 1 to 63 alphanumeric characters or hyphens.</li> <li>• Its first character must be a letter.</li> <li>• It cannot end with a hyphen or contain two consecutive hyphens.</li> <li>• It must be unique for all DB clusters for each AWS account, for each AWS Region.</li> </ul>
<b>Priority</b>	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 <b>tier-1</b> . For more information, see <a href="#">Fault tolerance for an Aurora DB cluster (p. 72)</a> .
<b>Database port</b>	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.
<b>Enhanced monitoring</b>	Choose <b>Enable enhanced monitoring</b> to turn on gathering metrics in real time for the operating system that your DB cluster runs on. For more information, see <a href="#">Monitoring OS metrics with Enhanced Monitoring (p. 518)</a> .
<b>Monitoring Role</b>	Only available if <b>Enhanced Monitoring</b> is set to <b>Enable enhanced monitoring</b> . Choose the IAM role that you created to permit Amazon RDS to communicate with Amazon CloudWatch Logs for you, or choose <b>Default</b> to have RDS create a role for you named <code>rds-monitoring-role</code> . For more information, see <a href="#">Monitoring OS metrics with Enhanced Monitoring (p. 518)</a> .
<b>Granularity</b>	Only available if <b>Enhanced Monitoring</b> is set to <b>Enable enhanced monitoring</b> . Set the interval, in seconds, between when metrics are collected for your DB cluster.

Option	Description
<b>Auto minor version upgrade</b>	This setting doesn't apply to Aurora MySQL DB clusters.  For more information about engine updates for Aurora MySQL, see <a href="#">Database engine updates for Amazon Aurora MySQL (p. 990)</a> .

7. 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, specify the `--kms-key-id` option and the `--storage-encrypted` option.

#### 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`.

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 \  
--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 ^  
--storage-encrypted
```

The `--source-region` option is required for cross-Region replication between the AWS GovCloud (US-East) and AWS GovCloud (US-West) Regions, where the DB cluster identified by `--replication-source-identifier` is encrypted. For `--source-region`, specify the AWS Region of the source DB cluster.

If `--source-region` isn't specified, specify a `--pre-signed-url` value. A *presigned URL* is a URL that contains a Signature Version 4 signed request for the `create-db-cluster` command that is called in the source AWS Region. To learn more about the `pre-signed-url` option, see [create-db-cluster in the AWS CLI Command Reference](#).

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](#) operation 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, specify the `KmsKeyId` parameter and set the `StorageEncrypted` parameter to `true`.

#### 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%252Frds.us-west-2.amazonaws.com%252F  
        %253Action%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
```

For cross-Region replication between the AWS GovCloud (US-East) and AWS GovCloud (US-West) Regions, where the DB cluster identified by `ReplicationSourceIdentifier` is encrypted, 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.

2. Check that the DB cluster has become available to use by using the RDS API [DescribeDBClusters](#) operation, 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=84c2e4f8fba7c577ac5d820711e34c6e4ffcd35be8a6b7c50f329a74f35f426
```

When `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. 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 \(p. 832\)](#).

### 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, make sure that your DB clusters and DB instances are publicly accessible. If the Aurora MySQL DB clusters are in private subnets in your VPC, use VPC peering between the AWS Regions. For more information, see [A DB cluster in a VPC accessed by an EC2 instance in a different VPC \(p. 1741\)](#).

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 \(p. 831\)](#).

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 Amazon Aurora MySQL \(p. 863\)](#). 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 \(Aurora MySQL version 3 and higher\) \(p. 984\)](#).

### 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 in this topic:

1. Turn on binary logging on the replication source (p. 842)
2. Retain binary logs on the replication source until no longer needed (p. 845)
3. Create a snapshot of your replication source (p. 847)
4. Load the snapshot into your replica target (p. 848)
5. Create a replication user on your replication source (p. 850)
6. Turn on replication on your replica target (p. 850)
7. Monitor your replica (p. 852)

## 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><b>To turn on binary logging on an Aurora MySQL DB cluster</b></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're 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 <a href="#">Amazon Aurora DB cluster and DB instance parameters (p. 217)</a> and <a href="#">Working with parameter groups (p. 215)</a>.</p>
RDS for MySQL	<p><b>To turn on binary logging on an Amazon RDS DB instance</b></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"><li>• 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 <a href="#">Creating a DB instance</a> in the <i>Amazon RDS User Guide</i>.</li><li>• Create a read replica for the DB instance. For more information, see <a href="#">Working with read replicas</a> in the <i>Amazon RDS User Guide</i>.</li></ul>
MySQL (external)	<p><b>To set up encrypted replication</b></p> <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><b>Note</b> If you don't need to use encrypted replication, you can skip these steps.</p>

Database engine	Instructions
	<p>The following are prerequisites for using encrypted replication:</p> <ul style="list-style-type: none"> <li>Secure Sockets Layer (SSL) must be enabled on the external MySQL source database.</li> <li>A client key and client certificate must be prepared for the Aurora MySQL DB cluster.</li> </ul> <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"> <li>Ensure that you are prepared for encrypted replication:           <ul style="list-style-type: none"> <li>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.</li> <li>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.</li> </ul> </li> </ol> <p>For more information, see <a href="#">Creating SSL certificates and keys using openssl</a> 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"> <li>Connect to the Aurora MySQL DB cluster as the master user using SSL.</li> </ol> <p>For information about connecting to an Aurora MySQL DB cluster with SSL, see <a href="#">Using SSL/TLS with Aurora MySQL DB clusters (p. 683)</a>.</p> <ol style="list-style-type: none"> <li>Run the <code>mysql.rds_import_binlog_ssl_material</code> stored procedure to import the SSL information into the Aurora MySQL DB cluster.</li> </ol> <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\nlsLnBItnckiJ7FbtxJMLLvvwJryDUilBMTjYtwB+QhYXUMOzce5Pjz5/i8SeJtjnV3iAoG/\ncQk+0Fzz\nqaeJAAHco+CY/5WrUBkrHmFJr6HcXkvJdWPkYQS3xqC0+FmUZofz221CBt5IMucxXPkX4rWi\n+z7wB3Rb\nBQoQzd8v7yeb7OzlPnWOyN0qFU0XA246RA8QFYiCNYwI3f05p6KLxEXAMPLE\n-----END CERTIFICATE-----\n", "ssl_cert": "-----BEGIN CERTIFICATE-----\nAAAAB3NzaC1yc2EAAAQABAAQClKsfkNkuSevGj3eYhCe53pcjqP3maAhDFcvBS706V\nhz2ItxCih+PnDSUaw+WNQn/mZphTk/a/gU8jEzoOWbkM4yxyb/wB96xbiFveSFJuOp/\nd6RJhJOI0iBXr\nlsLnBItnckiJ7FbtxJMLLvvwJryDUilBMTjYtwB+QhYXUMOzce5Pjz5/i8SeJtjnV3iAoG/\ncQk+0Fzz"}'); </pre>

Database engine	Instructions
	<pre>qaeJAAHco+CY/5WrUBkrHmFJr6HcXkvJdWPkYQS3xqC0+FmUzofz221CBt5IMucxXPkX4rWi +z7wB3Rb BQoQzd8v7yeb7OzlPnWOyN0qFU0XA246RA8QFYiCNYwI3f05p6KLxEXAMPLE ----END CERTIFICATE----\n", "ssl_key": "----BEGIN RSA PRIVATE KEY---- AAAAB3NzaC1yc2EAAAQABAAQClKsfkNkuSevGj3eYhCe53pcjqP3maAhDFcvBS706V hz2ItxCih+PnDSUaw+WNQn/mZphTk/a/gu8jEzoOWbkM4yxyb/wB96xbiFveSFJuOp/ d6RJhJOI0IBxr 1sLnBItnckij7FbtJMxLvvwJryDUilBMTjYtwB+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 \(p. 683\)](#).

**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
```

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
```

Database engine	Instructions
	<pre>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&gt; 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 <a href="#">Setting the replication source configuration</a> 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 you to restore your source database in the event of a failure.

Use the following instructions to retain binary logs for your database engine.

Database engine	Instructions
Aurora	<p><b>To retain binary logs on an Aurora MySQL DB cluster</b></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</p>

Database engine	Instructions
	<p>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 <a href="#">mysql_rds_set_configuration</a> procedure and specify a configuration parameter of 'binlog retention hours' along with the number of hours to retain binlog files on the DB cluster. The maximum value for Aurora MySQL version 1 is 2160 (90 days). The maximum value for Aurora MySQL version 2 and 3 is 168 (7 days).</p> <p>The following example sets the retention period for binlog files to 6 days:</p> <pre data-bbox="470 756 1302 783">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 the maximum value, then Aurora MySQL uses the maximum.</p>
RDS for MySQL	<p><b>To retain binary logs on an Amazon RDS DB instance</b></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 row.</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 <a href="#">mysql_rds_stop_replication</a> 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 <code>Seconds behind master</code> field (between your replication source and your permanent replica) reaches 0.</p>
MySQL (external)	<p><b>To retain binary logs on an external MySQL database</b></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 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, you can delete old binlog files.</p>

### 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.

Use the following instructions to create a snapshot of your replication source for your database engine.

Database engine	Instructions
Aurora	<p><b>To create a snapshot of an Aurora MySQL DB cluster</b></p> <ol style="list-style-type: none"> <li>1. Create a DB cluster snapshot of your Amazon Aurora DB cluster. For more information, see <a href="#">Creating a DB cluster snapshot (p. 373)</a>.</li> <li>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 <a href="#">Restoring from a DB cluster snapshot (p. 375)</a>.</li> <li>3. In the console, choose <b>Databases</b> and choose the primary instance (writer) for your restored Aurora DB cluster to show its details. Scroll to <b>Recent Events</b>. An event message shows that includes the binlog file name and position. The event message is in the following format.</li> </ol> <div style="border: 1px solid black; padding: 5px; background-color: #f9f9f9;"> <pre>Binlog position from crash recovery is binlog-file-name binlog-position</pre> </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 <b>describe-events</b> command from the AWS CLI. The following shows an example <b>describe-events</b> command with example output.</p> <div style="border: 1px solid black; padding: 5px; background-color: #f9f9f9;"> <pre>PROMPT&gt; aws rds describe-events</pre> </div> <div style="border: 1px solid black; padding: 5px; background-color: #f9f9f9;"> <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> <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> <ol style="list-style-type: none"> <li>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</li> </ol>

Database engine	Instructions
	<p>the copy of your Amazon Aurora DB cluster that you created. The following is an example.</p> <pre>PROMPT&gt; mysqldump --databases &lt;database_name&gt; --single-transaction --order-by-primary -r backup.sql -u &lt;local_user&gt; -p</pre> <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><b>To create a snapshot of an Amazon RDS DB instance</b></p> <ol style="list-style-type: none"> <li>1. Create a read replica of your Amazon RDS DB instance. For more information, see <a href="#">Creating a read replica</a> in the <i>Amazon Relational Database Service User Guide</i>.</li> <li>2. Connect to your read replica and stop replication by running the <a href="#">mysql_rds_stop_replication</a> procedure.</li> <li>3. While the read replica is <b>Stopped</b>, 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.</li> <li>4. While the read replica remains <b>Stopped</b>, create a DB snapshot of the read replica. For more information, see <a href="#">Creating a DB snapshot</a> in the <i>Amazon Relational Database Service User Guide</i>.</li> <li>5. Delete the read replica.</li> </ol>
MySQL (external)	<p><b>To create a snapshot of an external MySQL database</b></p> <ol style="list-style-type: none"> <li>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:</li> <pre>mysql&gt; FLUSH TABLES WITH READ LOCK;</pre> <li>2. Create a dump of your MySQL database using the <code>mysqldump</code> command as shown following:</li> <pre>PROMPT&gt; sudo mysqldump --databases &lt;database_name&gt; --master-data=2 --single-transaction \ --order-by-primary -r backup.sql -u &lt;local_user&gt; -p</pre> <li>3. After you have created the snapshot, unlock the tables in your MySQL database with the following command:</li> <pre>mysql&gt; UNLOCK TABLES;</pre> </ol>

#### 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.

Use the following instructions to load the snapshot of your replication source into your replica target for your database engine.

Database engine	Instructions
Aurora	<p><b>To load a snapshot into an Aurora MySQL DB cluster</b></p> <ul style="list-style-type: none"> <li>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 <a href="#">Restoring from a DB cluster snapshot (p. 375)</a>.</li> <li>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 <a href="#">Migrating data to an Amazon Aurora DB cluster (p. 242)</a>.</li> <li>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"> <li>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.</li> <li>Connect to your Aurora MySQL DB cluster using the <code>mysql</code> command. The following is an example.               <div style="border: 1px solid black; padding: 5px; margin-top: 10px;">                 PROMPT&gt; mysql -h &lt;host_name&gt; -port=3306 -u &lt;db_master_user&gt; -p               </div> </li> <li>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:               <div style="border: 1px solid black; padding: 5px; margin-top: 10px;">                 mysql&gt; source backup.sql;               </div> </li> </ol> </li> </ul>
RDS for MySQL	<p><b>To load a snapshot into an Amazon RDS DB instance</b></p> <ol style="list-style-type: none"> <li>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.</li> <li>Connect to your MySQL DB instance using the <code>mysql</code> command. The following is an example.               <div style="border: 1px solid black; padding: 5px; margin-top: 10px;">                 PROMPT&gt; mysql -h &lt;host_name&gt; -port=3306 -u &lt;db_master_user&gt; -p               </div> </li> <li>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:               <div style="border: 1px solid black; padding: 5px; margin-top: 10px;">                 mysql&gt; source backup.sql;               </div> </li> </ol>
MySQL (external)	<p><b>To load a snapshot into an external MySQL database</b></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> <ol style="list-style-type: none"> <li>Copy the output of the <code>mysqldump</code> command from your replication source to a location that can also connect to your MySQL database.</li> <li>Connect to your MySQL database using the <code>mysql</code> command. The following is an example.</li> </ol>

Database engine	Instructions
	<pre>PROMPT&gt; mysql -h &lt;host_name&gt; -port=3306 -u &lt;db_master_user&gt; -p</pre> <p>3. At the mysql prompt, run the source 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&gt; source backup.sql;</pre>

## 5. Create a replication user on your replication source

Create a user ID on the source 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.

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 REPLICATION CLIENT, REPLICATION SLAVE ON *.* TO 'repl_user'@'<domain_name>';
```

```
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.

## 6. 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.

Use the following instructions to turn on replication for your database engine.

Database engine	Instructions
Aurora MySQL	<p><b>To turn on replication from an Aurora MySQL DB cluster</b></p> <ol style="list-style-type: none"> <li>If your DB cluster replica target was created from a DB cluster snapshot, then connect to the DB cluster replica target and issue the SHOW MASTER STATUS command. Retrieve the current binary log file name from the File field and the log file position from the Position field.</li> </ol> <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 these values from the SHOW SLAVE STATUS (Aurora MySQL version 1 and 2) or SHOW</p>

Database engine	Instructions
	<p>REPLICA STATUS (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 <a href="#">mysql_rds_set_external_master</a> (Aurora MySQL version 1 and 2) <a href="#">mysql_rds_set_external_source</a> (Aurora MySQL version 3 and higher) and <a href="#">mysql_rds_start_replication</a> 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', '&lt;password&gt;', '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', '&lt;password&gt;', 'mysql-bin-changelog.000031', 107, 0);  For all versions: CALL mysql.rds_start_replication; </pre>
RDS for MySQL	<p>To use SSL encryption, set the final value to 1 instead of 0.</p> <p><b>To turn on replication from an Amazon RDS DB instance</b></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 SHOW SLAVE STATUS (Aurora MySQL version 1 and 2) or SHOW REPLICA STATUS (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 <a href="#">mysql_rds_set_external_master</a> and <a href="#">mysql_rds_start_replication</a> 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', '&lt;password&gt;', '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', '&lt;password&gt;', 'mysql-bin-changelog.000031', 107, 0);  For all versions: CALL mysql.rds_start_replication; </pre> <p>To use SSL encryption, set the final value to 1 instead of 0.</p>

Database engine	Instructions
MySQL (external)	<p><b>To turn on replication from an external MySQL database</b></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>mysqlldump</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 REPLICAS</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 = '&lt;password&gt;',   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 REPLICAS; -- 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.

## 7. 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 [Working with Amazon RDS event notification \(p. 572\)](#).

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. 853)

#### 2. Turn off binary logging on the replication source (p. 853)

### 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	<b>To stop binary log replication on an Aurora MySQL DB cluster replica target</b>  Connect to the Aurora DB cluster that is the replica target, and call the <a href="#">mysql_rds_stop_replication</a> procedure.
RDS for MySQL	<b>To stop binary log replication on an Amazon RDS DB instance</b>  Connect to the RDS DB instance that is the replica target and call the <a href="#">mysql_rds_stop_replication</a> 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)	<b>To stop binary log replication on an external MySQL database</b>  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	<b>To turn off binary logging on an Amazon Aurora DB cluster</b>  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 <a href="#">mysql_rds_set_configuration</a> 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 <a href="#">Amazon Aurora DB cluster and DB instance parameters (p. 217)</a> and <a href="#">Modifying parameters in a DB parameter group (p. 231)</a> .
RDS for MySQL	<p><b>To turn off binary logging on an Amazon RDS DB instance</b></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"> <li>1. Turn off automated backups for the DB instance. You can turn off automated backups by modifying an existing DB instance and setting the <b>Backup Retention Period</b> to 0. For more information, see <a href="#">Modifying an Amazon RDS DB instance</a> and <a href="#">Working with backups</a> in the <i>Amazon Relational Database Service User Guide</i>.</li> <li>2. Delete all read replicas for the DB instance. For more information, see <a href="#">Working with read replicas of MariaDB, MySQL, and PostgreSQL DB instances</a> in the <i>Amazon Relational Database Service User Guide</i>.</li> </ol>
MySQL (external)	<p><b>To turn off binary logging on an external MySQL database</b></p> <p>Connect to the MySQL database and call the <code>STOP REPLICATION</code> command.</p> <ol style="list-style-type: none"> <li>1. From a command shell, stop the <code>mysqld</code> service,</li> <div style="border: 1px solid black; padding: 5px; margin-top: 10px;"> <pre>sudo service mysqld stop</pre> </div> <li>2. Edit the <code>my.cnf</code> file (this file is usually under <code>/etc</code>).</li> <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 <a href="#">Setting the replication source configuration</a> in the MySQL documentation.</p> <li>3. Start the <code>mysql</code> service.</li> <div style="border: 1px solid black; padding: 5px; margin-top: 10px;"> <pre>sudo service mysqld start</pre> </div> </ol>

## 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 \(p. 127\)](#).

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 \(p. 1766\)](#).

**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 \(p. 270\)](#).

## 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 \(p. 859\)](#)
- [Related parameters \(p. 860\)](#)
- [Enabling the max-yield mechanism for binary log replication \(p. 861\)](#)
- [Turning off the binary log replication max-yield optimization \(p. 862\)](#)
- [Turning off the large read buffer \(p. 862\)](#)

## 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>	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		For Aurora MySQL versions 2.04.5–2.04.8 and 2.05–2.08.*, the maximum accepted

Parameter	Default	Valid Values	Description
			value is 45. You can tune it to a higher 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>	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 parameter groups \(p. 215\)](#).
  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 |  
+-----+  
+-----+  
| 1 |  
+-----+  
+-----+
```

## 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 parameter groups \(p. 215\)](#).
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 parameter groups \(p. 215\)](#).

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 Amazon 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 use this feature only 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 \(binary log replication\) \(p. 841\)](#).

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 \(GTIDs\) \(p. 864\)](#)
- [Parameters for GTID-based replication \(p. 864\)](#)
- [Configuring GTID-based replication for an Aurora MySQL cluster \(p. 865\)](#)
- [Disabling GTID-based replication for an Aurora MySQL DB cluster \(p. 865\)](#)

## 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	<p>OFF specifies that new transactions are anonymous transactions (that is, don't have GTIDs), and a transaction must be anonymous to be replicated.</p> <p>OFF_PERMISSIVE specifies that new transactions are anonymous transactions, but all transactions can be replicated.</p> <p>ON_PERMISSIVE specifies that new transactions are GTID transactions, but all transactions can be replicated.</p> <p>ON specifies that new transactions are GTID transactions, and a transaction must be a GTID transaction to be replicated.</p>
enforce_gtid_consistency	OFF, ON	<p>OFF allows transactions to violate GTID consistency.</p> <p>ON prevents transactions from violating GTID consistency.</p>

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 Aurora MySQL cluster. Both of these values cause your 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 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 Aurora DB cluster only accepts 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 parameter groups \(p. 215\)](#).

## 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 parameter groups \(p. 215\)](#).
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 \(Aurora MySQL version 3 and higher\) \(p. 984\)](#).

## 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 \(p. 984\)](#).

**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 parameter groups \(p. 215\)](#).

- 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

### Important

Amazon has announced an end-of-life policy for Amazon Aurora MySQL-Compatible Edition v1. For details about how long Aurora MySQL version 1 will remain available and how to migrate to a higher Aurora MySQL version, see [Preparing for Amazon Aurora MySQL-Compatible Edition version 1 end of life \(p. 994\)](#).

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 \(p. 867\)](#)
- [Creating an Aurora multi-master cluster \(p. 873\)](#)
- [Managing Aurora multi-master clusters \(p. 875\)](#)
- [Application considerations for Aurora multi-master clusters \(p. 878\)](#)
- [Performance considerations for Aurora multi-master clusters \(p. 887\)](#)
- [Approaches to Aurora multi-master clusters \(p. 889\)](#)

## 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 \(p. 868\)](#)
- [Multi-master cluster architecture \(p. 869\)](#)
- [Recommended workloads for multi-master clusters \(p. 870\)](#)
- [Advantages of multi-master clusters \(p. 871\)](#)
- [Limitations of multi-master clusters \(p. 871\)](#)
- [Migrating from multi-master clusters \(p. 872\)](#)

## 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 \(p. 888\)](#).

### 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 \(p. 880\)](#).

## 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 \(p. 887\)](#).

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 \(p. 877\)](#).

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 \(p. 888\)](#).

## 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 \(p. 889\)](#).

## 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 \(p. 889\)](#).
- Aurora detects write conflicts immediately, not when the transaction commits. For details about the write conflict mechanism, see [Conflict resolution for multi-master clusters \(p. 888\)](#).

## Limitations of multi-master clusters

### Note

Aurora multi-master clusters are only available for Amazon Aurora MySQL-Compatible Edition v1. Amazon has announced an end-of-life policy for this major Aurora MySQL version.

For details about how long Aurora MySQL version 1 will remain available and how to migrate to a higher Aurora MySQL version, see [Preparing for Amazon Aurora MySQL-Compatible Edition version 1 end of life \(p. 994\)](#).

### 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 \(p. 879\)](#).

- 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.
- 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.
- When you restore a snapshot created on a multi-master cluster, make sure to include the `--engine-mode multimaster` option. If you don't use this option, you will receive an error.
- 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 \(p. 878\)](#).

## Migrating from multi-master clusters

Migration from an Aurora multi-master cluster means changing back to an Aurora single-master DB cluster. Use a logical dump produced by a tool such as AWS Database Migration Service (AWS DMS) or the `mysqldump` command.

# 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 \(p. 127\)](#).

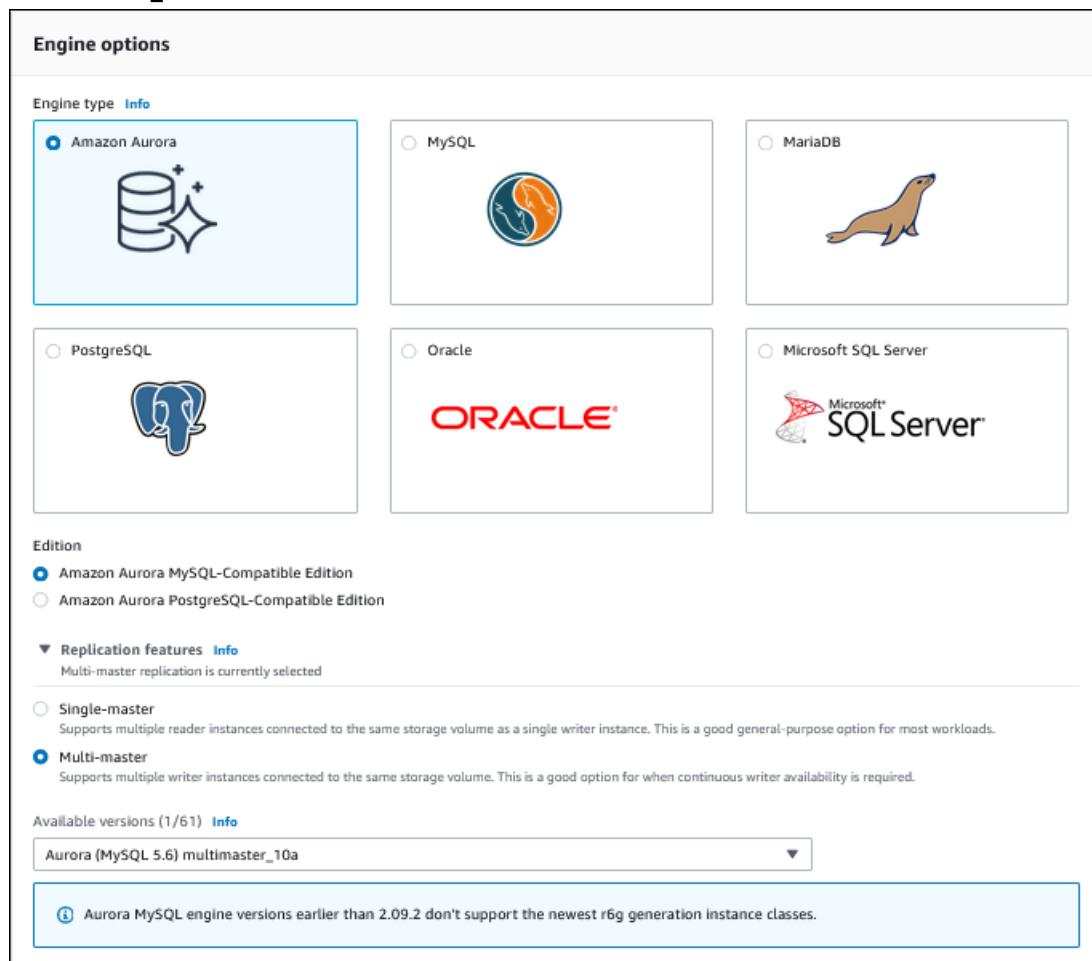
## Console

Use the following procedure.

### To create an Aurora multi-master cluster

1. For **Engine type**, choose **Amazon Aurora**.
2. For **Edition**, choose **Amazon Aurora MySQL-Compatible Edition**.
3. Expand **Replication features**, then choose **Multi-master**.

For **Available versions**, the only version supported for multi-master clusters is **Aurora (MySQL 5.6) multimaster\_10a**.



4. 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 \(p. 131\)](#).
5. Choose **Create database**.

The multi-master cluster is created with one writer instance.

6. After you create the multi-master cluster, add a second writer instance to it by using the following procedure:
  - a. On the **Databases** page, choose the multi-master cluster.
  - b. For **Actions**, choose **Add DB instance**.
  - c. Enter the DB instance identifier.
  - d. Make other choices as needed, then choose **Add DB instance**.

**DB instance specifications**

**DB engine**  
Name of the database engine to be used for this instance.

**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.

The DB instance 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.

**Availability Zone**

(i) After the DB instance is created, you can't change the availability zone selection.

**Instance configuration**  
The DB instance configuration options below are limited to those supported by the engine that you selected above.

**DB instance class** Info  
 Memory optimized classes (includes r classes)  
 Include previous generation classes

8 vCPUs 61 GiB RAM Network: 1,700 Mbps

**Add DB instance**

After you create the multi-master cluster and associated DB instances, you see the cluster on the **Databases** page as follows. All DB instances show the role **Writer instance**.

<input type="radio"/>	<input type="checkbox"/> db-multi-master	Regional cluster	Aurora MySQL	5.6.10a
<input type="radio"/>	<input type="checkbox"/> db-multi-master-instance-1	Writer instance	Aurora MySQL	5.6.10a
<input type="radio"/>	<input type="checkbox"/> db-multi-master-instance-2	Writer instance	Aurora MySQL	5.6.10a

## AWS CLI

To create a multi-master cluster with the AWS CLI, run the [create-db-cluster](#) AWS CLI command and include the `--engine-mode` option with the value `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 \(p. 131\)](#).

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 \(p. 270\)](#).

## 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 \(p. 131\)](#).

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 \(p. 270\)](#).

## 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. The difference for multi-master clusters is that the new DB instances all have read/write capability instead of being read-only Aurora Replicas.

## 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 \(p. 875\)](#)
- [Data ingestion performance for multi-master clusters \(p. 876\)](#)
- [Exporting data from a multi-master cluster \(p. 877\)](#)
- [High availability considerations for Aurora multi-master clusters \(p. 877\)](#)
- [Replication between multi-master clusters and other clusters \(p. 877\)](#)
- [Upgrading a multi-master cluster \(p. 877\)](#)

## 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 \(p. 890\)](#).

## 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 \(p. 878\)](#)
- [Connection management for multi-master clusters \(p. 879\)](#)
- [Consistency model for multi-master clusters \(p. 880\)](#)
- [Multi-master clusters and transactions \(p. 881\)](#)
- [Write conflicts and deadlocks in multi-master clusters \(p. 881\)](#)
- [Multi-master clusters and locking reads \(p. 882\)](#)
- [Performing DDL operations on a multi-master cluster \(p. 883\)](#)
- [Using autoincrement columns \(p. 884\)](#)
- [Multi-master clusters feature reference \(p. 885\)](#)

## 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 \(p. 883\)](#).
- 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 \(p. 884\)](#).

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 \(p. 883\)](#).
- 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 \(p. 884\)](#).
- 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 \(p. 886\)](#).

### 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 \(p. 880\)](#).

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 \(p. 887\)](#).

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 \(p. 880\)](#).
- 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 \(p. 886\)](#).

## 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 \(p. 875\)](#).

### Topics

- [Query performance for multi-master clusters \(p. 888\)](#)
- [Conflict resolution for multi-master clusters \(p. 888\)](#)
- [Optimizing buffer pool and dictionary cache usage \(p. 888\)](#)

## 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 \(p. 887\)](#).

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 \(p. 888\)](#).
- If you need read/write workload isolation, designate one or more DB instances as read-only, as described in [Using instance read-only mode \(p. 887\)](#). 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 \(p. 889\)](#)
- [Using a multi-master cluster without sharding \(p. 890\)](#)
- [Using a multi-master cluster as an active standby \(p. 890\)](#)

## 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 \(p. 917\)](#).
- 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 \(p. 903\)](#).
- 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 \(p. 911\)](#).
- Automatically add or remove Aurora Replicas with Application Auto Scaling. For more information, see [Using Amazon Aurora Auto Scaling with Aurora replicas \(p. 305\)](#).
- 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 \(p. 320\)](#).

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 \(p. 891\)](#).

## 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 \(p. 990\)](#).

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 \(p. 892\)](#).

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 \(p. 902\)](#).

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 \(p. 918\)](#).
- 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 \(p. 903\)](#).
- 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 \(p. 911\)](#).
- Export log data to Amazon CloudWatch Logs MySQL. For more information, see [Publishing Amazon Aurora MySQL logs to Amazon CloudWatch Logs \(p. 924\)](#).

- Automatically add or remove Aurora Replicas with Application Auto Scaling. For more information, see [Using Amazon Aurora Auto Scaling with Aurora replicas \(p. 305\)](#).

## 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 \(p. 892\)](#)
  - [Creating an IAM policy to access AWS Lambda resources \(p. 894\)](#)
  - [Creating an IAM policy to access CloudWatch Logs resources \(p. 895\)](#)
  - [Creating an IAM policy to access AWS KMS resources \(p. 896\)](#)
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 \(p. 897\)](#).
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 \(p. 898\)](#).

### 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:GetObject",  
            "Resource": "arn:aws:s3:::examplebucket/*"  
        }  
    ]  
}
```

```
        "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 \(p. 897\)](#).

## 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 \(p. 897\)](#).

## 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 \(p. 897\)](#).

### 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 \(p. 897\)](#).

### 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**.

7. On the **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 \(p. 892\)](#)
  - [Creating an IAM policy to access AWS Lambda resources \(p. 894\)](#)
  - [Creating an IAM policy to access CloudWatch Logs resources \(p. 895\)](#)
  - [Creating an IAM policy to access AWS KMS resources \(p. 896\)](#)
9. Choose **Next**.
10. In **Role name**, enter a name for your IAM role, for example `RDSLoadFromS3`. You can also add an optional **Description** value.
11. Choose **Create Role**.
12. Complete the steps in [Associating an IAM role with an Amazon Aurora MySQL DB cluster \(p. 898\)](#).

## 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 \(p. 897\)](#) with that DB cluster.

### Note

You can't associate an IAM role with an Aurora Serverless v1 DB cluster. For more information, see [Using Amazon Aurora Serverless v1 \(p. 1543\)](#).

You can associate an IAM role with an Aurora Serverless v2 DB cluster.

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.

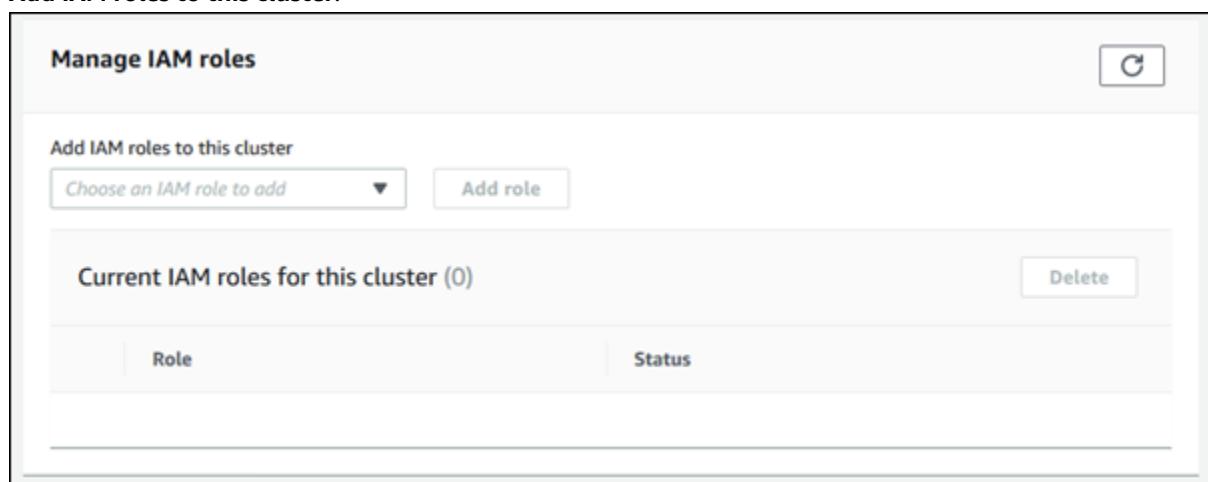
Cluster-level parameter	Description
<code>aws_default_lambda_role</code>	Used when invoking a Lambda function from your DB cluster.
<code>aws_default_logs_role</code>	This parameter is no longer required for exporting log data from your DB cluster to Amazon CloudWatch Logs. Aurora MySQL now uses a service-linked role for the required permissions. For more information about service-linked roles, see <a href="#">Using service-linked roles for Amazon Aurora (p. 1724)</a> .
<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.  The IAM role specified in this parameter is used if an IAM role isn't specified

Cluster-level parameter	Description	
	<p>for <code>aurora_load_from_s3_role</code> or <code>aurora_select_into_s3_role</code> for the appropriate statement.</p> <p>For Aurora MySQL version 3, the IAM role specified for this parameter is always used.</p>	
<code>aurora_load_from_s3</code>	<p><del>Used</del> when invoking the <code>LOAD DATA FROM S3</code> or <code>LOAD XML FROM S3</code> statement from your DB cluster. If an IAM role is not specified for this parameter, the IAM role specified in <code>aws_default_s3_role</code> is used.</p> <p>For Aurora MySQL version 3, this parameter isn't available.</p>	
<code>aurora_select_into_s3</code>	<p><del>Used</del> when invoking the <code>SELECT INTO OUTFILE S3</code> statement from your DB cluster. If an IAM role is not specified for this parameter, the IAM role specified in <code>aws_default_s3_role</code> is used.</p> <p>For Aurora MySQL version 3, this parameter isn't 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. It has a title 'Create parameter group' at the top. Below it is a section titled 'Parameter group details' with the sub-instruction: 'To create a parameter group, select a parameter group family, then name and describe your parameter group'. There are four input fields: 'Parameter group family' (dropdown menu showing 'aurora5.6'), 'Type' (dropdown menu showing 'DB Cluster Parameter Group'), 'Group name' (text input field containing 'AllowAWSAccess'), and 'Description' (text input field containing 'Allow access to S3'). At the bottom right are 'Cancel' and 'Create' buttons.

- 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 \(p. 949\)](#).

### 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 \(p. 949\)](#).

## 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 \(p. 918\)](#).
- 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 \(p. 903\)](#) and [Saving data from an Amazon Aurora MySQL DB cluster into text files in an Amazon S3 bucket \(p. 911\)](#).
- 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 \(p. 619\)](#).
- 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 \(p. 927\)](#).

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 cluster in a VPC \(p. 1729\)](#). 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](#).

You might also have to open the ephemeral ports for your network access control lists (ACLs) in the outbound rules for your VPC security group. For more information on ephemeral ports for network ACLs, see [Ephemeral ports](#) in the *Amazon Virtual Private Cloud User Guide*.

## Related topics

- [Integrating Aurora with other AWS services \(p. 304\)](#)
- [Managing an Amazon Aurora DB cluster \(p. 243\)](#)

# 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 \(p. 990\)](#).

This feature isn't supported for Aurora Serverless v1. It is supported for Aurora Serverless v2.

## 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 \(p. 892\)](#).
2. Create an IAM role, and attach the IAM policy you created in [Creating an IAM policy to access Amazon S3 resources \(p. 892\)](#) to the new IAM role. For instructions, see [Creating an IAM role to allow Amazon Aurora to access AWS services \(p. 897\)](#).
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 \(p. 219\)](#).

4. For Aurora MySQL version 1 or 2, 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`.

For Aurora MySQL version 3, use `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 \(p. 217\)](#).

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 \(p. 897\)](#) 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 \(p. 898\)](#).
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 \(p. 902\)](#).

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 \(p. 664\)](#). You can also find more details in [Using Roles in the MySQL Reference Manual](#).

This only applies to the current active session. When you reconnect, you have to run the `SET ROLE` statement again to grant privileges. For more information, see [SET ROLE statement](#) in the [MySQL Reference Manual](#).

You can also use the `activate_all_roles_on_login` DB cluster parameter to automatically activate all roles when a user connects to a DB instance. When this parameter is set, you don't have to call the `SET ROLE` statement explicitly to activate a role. For more information, see [activate\\_all\\_roles\\_on\\_login](#) 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 \(p. 906\)](#).

## 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 \(p. 904\)](#).
- **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"> <li>• A single data file for a <code>LOAD DATA FROM S3 FILE</code> statement</li> <li>• An Amazon S3 prefix that maps to multiple data files for a <code>LOAD DATA FROM S3 PREFIX</code> statement</li> <li>• A single manifest file that contains the names of files to be loaded for a <code>LOAD DATA FROM S3 MANIFEST</code> statement</li> </ul>
<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 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. `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 \(p. 890\)](#)
- [Saving data from an Amazon Aurora MySQL DB cluster into text files in an Amazon S3 bucket \(p. 911\)](#)
- [Managing an Amazon Aurora DB cluster \(p. 243\)](#)
- [Migrating data to an Amazon Aurora DB cluster \(p. 242\)](#)

# 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 \(p. 903\)](#).

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 cluster snapshot data to Amazon S3 \(p. 396\)](#).

## 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 \(p. 892\)](#).
2. Create an IAM role, and attach the IAM policy you created in [Creating an IAM policy to access Amazon S3 resources \(p. 892\)](#) to the new IAM role. For instructions, see [Creating an IAM role to allow Amazon Aurora to access AWS services \(p. 897\)](#).
3. For Aurora MySQL version 1 or 2, 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`.

For Aurora MySQL version 3, use `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 \(p. 217\)](#).

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 \(p. 897\)](#) 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 \(p. 898\)](#).

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 \(p. 902\)](#).

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 \(p. 664\)](#). You can also find more details in [Using Roles in the MySQL Reference Manual](#).

This only applies to the current active session. When you reconnect, you have to run the `SET ROLE` statement again to grant privileges. For more information, see [SET ROLE statement](#) in the [MySQL Reference Manual](#).

You can also use the `activate_all_roles_on_login` DB cluster parameter to automatically activate all roles when a user connects to a DB instance. When this parameter is set, you don't have to call the `SET ROLE` statement explicitly to activate a role. For more information, see [activate\\_all\\_roles\\_on\\_login](#) 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 \(p. 906\)](#).

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"
    }
  ]
}
```

```

        "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}]]
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 \(p. 913\)](#).
- **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 \(p. 933\)](#).

- **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 \(p. 903\)](#).

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 \(p. 913\)](#).

- **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 \(p. 304\)](#)
- [Loading data into an Amazon Aurora MySQL DB cluster from text files in an Amazon S3 bucket \(p. 903\)](#)
- [Managing an Amazon Aurora DB cluster \(p. 243\)](#)
- [Migrating data to an Amazon Aurora DB cluster \(p. 242\)](#)

## 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 \(p. 917\)](#). For information about the `lambda_sync` and `lambda_async` stored functions, see [Invoking a Lambda function with an Aurora MySQL native function \(p. 918\)](#).

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 \(p. 917\)](#)
- [Invoking a Lambda function with an Aurora MySQL native function \(p. 918\)](#)
- [Invoking a Lambda function with an Aurora MySQL stored procedure \(deprecated\) \(p. 921\)](#)

## 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 \(p. 894\)](#).
2. Create an IAM role, and attach the IAM policy you created in [Creating an IAM policy to access AWS Lambda resources \(p. 894\)](#) to the new IAM role. For instructions, see [Creating an IAM role to allow Amazon Aurora to access AWS services \(p. 897\)](#).
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 \(p. 217\)](#).

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 \(p. 897\)](#) 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 \(p. 898\)](#).

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 \(p. 902\)](#).

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 \(p. 990\)](#).

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 \(p. 664\)](#). You can also find more details in [Using Roles in the MySQL Reference Manual](#).

This only applies to the current active session. When you reconnect, you have to run the `SET ROLE` statement again to grant privileges. For more information, see [SET ROLE statement](#) in the [MySQL Reference Manual](#).

You can also use the `activate_all_roles_on_login` DB cluster parameter to automatically activate all roles when a user connects to a DB instance. When this parameter is set, you don't have to call the `SET ROLE` statement explicitly to activate a role. For more information, see [activate\\_all\\_roles\\_on\\_login](#) 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 \(p. 918\)](#).

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 \(p. 823\)](#).

#### 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 \(p. 892\)](#). 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 \(p. 1724\)](#).

- 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 v1 clusters than for clusters with provisioned or Aurora Serverless v2 instances. Aurora Serverless v1 clusters automatically upload all the kinds of logs that you enable through the configuration parameters. Therefore, you turn on or turn off 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 turning on and off MySQL logs for Aurora Serverless v1 clusters, see [Parameter groups for Aurora Serverless v1 \(p. 1554\)](#).

## 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 \(p. 936\)](#). 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 \(p. 934\)](#). For general information about SageMaker, see [SageMaker](#).

#### Topics

- [Prerequisites for Aurora machine learning \(p. 928\)](#)
- [Enabling Aurora machine learning \(p. 928\)](#)
- [Exporting data to Amazon S3 for SageMaker model training \(p. 933\)](#)
- [Using SageMaker to run your own ML models \(p. 934\)](#)
- [Using Amazon Comprehend for sentiment detection \(p. 936\)](#)
- [Performance considerations for Aurora machine learning \(p. 937\)](#)
- [Monitoring Aurora machine learning \(p. 938\)](#)
- [Limitations of Aurora machine learning \(p. 939\)](#)

## Prerequisites for 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 \(p. 990\)](#).

For more information about Regions and Aurora version availability, see [Aurora machine learning \(p. 24\)](#).

## 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 \(p. 928\)](#)
- [Granting SQL privileges for invoking Aurora machine learning services \(p. 933\)](#)
- [Enabling network communication from Aurora MySQL to other AWS services \(p. 933\)](#)

## 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 \(p. 929\)](#).

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 \(p. 891\)](#).

#### [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 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 \(p. 897\)](#).

### 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 \(p. 897\)](#). 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 \(p. 664\)](#). You can also find more details in [Using Roles in the MySQL Reference Manual](#).

This only applies to the current active session. When you reconnect, you have to run the `SET ROLE` statement again to grant privileges. For more information, see [SET ROLE statement](#) in the [MySQL Reference Manual](#).

You can also use the `activate_all_roles_on_login` DB cluster parameter to automatically activate all roles when a user connects to a DB instance. When this parameter is set, you don't have to call the `SET ROLE` statement explicitly to activate a role. For more information, see [activate\\_all\\_roles\\_on\\_login](#) in the [MySQL Reference Manual](#).

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 \(p. 902\)](#).

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 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 \(p. 937\)](#).

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 \(p. 940\)](#).

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. 231). For information on parameter groups and Amazon Aurora, see [Aurora MySQL configuration parameters \(p. 949\)](#).

## 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 <a href="#">Optimizing large Aurora MySQL join queries with hash joins (p. 945)</a> .
Fast DDL	This feature allows you to run an <code>ALTER TABLE <i>tbl_name</i> ADD COLUMN <i>col_name 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 <a href="#">Altering tables in Amazon Aurora using fast DDL (p. 741)</a> .

## 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 \(p. 243\)](#).

### Contents

- [Determining which DB instance you are connected to \(p. 941\)](#)
- [Best practices for Aurora MySQL performance and scaling \(p. 941\)](#)
  - [Using T instance classes for development and testing \(p. 941\)](#)
  - [Optimizing Amazon Aurora indexed join queries with asynchronous key prefetch \(p. 943\)](#)
    - [Enabling asynchronous key prefetch \(p. 943\)](#)
    - [Optimizing queries for asynchronous key prefetch \(p. 944\)](#)

- Optimizing large Aurora MySQL join queries with hash joins (p. 945)
  - Enabling hash joins (p. 945)
  - Optimizing queries for hash joins (p. 946)
- Using Amazon Aurora to scale reads for your MySQL database (p. 947)
- Best practices for Aurora MySQL high availability (p. 947)
  - Using Amazon Aurora for Disaster Recovery with your MySQL databases (p. 947)
  - Migrating from MySQL to Amazon Aurora MySQL with reduced downtime (p. 948)
- Recommendations for MySQL features (p. 948)
  - Using multithreaded replication in Aurora MySQL version 3 (p. 948)
  - Invoking AWS Lambda functions using native MySQL functions (p. 948)
  - Avoiding XA transactions with Amazon Aurora MySQL (p. 949)
  - Keeping foreign keys turned on during DML statements (p. 949)

## 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 Aurora MySQL performance and scaling

You can apply the following best practices to improve the performance and scalability of your Aurora MySQL clusters.

### Topics

- Using T instance classes for development and testing (p. 941)
- Optimizing Amazon Aurora indexed join queries with asynchronous key prefetch (p. 943)
- Optimizing large Aurora MySQL join queries with hash joins (p. 945)
- Using Amazon Aurora to scale reads for your MySQL database (p. 947)

## 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.

**Tip**

If your database is sometimes idle but at other times has a substantial workload, you can use Aurora Serverless v2 as an alternative to T instances. With Aurora Serverless v2, you define a capacity range and Aurora automatically scales your database up or down depending on the current workload. For usage details, see [Using Aurora Serverless v2 \(p. 1482\)](#). For the database engine versions that you can use with Aurora Serverless v2, see [Requirements for Aurora Serverless v2 \(p. 1490\)](#).

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 \(p. 449\)](#).

- 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).

## 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 \(p. 990\)](#).

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 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 the use of `EXPLAIN` to view the execution plan for a query that can take advantage of AKP.

```
mysql> explain 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
|           |             |           |         |          |             | |
|           |             |           |         |          |             |
| 1 | PRIMARY    | supplier   | ref    | PRIMARY,i_s_nationkey | i_s_nationkey | 5       |
| dbt3_scale_10.supplier.s_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
|           |             |           |         |          |             |
|           |             |           |         |          |             |
```

```

| 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 `EXPLAIN` output format, see [Extended EXPLAIN output format](#) in the MySQL product documentation.

**Note**

For MySQL 5.6-compatible and older 5.7-compatible versions, you use the `EXPLAIN` statement with the `EXTENDED` keyword. This keyword has been deprecated in MySQL 5.7 and 8.0.

For more information about the extended `EXPLAIN` output format in MySQL 5.6, 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**

You can't compare data types in different categories.

The following restrictions apply to hash joins for Aurora MySQL:

- Left-right outer joins aren't supported for Aurora MySQL versions 1 and 2, but are supported for version 3.
- Semijoins such as subqueries aren't supported, unless the subqueries are materialized first.
- Multiple-table updates or deletes aren't supported.

**Note**

Single-table updates or deletes are supported.

- BLOB and spatial data type columns can't 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';
```

**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`.

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**

This setting overrides the decisions of the cost-based optimizer. While the setting can be useful for testing and development, we recommend that you not use it in production.

## 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 \(p. 982\)](#).

## 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 \(p. 854\)](#).

## 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 \(p. 947\)](#)
- [Migrating from MySQL to Amazon Aurora MySQL with reduced downtime \(p. 948\)](#)

## 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 \(p. 947\)](#).

For more information on disaster recovery models, see [How to choose the best disaster recovery option for your Amazon Aurora MySQL cluster](#).

## 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.

## Recommendations for MySQL features

The following features are available in Aurora MySQL for MySQL compatibility. However, they have performance, scalability, stability, or compatibility issues in the Aurora environment. Thus, we recommend that you follow certain guidelines in 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 \(p. 948\)](#)
- [Invoking AWS Lambda functions using native MySQL functions \(p. 948\)](#)
- [Avoiding XA transactions with Amazon Aurora MySQL \(p. 949\)](#)
- [Keeping foreign keys turned on during DML statements \(p. 949\)](#)

## 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 \(p. 73\)](#). For information about multithreaded replication in Aurora MySQL version 3, see [Multithreaded binary log replication \(Aurora MySQL version 3 and higher\) \(p. 857\)](#).

## Invoking AWS Lambda functions using native MySQL functions

If you are using Amazon Aurora version 1.16 or later, we recommend using the native MySQL 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 \(p. 917\)](#).

## 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 \(p. 949\)](#)
- [MySQL parameters that don't apply to Aurora MySQL \(p. 969\)](#)
- [MySQL status variables that don't apply to Aurora MySQL \(p. 970\)](#)
- [Aurora MySQL wait events \(p. 971\)](#)
- [Aurora MySQL thread states \(p. 975\)](#)
- [Aurora MySQL isolation levels \(p. 978\)](#)
- [Aurora MySQL hints \(p. 982\)](#)
- [Aurora MySQL stored procedures \(p. 984\)](#)

## 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. 215) 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 parameter groups](#) (p. 215). For rules and restrictions for Aurora Serverless clusters, see [Parameter groups for Aurora Serverless v1](#) (p. 1554).

**Topics**

- [Cluster-level parameters](#) (p. 950)
- [Instance-level parameters](#) (p. 956)

## 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 <a href="#">Replication between Aurora and MySQL or between Aurora and another Aurora DB cluster (binary log replication)</a> (p. 841). 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 <a href="#">Replication between Aurora and MySQL or between Aurora and another Aurora DB cluster (binary log replication)</a> (p. 841).

Parameter name	Modifiable	Notes
aurora_binlog_use_large_read_buffer	Yes	Only affects clusters that use binary log (binlog) replication. For information about binlog replication, see <a href="#">Replication between Aurora and MySQL or between Aurora and another Aurora DB cluster (binary log replication) (p. 841)</a> . Removed from Aurora MySQL version 3.
aurora_enable_replica_log_compression	Yes	For more information, see <a href="#">Performance considerations for Amazon Aurora MySQL replication (p. 828)</a> . 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 <a href="#">Performance considerations for Amazon Aurora MySQL replication (p. 828)</a> . Doesn't apply to clusters that are part of an Aurora global database. Removed from Aurora MySQL version 3.
aurora_enable_staggered_replica_restart	Yes	This setting is available in Aurora MySQL versions 1 and 3, but it isn't used.
aurora_enable_zdr	Yes	This setting is turned on by default in Aurora MySQL 2.10 and higher. For more information, see <a href="#">Zero-downtime restart (ZDR) for Amazon Aurora MySQL (p. 829)</a> .
aurora_load_from_s3_role	Yes	For more information, see <a href="#">Loading data into an Amazon Aurora MySQL DB cluster from text files in an Amazon S3 bucket (p. 903)</a> . Currently not available in Aurora MySQL version 3. Use <code>aws_default_s3_role</code> .
aurora_select_into_s3_role	Yes	For more information, see <a href="#">Saving data from an Amazon Aurora MySQL DB cluster into text files in an Amazon S3 bucket (p. 911)</a> . Currently not available in Aurora MySQL version 3. Use <code>aws_default_s3_role</code> .
auto_increment_increment	Yes	
auto_increment_offset	Yes	
aws_default_lambda_role	Yes	For more information, see <a href="#">Invoking a Lambda function from an Amazon Aurora MySQL DB cluster (p. 917)</a> .
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 <a href="#">Aurora MySQL database engine updates 2020-09-02 (version 1.23.0)</a> and <a href="#">Aurora MySQL database engine updates 2020-03-05 (version 1.22.2)</a> in the <i>Release Notes for Aurora MySQL</i> .
binlog-do-db	Yes	This parameter applies to Aurora MySQL version 3 and higher.
binlog_format	Yes	For more information, see <a href="#">Replication between Aurora and MySQL or between Aurora and another Aurora DB cluster (binary log replication) (p. 841)</a> .
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_history_size_Yes_size	Yes	This parameter applies to Aurora MySQL version 3 and higher.
binlog_transaction_dependency_tracking_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	

Parameter name	Modifiable	Notes
character_set_database	Yes	
character_set_filesystem	Yes	
character_set_results	Yes	
character_set_server	Yes	
collation_connection	Yes	
collation_server	Yes	
completion_type	Yes	
default_storage_engine	No	Aurora MySQL clusters use the InnoDB storage engine for all of your data.
enforce_gtid_consistency	Sometimes	Modifiable in Aurora MySQL version 2.04 and later.
event_scheduler	Yes	Indicates the status of the Event Scheduler.  Modifiable only at the cluster level in Aurora MySQL version 3.
gtid-mode	Sometimes	Modifiable in Aurora MySQL version 2.04 and later.
innodb_autoinc_lock_mode	Yes	
innodb_checksums	No	Removed from Aurora MySQL version 3.
innodb_cmp_per_index_enabled	Yes	
innodb_commit_concurrency	Yes	
innodb_data_home_dir	No	Aurora MySQL uses managed instances where you don't access the file system directly.
innodb_deadlock_detect	Yes	This option is used to disable deadlock detection on Aurora MySQL version 3.  On high-concurrency systems, deadlock detection can cause a slowdown when numerous threads wait for the same lock. Consult the MySQL documentation for more information on this parameter.
innodb_file_per_table	Yes	
innodb_flush_log_at_trx_commit	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.

Parameter name	Modifiable	Notes
innodb_ft_max_token_size	Yes	
innodb_ft_min_token_size	Yes	
innodb_ft_num_word_optimize	Yes	
innodb_ft_sort_pll_degree	Yes	
innodb_online_alter_log_max_size	Yes	
innodb_optimize_fulltext_only	Yes	
innodb_page_size	No	
innodb_purge_batch_size	Yes	
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	

Parameter name	Modifiable	Notes
<code>lower_case_table_names</code>	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 <a href="#">Rebooting an Aurora MySQL cluster (version 2.10 and higher) (p. 330)</a> . 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.
<code>master-info-repository</code>	Yes	Removed from Aurora MySQL version 3.
<code>master_verify_checksum</code>	Yes	Aurora MySQL version 1 and 2. Use <code>source_verify_checksum</code> in Aurora MySQL version 3.
<code>partial_revokes</code>	No	This parameter applies to Aurora MySQL version 3 and higher.
<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 <a href="#">Using SSL/TLS with Aurora MySQL DB clusters (p. 683)</a> .
<code>rpl_read_size</code>	Yes	This parameter applies to Aurora MySQL version 3 and higher.

Parameter name	Modifiable	Notes
server_audit_events	Yes	
server_audit_excl_users	Yes	
server_audit_incl_users	Yes	
server_audit_logging	Yes	For instructions on uploading the logs to Amazon CloudWatch Logs, see <a href="#">Publishing Amazon Aurora MySQL logs to Amazon CloudWatch Logs (p. 924)</a> .
server_id	No	
skip-character-set-client-handshake	Yes	
skip_name_resolve	No	
slave-skip-errors	Yes	Only applies to Aurora MySQL version 2 clusters, with MySQL 5.7 compatibility.
source_verify_checksum	Yes	Aurora MySQL version 3 and higher
sync_frm	Yes	Removed from Aurora MySQL version 3.
time_zone	Yes	
tls_version	Yes	For more information, see <a href="#">TLS versions for Aurora MySQL (p. 684)</a> .

## 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 <a href="#">Amazon Aurora MySQL lab mode (p. 939)</a> . 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 <a href="#">Amazon Aurora MySQL out of memory issues (p. 1765)</a> .
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.

Parameter name	Modifiable	Notes
		For more information, see <a href="#">Working with parallel query for Amazon Aurora MySQL (p. 790)</a> .
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 <code>aurora_parallel_query</code> instead. For more information, see <a href="#">Working with parallel query for Amazon Aurora MySQL (p. 790)</a> .
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	
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_zstd	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	

Parameter name	Modifiable	Notes
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	Sometimes	Indicates the status of the Event Scheduler.  Modifiable only at the cluster level in Aurora MySQL version 3.
explicit_defaults_for_timestamp	Yes	
flush	No	
flush_time	Yes	
ft_boolean_syntax	No	
ft_max_word_len	Yes	
ft_min_word_len	Yes	
ft_query_expansion_limit	Yes	
ft_stopword_file	Yes	
general_log	Yes	For instructions on uploading the logs to CloudWatch Logs, see <a href="#">Publishing Amazon Aurora MySQL logs to Amazon CloudWatch Logs (p. 924)</a> .
general_log_file	No	Aurora MySQL uses managed instances where you don't access the file system directly.
group_concat_max_len	Yes	
host_cache_size	Yes	
init_connect	Yes	
innodb_adaptive_hash_index	Yes	
innodb_adaptive_max_sleep_delay	Yes	Modifying this parameter has no effect, because innodb_thread_concurrency is always 0 for Aurora.
innodb_autoextend_increment	Yes	
innodb_buffer_pool_dump_at_shutdown	No	

Parameter name	Modifiable	Notes
innodb_buffer_pool_dump_now	No	
innodb_buffer_pool_filename	No	
innodb_buffer_pool_load_abort	No	
innodb_buffer_pool_load_at_startup	No	
innodb_buffer_pool_load_now	No	
innodb_buffer_pool_size	Yes	The default value is represented by a formula. For details about how the DBInstanceClassMemory value in the formula is calculated, see <a href="#">DB parameter formula variables (p. 239)</a> .
innodb_change_buffer_max_size	No	Aurora MySQL doesn't use the InnoDB change buffer at all.
innodb_compression_failure_threshold_pct	Yes	
innodb_compression_level	Yes	
innodb_compression_pad_pct_max	Yes	
innodb_concurrency_tickets	Yes	Modifying this parameter has no effect, because innodb_thread_concurrency is always 0 for Aurora.
innodb_deadlock_detect	Yes	This option is used to disable deadlock detection on Aurora MySQL version 3.  On high-concurrency systems, deadlock detection can cause a slowdown when numerous threads wait for the same lock. Consult the MySQL documentation for more information on this parameter.
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	

Parameter name	Modifiable	Notes
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	
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 innodb_thread_concurrency is always 0 for Aurora.

Parameter name	Modifiable	Notes
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_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 <a href="#">Replication between Aurora and MySQL or between Aurora and another Aurora DB cluster (binary log replication) (p. 841)</a> .
log_bin_trust_function_creators	Yes	
log_bin_use_v1_row_events	Yes	Removed from Aurora MySQL version 3.
log_error	No	
log_output	Yes	
log_queries_not_using_indexes	Yes	
log_slave_updates	No	Aurora MySQL version 1 and 2. Use <code>log_replica_updates</code> 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	

Parameter name	Modifiable	Notes
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 <a href="#">DB parameter formula variables (p. 239)</a> . For the default values depending on the instance class, see <a href="#">Maximum connections to an Aurora MySQL DB instance (p. 722)</a> .
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.
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	

Parameter name	Modifiable	Notes
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 <a href="#">Best practices with Amazon Aurora MySQL (p. 940)</a> .
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	
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	

Parameter name	Modifiable	Notes
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	
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	

Parameter name	Modifiable	Notes
performance_schema_max_table_instances	Yes	
performance_schema_max_table_lock_time	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.
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 <a href="#">DB parameter formula variables (p. 239)</a> .  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	

Parameter name	Modifiable	Notes
<code>read_only</code>	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.
<code>read_rnd_buffer_size</code>	Yes	
<code>relay-log</code>	No	
<code>relay_log_info_repository</code>	Yes	Removed from Aurora MySQL version 3.
<code>relay_log_recovery</code>	No	
<code>safe-user-create</code>	Yes	
<code>secure_auth</code>	Yes	Removed from Aurora MySQL version 3.
<code>secure_file_priv</code>	No	Aurora MySQL uses managed instances where you don't access the file system directly.
<code>skip-slave-start</code>	No	
<code>skip_external_locking</code>	No	
<code>skip_show_database</code>	Yes	
<code>slave_checkpoint_group</code>	Yes	Aurora MySQL version 1 and 2. Use <code>replica_checkpoint_group</code> in Aurora MySQL version 3.
<code>replica_checkpoint_group</code>	Yes	Aurora MySQL version 3 and higher
<code>slave_checkpoint_period</code>	Yes	Aurora MySQL version 1 and 2. Use <code>replica_checkpoint_period</code> in Aurora MySQL version 3.
<code>replica_checkpoint_period</code>	Yes	Aurora MySQL version 3 and higher
<code>slave_parallel_workers</code>	Yes	Aurora MySQL version 1 and 2. Use <code>replica_parallel_workers</code> in Aurora MySQL version 3.
<code>replica_parallel_workers</code>	Yes	Aurora MySQL version 3 and higher
<code>slave_pending_jobs_size_max</code>	Yes	Aurora MySQL version 1 and 2. Use <code>replica_pending_jobs_size_max</code> in Aurora MySQL version 3.
<code>replica_pending_jobs_size_max</code>	Yes	Aurora MySQL version 3 and higher
<code>replica_skip_errors</code>	Yes	Aurora MySQL version 3 and higher

Parameter name	Modifiable	Notes
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 <a href="#">Publishing Amazon Aurora MySQL logs to Amazon CloudWatch Logs (p. 924)</a> .
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	
table_cache_element_ttl	No	
table_definition_cache	Yes	The default value is represented by a formula. For details about how the <code>DBInstanceClassMemory</code> value in the formula is calculated, see <a href="#">DB parameter formula variables (p. 239)</a> .
table_open_cache	Yes	The default value is represented by a formula. For details about how the <code>DBInstanceClassMemory</code> value in the formula is calculated, see <a href="#">DB parameter formula variables (p. 239)</a> .
table_open_cache_instances	Yes	
temp_pool	Yes	Removed from Aurora MySQL version 3.

Parameter name	Modifiable	Notes
temptable_max_mmap	Yes	This parameter applies to Aurora MySQL version 3 and higher. For details, see <a href="#">New temporary table behavior in Aurora MySQL version 3 (p. 655)</a> .
temptable_max_ram	Yes	This parameter applies to Aurora MySQL version 3 and higher. For details, see <a href="#">New temporary table behavior in Aurora MySQL version 3 (p. 655)</a> .
temptable_use_mmap	Yes	This parameter applies to Aurora MySQL version 3 and higher. For details, see <a href="#">New temporary table behavior in Aurora MySQL version 3 (p. 655)</a> .
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 <code>tx_isolation</code> .
transaction_prealloc_size	Yes	
tx_isolation	Yes	Removed from Aurora MySQL version 3. It is replaced by <code>transaction_isolation</code> .
updatable_views_with_limit	Yes	
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	

Parameter name	Modifiable	Notes
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`. This parameter doesn't apply to Aurora MySQL version 1 and 2. It is available in Aurora MySQL version 3.
- `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`
- `key_buffer_size` – Key cache for MyISAM tables. This parameter doesn't apply to Aurora.
- `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 \(p. 750\)](#).

#### 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 \(p. 753\)](#).

#### 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 \(p. 756\)](#).

#### 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 \(p. 758\)](#).

### **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 \(p. 760\)](#).

### **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 \(p. 762\)](#).

### **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 \(p. 766\)](#).

### **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 \(p. 773\)](#).

### **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 \(p. 775\)](#).

### **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 \(p. 777\)](#).

### **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 \(p. 780\)](#).

#### **synch/mutex/mysys/KEY\_CACHE::cache\_lock**

The keycache->cache\_lock mutex controls access to the key cache for MyISAM tables. This wait event doesn't apply to Aurora MySQL, because the MyISAM storage engine isn't supported for persistent tables.

#### **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. For more information, see [How MySQL opens and closes tables](#).

#### **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. For more information, see [How MySQL opens and closes tables](#).

#### **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/innodb/dict**

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\_operation\_lock**

In this wait event, there are threads waiting on an rwlock held on the InnoDB data dictionary.

#### **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 \(p. 781\)](#).

#### **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.

For more information on troubleshooting synch wait events, see [Why is my MySQL DB instance showing a high number of active sessions waiting on SYNCH wait events in Performance Insights?](#).

## 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 \(p. 785\)](#).

### **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 \(p. 788\)](#).

### **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 \(p. 181\)](#).

## 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		<b>Q1:</b> <code>SELECT COUNT(*) FROM big_table;</code>	<b>Q2:</b> <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		<b>Q3:</b> SELECT COUNT(*) FROM big_table;	<b>Q4:</b> SELECT COUNT(*) FROM big_table;
T10		If Q3 finishes now, result is 999,999.	If Q4 finishes now, result is 999,999.
T11		<b>Q5:</b> SELECT COUNT(*) FROM parent_table p JOIN child_table c ON (p.id = c.id) WHERE p.id = 1000;	<b>Q6:</b> 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 \(p. 945\)](#).

## 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 Amazon Aurora MySQL \(p. 863\)](#).

### Topics

- [mysql.rds\\_assign\\_gtids\\_to\\_anonymous\\_transactions \(Aurora MySQL version 3 and higher\) \(p. 984\)](#)
- [mysql.rds\\_set\\_master\\_auto\\_position \(Aurora MySQL version 1 and 2\) \(p. 985\)](#)
- [mysql.rds\\_set\\_source\\_auto\\_position \(Aurora MySQL version 3 and higher\) \(p. 986\)](#)
- [mysql.rds\\_set\\_external\\_master\\_with\\_auto\\_position \(Aurora MySQL version 1 and 2\) \(p. 986\)](#)
- [mysql.rds\\_set\\_external\\_source\\_with\\_auto\\_position \(Aurora MySQL version 3 and higher\) \(p. 988\)](#)
- [mysql.rds\\_skip\\_transaction\\_with\\_gtid \(p. 989\)](#)

## 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);
```

### 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.

## 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 \(p. 989\)](#) stored procedure. For more information about working with GTID-based replication, see [Using GTID-based replication for Amazon Aurora MySQL \(p. 863\)](#).

## 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 \(p. 989\)](#) stored procedure. For more information about working with GTID-based replication, see [Using GTID-based replication for Amazon Aurora MySQL \(p. 863\)](#).

## 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',  
    'SomePassW0rd');
```

## [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.

Amazon Aurora releases are made available to all AWS Regions over the course of multiple days. Some Regions might temporarily show an engine version that isn't available in a different Region yet.

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](#).

For details about the Aurora MySQL versions that are supported by Amazon Aurora, see the [Release Notes for Aurora MySQL](#).

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 \(p. 991\)](#)
- [Preparing for Amazon Aurora MySQL-Compatible Edition version 1 end of life \(p. 994\)](#)
- [Upgrading Amazon Aurora MySQL DB clusters \(p. 996\)](#)
- [Database engine updates for Amazon Aurora MySQL version 3 \(p. 1016\)](#)
- [Database engine updates for Amazon Aurora MySQL version 2 \(p. 1016\)](#)
- [Database engine updates for Amazon Aurora MySQL version 1 \(p. 1017\)](#)
- [Database engine updates for Aurora MySQL Serverless clusters \(p. 1017\)](#)
- [MySQL bugs fixed by Aurora MySQL database engine updates \(p. 1017\)](#)

- [Security vulnerabilities fixed in Amazon Aurora MySQL \(p. 1017\)](#)

## 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 \(p. 991\)](#)
- [Checking Aurora MySQL versions using SQL \(p. 992\)](#)
- [Aurora MySQL long-term support \(LTS\) releases \(p. 993\)](#)
- [Upgrade paths between 5.6-compatible and 5.7-compatible clusters \(p. 993\)](#)

## 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 1.17.9, 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 onwards. You can refer to release notes of specific 3.x version to find the corresponding MySQL compatible version.

For example, the engine versions for Aurora MySQL 3.02.0, 2.03.2, and 1.17.9 are the following.

```
8.0.mysql_aurora.3.02.0
5.7.mysql_aurora.2.03.2
5.6.mysql_aurora.1.17.9
```

### 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](#), [Database engine updates for Amazon](#)

Aurora MySQL version 2 and [Database engine updates for Amazon Aurora MySQL version 1](#) in the *Release Notes for Aurora MySQL*. For a chronological list of new features and releases, see [Document history \(p. 1771\)](#). To check the minimum version required for a security-related fix, see [Security vulnerabilities fixed in Aurora MySQL](#) in the *Release Notes for Aurora MySQL*.

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 1.17.9 are represented by the engine version 5.6.10a. These older version designations don't include the 5.7.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 1.17.9 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 \(p. 992\)](#).

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 1.17.9 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 1.17.9 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 \(p. 991\)](#) 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 three years 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 \(version 2.07.7\)](#) in the *Release Notes for Aurora MySQL*.
- Aurora MySQL version 1.22.\*. For more details about this version, see [Aurora MySQL database engine updates 2021-06-03 \(version 1.22.5\)](#) in the *Release Notes for Aurora MySQL*.

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 (provisioned and Aurora Serverless) 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 \(p. 1006\)](#). 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 \(p. 1002\)](#)

For Aurora Serverless v1 DB clusters, you can perform an in-place upgrade from Aurora MySQL version 1 to Aurora MySQL version 2. For more details about this method, see [Modifying an Aurora Serverless v1 DB cluster \(p. 1570\)](#).

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 \(p. 1002\)](#). Please note the [Feature differences between Aurora MySQL version 2 and 3 \(p. 658\)](#).

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 (provisioned and Aurora Serverless) 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 \(p. 193\)](#), because you can't complete steps 5 and 6 after this time. You will also be unable to create a new cross-Region read replica running Aurora MySQL version 1. 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 (not applicable for Aurora Serverless DB clusters).
  - Change instance configuration.
  - Perform point-in-time restore.
  - Create clones of existing version 1 clusters.
  - Create a new cross-Region read replica running Aurora MySQL version 2 or higher.

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 \(p. 1001\)](#).

## Finding clusters affected by this end-of-life process

To find clusters affected by this end-of-life process, use the following procedures.

### Important

Be sure to perform these instructions in every AWS Region where your resources are located.

### 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 \(p. 991\)](#).

### Topics

- [Upgrading the minor version or patch level of an Aurora MySQL DB cluster \(p. 996\)](#)
- [Upgrading the major version of an Aurora MySQL DB cluster \(p. 1001\)](#)

## 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 \(p. 996\)](#) (for Aurora MySQL 1.19.0 and higher, or 2.03.2 and higher)
- [Enabling automatic upgrades between minor Aurora MySQL versions \(p. 997\)](#)
- [Upgrading Aurora MySQL by applying pending maintenance to an Aurora MySQL DB cluster \(p. 998\)](#) (before Aurora MySQL 1.19.0)

For information about how zero-downtime patching can reduce interruptions during the upgrade process, see [Using zero-downtime patching \(p. 999\)](#).

### 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 \(p. 248\)](#).
- **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. You do so by using 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.

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.

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 \(p. 249\)](#). 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 `DBClusterIdentifier` 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 \(p. 321\)](#).

**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.

The following table shows the Aurora MySQL versions and DB instance classes where ZDP is available.

Version	db.r* instance classes	db.t* instance classes	db.serverless instance class
2.07.2 and higher 2.07 versions	No	Yes	N/A
2.10.0 and higher 2.10 versions	Yes	Yes	N/A
3.01.0 and 3.01.1	Yes	Yes	N/A
3.02.0 and higher 3.x versions	Yes	Yes	Yes

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 to use ZDP on the **Events** page in the AWS Management Console.

In Aurora MySQL version 2.10 and higher and version 3, 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 1.x and 2.x 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

Original version	Upgraded version	Does ZDP apply?
Aurora MySQL 2.*, before 2.07.2	Any	No
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.
Aurora MySQL 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.
Aurora MySQL 2.10.*	2.10.*	Yes, on the writer instance for db.t* and db.r* instances. 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 \(p. 1002\)](#)
- [Upgrading from Aurora MySQL 1.x to 2.x \(p. 1002\)](#)
- [Planning a major version upgrade for an Aurora MySQL cluster \(p. 1002\)](#)
- [Aurora MySQL major version upgrade paths \(p. 1003\)](#)
- [How the Aurora MySQL in-place major version upgrade works \(p. 1005\)](#)
- [How to perform an in-place upgrade \(p. 1006\)](#)
- [How in-place upgrades affect the parameter groups for a cluster \(p. 1008\)](#)
- [Changes to cluster properties between Aurora MySQL version 1 and 2 \(p. 1008\)](#)
- [In-place major upgrades for global databases \(p. 1009\)](#)
- [After the upgrade \(p. 1010\)](#)
- [Troubleshooting for Aurora MySQL in-place upgrade \(p. 1010\)](#)
- [Aurora MySQL in-place upgrade tutorial \(p. 1011\)](#)
- [Alternative blue-green upgrade technique \(p. 1016\)](#)

## 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 \(p. 653\)](#). For details and examples of performing this type of upgrade, see [Upgrade planning for Aurora MySQL version 3 \(p. 667\)](#) and [Upgrading to Aurora MySQL version 3 \(p. 666\)](#).

### 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 \(p. 375\)](#), or other upgrade techniques such as the one described in [Alternative blue-green upgrade technique \(p. 1016\)](#).

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 \(p. 1008\)](#) and [Changes to cluster properties between Aurora MySQL version 1 and 2 \(p. 1008\)](#).

You can learn the sorts of issues that you might encounter during the upgrade by reading [Troubleshooting for Aurora MySQL in-place upgrade \(p. 1010\)](#). 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 Amazon Aurora DB cluster \(p. 280\)](#).
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 \(p. 1006\)](#). 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, lower than 1.22.3	Yes	If your cluster is running a version that's lower than Aurora MySQL 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.
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.

Type of Aurora MySQL DB cluster	Can it use in-place upgrade?	Action
Aurora MySQL provisioned cluster, 3.1.0 or higher	No	For information about upgrading to Aurora MySQL version 3, see <a href="#">Upgrade planning for Aurora MySQL version 3 (p. 667)</a> and <a href="#">Upgrading to Aurora MySQL version 3 (p. 666)</a> .
Aurora Serverless v1 cluster	Yes	You can upgrade from a MySQL 5.6-compatible Aurora Serverless v1 version to a MySQL 5.7-compatible one by performing an in-place upgrade. For more information, see <a href="#">Modifying an Aurora Serverless v1 DB cluster (p. 1570)</a> .
Cluster in an Aurora global database	Yes	Follow the <a href="#">procedure for doing an in-place upgrade (p. 1006)</a> 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. You also can't upgrade a multi-master cluster by performing a snapshot restore. If you need to move your data from a multi-master cluster to an Aurora MySQL 5.7-compatible or 8.0-compatible cluster, use a logical dump produced by a tool such as AWS Database Migration Service (AWS DMS) or the <code>mysqldump</code> command.
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 <a href="#">Upgrade considerations for parallel query (p. 802)</a> . 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.

Type of Aurora MySQL DB cluster	Can it use in-place upgrade?	Action
Cluster with zero DB instances	No	<p>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.</p> <p>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.</p>
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 [Working with Amazon RDS event notification \(p. 572\)](#).

### 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 \(p. 1010\)](#). 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 \(p. 1005\)](#). Perform any pre-upgrade planning and testing, as described in [Planning a major version upgrade for an Aurora MySQL cluster \(p. 1002\)](#).
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 \(p. 1008\)](#).
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 \(p. 1010\)](#) 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 \(p. 1008\)](#).

**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 \(p. 1011\)](#).

**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 \(p. 1009\)](#).

## 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 \(p. 1009\)](#).

## 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.  
aws rds 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.  
aws rds 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 \(p. 1005\)](#). 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	En
global-cluster	Global database	Au
cluster1	Primary cluster	Au
dbinstance-1	Writer instance	Au
cluster-2	Secondary cluster	Au
dbinstance-2	Reader instance	Au

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 specify a custom parameter group for the global database cluster while you're performing a major version upgrade of that Aurora global database. Create your custom parameter groups in each Region of the global cluster and then apply them manually to the Regional clusters after the upgrade.

## 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

Use the following tips to help troubleshoot problems with Aurora MySQL in-place upgrades. These tips don't apply to Aurora Serverless DB clusters.

Reason for in-place upgrade being canceled or slow	Effect	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.
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

Reason for in-place upgrade being canceled or slow	Effect	Solution to allow in-place upgrade to complete within maintenance window
		<p>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.</p> <p>Consider performing the upgrade only after the history list length is smaller.</p>
Cluster is in the process of committing a large binary log transaction	Upgrade might take a long time.	<p>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.</p> <p>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.</p>

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,DBInstanceState:DBInstanceState} | [0]{
{
    "DBInstanceIdentifier": "instance-2020-11-17-5158",
    "DBInstanceState": "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

For this information that was formerly in this guide, see [the corresponding page in the \*Release notes for Amazon Aurora MySQL-Compatible Edition\*](#).

## Database engine updates for Amazon Aurora MySQL version 2

For this information that was formerly in this guide, see [the corresponding page in the \*Release notes for Amazon Aurora MySQL-Compatible Edition\*](#).

## Database engine updates for Amazon Aurora MySQL version 1

For this information that was formerly in this guide, see [the corresponding page in the \*Release notes for Amazon Aurora MySQL-Compatible Edition\*](#).

### Database engine updates for Aurora MySQL Serverless clusters

For this information that was formerly in this guide, see [the corresponding page in the \*Release notes for Amazon Aurora MySQL-Compatible Edition\*](#).

### MySQL bugs fixed by Aurora MySQL database engine updates

For this information that was formerly in this guide, see [the corresponding page in the \*Release notes for Amazon Aurora MySQL-Compatible Edition\*](#).

### Security vulnerabilities fixed in Amazon Aurora MySQL

For this information that was formerly in this guide, see [the corresponding page in the \*Release notes for Amazon Aurora MySQL-Compatible Edition\*](#).

# Working with Amazon Aurora PostgreSQL

Amazon Aurora PostgreSQL is a fully managed, PostgreSQL-compatible, and ACID-compliant relational database engine that combines the speed, reliability, and manageability of Amazon Aurora 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. To learn more about Aurora in general, see [What is Amazon Aurora? \(p. 1\)](#).

In addition to the benefits of Aurora, Aurora PostgreSQL offers a convenient migration pathway from Amazon RDS into Aurora, with push-button migration tools that convert your existing RDS for PostgreSQL applications to Aurora PostgreSQL. Routine database tasks such as provisioning, patching, backup, recovery, failure detection, and repair are also easy to manage with 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 \(p. 1018\)](#)
- [Updating applications to connect to Aurora PostgreSQL DB clusters using new SSL/TLS certificates \(p. 1032\)](#)
- [Using Kerberos authentication with Aurora PostgreSQL \(p. 1035\)](#)
- [Migrating data to Amazon Aurora with PostgreSQL compatibility \(p. 1048\)](#)
- [Using Babelfish for Aurora PostgreSQL \(p. 1063\)](#)
- [Managing Amazon Aurora PostgreSQL \(p. 1143\)](#)
- [Tuning with wait events for Aurora PostgreSQL \(p. 1152\)](#)
- [Best practices with Amazon Aurora PostgreSQL \(p. 1200\)](#)
- [Replication with Amazon Aurora PostgreSQL \(p. 1224\)](#)
- [Integrating Amazon Aurora PostgreSQL with other AWS services \(p. 1230\)](#)
- [Managing query execution plans for Aurora PostgreSQL \(p. 1290\)](#)
- [Working with extensions and foreign data wrappers \(p. 1318\)](#)
- [Amazon Aurora PostgreSQL reference \(p. 1341\)](#)
- [Amazon Aurora PostgreSQL updates \(p. 1413\)](#)

## Security with Amazon Aurora PostgreSQL

For a general overview of Aurora security, see [Security in Amazon Aurora \(p. 1634\)](#). You can manage security for Amazon Aurora PostgreSQL at a few different levels:

- To control who can perform Amazon RDS management actions on Aurora PostgreSQL DB clusters and DB instances, use AWS Identity and Access Management (IAM). IAM handles the authentication of user identity before the user can access the service. It also handles authorization, that is, whether the user is allowed to do what they're trying to do. IAM database authentication is an additional authentication method that you can choose when you create your Aurora PostgreSQL DB cluster. For more information, see [Identity and access management for Amazon Aurora \(p. 1653\)](#).

If you do use IAM with your Aurora PostgreSQL DB cluster, sign in to the AWS Management Console with your IAM credentials first, before opening the Amazon RDS console at <https://console.aws.amazon.com/rds/>.

- Make sure to create Aurora DB clusters in a virtual public 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, use a VPC security group. You can make these endpoint and port connections by 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 VPC VPCs and Amazon Aurora \(p. 1729\)](#).

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 \(p. 56\)](#). For more information about default and dedicated VPC tenancy, see [Dedicated instances](#) in the [Amazon Elastic Compute Cloud User Guide](#).

- To grant permissions to the PostgreSQL databases running on your Amazon Aurora DB cluster, you can take the same general approach as with stand-alone instances 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 databases, schemas, and tables.

PostgreSQL manages privileges by using *roles*. The `rds_superuser` role is the most privileged role on an Aurora PostgreSQL DB cluster. This role is created automatically, and it's granted to the user that creates the DB cluster (the master user account, `postgres` by default). To learn more, see [Understanding PostgreSQL roles and permissions \(p. 1019\)](#).

Aurora PostgreSQL 14.3 and higher releases support the Salted Challenge Response Authentication Mechanism (SCRAM) for passwords as an alternative to message digest (MD5). We recommend that you use SCRAM because it's more secure than MD5. For more information, including how to migrate database user passwords from MD5 to SCRAM, see [Using SCRAM for PostgreSQL password encryption \(p. 1024\)](#).

## Understanding PostgreSQL roles and permissions

When you create an Aurora PostgreSQL DB cluster using the AWS Management Console, an administrator account is created at the same time. By default, its name is `postgres`, as shown in the following screenshot:

The screenshot shows the 'Credentials Settings' step of the 'Create DB Cluster' wizard. It includes fields for 'Master username' (set to 'postgres'), 'Master password', and 'Confirm password', each with a masked input field.

You can choose another name rather than accept the default (`postgres`). If you do, the name you choose must start with a letter and be between 1 and 16 alphanumeric characters. For simplicity's sake, we refer to this main user account by its default value (`postgres`) throughout this guide.

If you use the `create-db-cluster` AWS CLI rather than the AWS Management Console, you create the user name by passing it with the `master-username` parameter. For more information, see [Create an Aurora PostgreSQL DB cluster \(p. 101\)](#).

Whether you use the AWS Management Console, the AWS CLI, or the Amazon RDS API, and whether you use the default `postgres` name or choose a different name, this first database user account is a member of the `rds_superuser` group and has `rds_superuser` privileges.

#### Topics

- [Understanding the rds\\_superuser role \(p. 1020\)](#)
- [Controlling user access to the PostgreSQL database \(p. 1022\)](#)
- [Delegating and controlling user password management \(p. 1023\)](#)
- [Using SCRAM for PostgreSQL password encryption \(p. 1024\)](#)

## Understanding the `rds_superuser` role

In PostgreSQL, a *role* can define a user, a group, or a set of specific permissions granted to a group or user for various objects in the database. PostgreSQL commands to `CREATE USER` and `CREATE GROUP` have been replaced by the more general, `CREATE ROLE` with specific properties to distinguish database users. A database user can be thought of as a role with the `LOGIN` privilege.

#### Note

The `CREATE USER` and `CREATE GROUP` commands can still be used. For more information, see [Database Roles](#) in the PostgreSQL documentation.

The `postgres` user is the most highly privileged database user on your Aurora PostgreSQL DB cluster. It has the characteristics defined by the following `CREATE ROLE` statement.

```
CREATE ROLE postgres WITH LOGIN NOSUPERUSER INHERIT CREATEDB CREATEROLE NOREPLICATION VALID UNTIL 'infinity'
```

The properties `NOSUPERUSER`, `NOREPLICATION`, `INHERIT`, and `VALID UNTIL 'infinity'` are the default options for `CREATE ROLE`, unless otherwise specified.

By default, `postgres` has privileges granted to the `rds_superuser` role. The `rds_superuser` role allows the `postgres` user to do the following:

- Add extensions that are available for use with Aurora PostgreSQL. For more information, see [Working with extensions and foreign data wrappers \(p. 1318\)](#).
- Create roles for users and grant privileges to users. For more information, see [CREATE ROLE](#) and [GRANT](#) in the PostgreSQL documentation.
- Create databases. For more information, see [CREATE DATABASE](#) in the PostgreSQL documentation.
- Grant `rds_superuser` privileges to user roles that don't have these privileges, and revoke privileges as needed. We recommend that you grant this role only to those users who perform superuser tasks. In other words, you can grant this role to database administrators (DBAs) or system administrators.
- Grant (and revoke) the `rds_replication` role to database users that don't have the `rds_superuser` role.
- Grant (and revoke) the `rds_password` role to database users that don't have the `rds_superuser` role.
- Obtain status information about all database connections by using the `pg_stat_activity` view. When needed, `rds_superuser` can stop any connections by using `pg_terminate_backend` or `pg_cancel_backend`.

In the `CREATE ROLE postgres...` statement, you can see that the `postgres` user role specifically disallows PostgreSQL superuser permissions. Aurora PostgreSQL is a managed service, so you can't access the host OS, and you can't connect using the PostgreSQL superuser account. Many of the tasks that require superuser access on a stand-alone PostgreSQL are managed automatically by Aurora.

For more information about granting privileges, see [GRANT](#) in the PostgreSQL documentation.

The `rds_superuser` role is one of several *predefined* roles in an Aurora PostgreSQL DB cluster.

**Note**

In PostgreSQL 13 and earlier releases, *predefined* roles are known as *default* roles.

In the following list, you find some of the other predefined roles that are created automatically for a new Aurora PostgreSQL DB cluster. Predefined roles and their privileges can't be changed. You can't drop, rename, or modify privileges for these predefined roles. Attempting to do so results in an error.

- **rds\_password** – A role that can change passwords and set up password constraints for database users. The `rds_superuser` role is granted this role by default, and can grant the role to database users. For more information, see [Controlling user access to the PostgreSQL database \(p. 1022\)](#).
- **rdsadmin** – A role that's created to handle many of the management tasks that the administrator with `superuser` privileges would perform on a standalone PostgreSQL database. This role is used internally by Aurora PostgreSQL for many management tasks.

To see all predefined roles, you can connect to the primary instance of your Aurora PostgreSQL DB cluster and use the `\du` metacommand. The output looks as follows:

List of roles		
Role name	Attributes	Member of
postgres	Create role, Create DB   Password valid until infinity	+  {rds_superuser}
rds_superuser	Cannot login	{pg_monitor,pg_signal_backend, +  rds_replication,rds_password}
...		

In the output, you can see that `rds_superuser` isn't a database user role (it can't login), but it has the privileges of many other roles. You can also see that database user `postgres` is a member of the `rds_superuser` role. As mentioned previously, `postgres` is the default value in the Amazon RDS console's [Create database](#) page. If you chose another name, that name is shown in the list of roles instead.

## Controlling user access to the PostgreSQL database

New databases in PostgreSQL are always created with a default set of privileges in the database's `public` schema that allow all database users and roles to create objects. These privileges allow database users to connect to the database, for example, and create temporary tables while connected.

To better control user access to the databases instances that you create on your Aurora PostgreSQL DB cluster primary node , we recommend that you revoke these default `public` privileges. After doing so, you then grant specific privileges for database users on a more granular basis, as shown in the following procedure.

### To set up roles and privileges for a new database instance

Suppose you're setting up a database on a newly created Aurora PostgreSQL DB cluster for use by several researchers, all of whom need read-write access to the database.

1. Use `psql` (or `pgAdmin`) to connect to the primary DB instance on your Aurora PostgreSQL DB cluster:

```
psql --host=your-cluster-instance-1.666666666666.aws-region.rds.amazonaws.com --  
port=5432 --username=postgres --password
```

When prompted, enter your password. The `psql` client connects and displays the default administrative connection database, `postgres=>`, as the prompt.

2. To prevent database users from creating objects in the `public` schema, do the following:

```
postgres=> REVOKE CREATE ON SCHEMA public FROM PUBLIC;  
REVOKE
```

3. Next, you create a new database instance:

```
postgres=> CREATE DATABASE lab_db;  
CREATE DATABASE
```

4. Revoke all privileges from the `PUBLIC` schema on this new database.

```
postgres=> REVOKE ALL ON DATABASE lab_db FROM public;  
REVOKE
```

5. Create a role for database users.

```
postgres=> CREATE ROLE lab_tech;  
CREATE ROLE
```

6. Give database users that have this role the ability to connect to the database.

```
postgres=> GRANT CONNECT ON DATABASE lab_db TO lab_tech;  
GRANT
```

7. Grant all users with the `lab_tech` role all privileges on this database.

```
postgres=> GRANT ALL PRIVILEGES ON DATABASE lab_db TO lab_tech;  
GRANT
```

8. Create database users, as follows:

```
postgres=> CREATE ROLE lab_user1 LOGIN PASSWORD 'change_me';
```

```
CREATE ROLE
postgres=> CREATE ROLE lab_user2 LOGIN PASSWORD 'change_me';
CREATE ROLE
```

9. Grant these two users the privileges associated with the lab\_tech role:

```
postgres=> GRANT lab_tech TO lab_user1;
GRANT ROLE
postgres=> GRANT lab_tech TO lab_user2;
GRANT ROLE
```

At this point, lab\_user1 and lab\_user2 can connect to the lab\_db database. This example doesn't follow best practices for enterprise usage, which might include creating multiple database instances, different schemas, and granting limited permissions. For more complete information and additional scenarios, see [Managing PostgreSQL Users and Roles](#).

For more information about privileges in PostgreSQL databases, see the [GRANT command](#) in the PostgreSQL documentation.

## Delegating and controlling user password management

As a DBA, you might want to delegate the management of user passwords. Or, you might want to prevent database users from changing their passwords or reconfiguring password constraints, such as password lifetime. To ensure that only the database users that you choose can change password settings, you can turn on the restricted password management feature. When you activate this feature, only those database users that have been granted the rds\_password role can manage passwords.

### Note

To use restricted password management, your Aurora PostgreSQL DB cluster must be running Amazon Aurora PostgreSQL 10.6 or higher.

By default, this feature is off, as shown in the following:

```
postgres=> SHOW rds.restrict_password_commands;
rds.restrict_password_commands
-----
off
(1 row)
```

To turn on this feature, you use a custom parameter group and change the setting for rds.restrict\_password\_commands to 1. Be sure to reboot your Aurora PostgreSQL's primary DB instance so that the setting takes effect.

With this feature active, rds\_password privileges are needed for the following SQL commands:

```
CREATE ROLE myrole WITH PASSWORD 'mypassword';
CREATE ROLE myrole WITH PASSWORD 'mypassword' VALID UNTIL '2023-01-01';
ALTER ROLE myrole WITH PASSWORD 'mypassword' VALID UNTIL '2023-01-01';
ALTER ROLE myrole WITH PASSWORD 'mypassword';
ALTER ROLE myrole VALID UNTIL '2023-01-01';
ALTER ROLE myrole RENAME TO myrole2;
```

Renaming a role (ALTER ROLE myrole RENAME TO newname) is also restricted if the password uses the MD5 hashing algorithm.

With this feature active, attempting any of these SQL commands without the rds\_password role permissions generates the following error:

```
ERROR: must be a member of rds_password to alter passwords
```

We recommend that you grant the `rds_password` to only a few roles that you use solely for password management. If you grant `rds_password` privileges to database users that don't have `rds_superuser` privileges, you need to also grant them the `CREATEROLE` attribute.

Make sure that you verify password requirements such as expiration and needed complexity on the client side. If you use your own client-side utility for password related changes, the utility needs to be a member of `rds_password` and have `CREATE ROLE` privileges.

## Using SCRAM for PostgreSQL password encryption

The *Salted Challenge Response Authentication Mechanism (SCRAM)* is an alternative to PostgreSQL's default message digest (MD5) algorithm for encrypting passwords. The SCRAM authentication mechanism is considered more secure than MD5. To learn more about these two different approaches to securing passwords, see [Password Authentication](#) in the PostgreSQL documentation.

We recommend that you use SCRAM rather than MD5 as the password encryption scheme for your Aurora PostgreSQL DB cluster. As of the Aurora PostgreSQL 14 release, SCRAM is supported in all available Aurora PostgreSQL versions, including versions 10, 11, 12, 13, and 14. It's a cryptographic challenge-response mechanism that uses the `scram-sha-256` algorithm for password authentication and encryption.

You might need to update libraries for your client applications to support SCRAM. For example, JDBC versions before 42.2.0 don't support SCRAM. For more information, see [PostgreSQL JDBC Driver](#) in the PostgreSQL JDBC Driver documentation. For a list of other PostgreSQL drivers and SCRAM support, see [List of drivers](#) in the PostgreSQL documentation.

### Note

Aurora PostgreSQL version 14 and higher support `scram-sha-256` for password encryption by default for new DB clusters. That is, the default DB cluster parameter group (`default.aurora-postgresql14`) has its `password_encryption` value set to `scram-sha-256`.

## Setting up Aurora PostgreSQL DB cluster to require SCRAM

For Aurora PostgreSQL 14.3 and higher versions, you can require the Aurora PostgreSQL DB cluster to accept only passwords that use the `scram-sha-256` algorithm.

### Note

Currently, this capability isn't supported with RDS Proxy.

Before making changes to your system, be sure you understand the complete process, as follows:

- Get information about all roles and password encryption for all database users.
- Double-check the parameter settings for your Aurora PostgreSQL DB cluster for the parameters that control password encryption.
- If your Aurora PostgreSQL DB cluster uses a default parameter group, you need to create a custom DB cluster parameter group and apply it to your Aurora PostgreSQL DB cluster so that you can modify parameters when needed. If your Aurora PostgreSQL DB cluster uses a custom parameter group, you can modify the necessary parameters later in the process, as needed.
- Change the `password_encryption` parameter to `scram-sha-256`.
- Notify all database users that they need to update their passwords. Do the same for your `postgres` account. The new passwords are encrypted and stored using the `scram-sha-256` algorithm.
- Verify that all passwords are encrypted using as the type of encryption.

- If all passwords use scram-sha-256, you can change the `rds.accepted_password_auth_method` parameter from `md5+scram` to `scram-sha-256`.

**Warning**

After you change `rds.accepted_password_auth_method` to `scram-sha-256` alone, any users (roles) with `md5`-encrypted passwords can't connect.

## Getting ready to require SCRAM for your Aurora PostgreSQL DB cluster

Before making any changes to your Aurora PostgreSQL DB cluster, check all existing database user accounts. Also, check the type of encryption used for passwords. You can do these tasks by using the `rds_tools` extension. This extension is supported on Aurora PostgreSQL 13.1 and higher releases.

### To get a list of database users (roles) and password encryption methods

1. Use `psql` to connect to the primary instance of your Aurora PostgreSQL DB cluster , as shown in the following.

```
psql --host=cluster-name-instance-1.111122223333.aws-region.rds.amazonaws.com --  
port=5432 --username=postgres --password
```

2. Install the `rds_tools` extension.

```
postgres=> CREATE EXTENSION rds_tools;  
CREATE EXTENSION
```

3. Get a listing of roles and encryption.

```
postgres=> SELECT * FROM  
rds_tools.role_password_encryption_type();
```

You see output similar to the following.

rolname	encryption_type
pg_monitor	
pg_read_all_settings	
pg_read_all_stats	
pg_stat_scan_tables	
pg_signal_backend	
lab_tester	md5
user_465	md5
postgres	md5
(8 rows)	

## Creating a custom DB cluster parameter group

**Note**

If your Aurora PostgreSQL DB cluster already uses a custom parameter group, you don't need to create a new one.

For an overview of parameter groups for Aurora, see [Creating a DB cluster parameter group \(p. 219\)](#).

The password encryption type used for passwords is set in one parameter, `password_encryption`. The encryption that the Aurora PostgreSQL DB cluster allows is set in another parameter, `rds.accepted_password_auth_method`. Changing either of these from the default values requires that you create a custom DB cluster parameter group and apply it to your cluster.

You can also use the AWS Management Console or the RDS API to create a custom DB cluster parameter group . For more information, see [Creating a DB cluster parameter group \(p. 219\)](#).

You can now associate the custom parameter group with your DB instance.

### To create a custom DB cluster parameter group

1. Use the `create-db-cluster-parameter-group` CLI command to create the custom parameter group for the cluster. The following example uses `aurora-postgresql13` as the source for this custom parameter group.

For Linux, macOS, or Unix:

```
aws rds create-db-cluster-parameter-group --db-cluster-parameter-group-name 'docs-lab-scram-passwords' \
    --db-parameter-group-family aurora-postgresql13 --description 'Custom DB cluster parameter group for SCRAM'
```

For Windows:

```
aws rds create-db-cluster-parameter-group --db-cluster-parameter-group-name "docs-lab-scram-passwords" ^
    --db-parameter-group-family aurora-postgresql13 --description "Custom DB cluster parameter group for SCRAM"
```

You can now associate the custom parameter group with your cluster.

2. Use the `modify-db-cluster` CLI command to apply this custom parameter group to your Aurora PostgreSQL DB cluster.

For Linux, macOS, or Unix:

```
aws rds modify-db-cluster --db-cluster-identifier 'your-instance-name' \
    ----db-cluster-parameter-group-name "docs-lab-scram-passwords"
```

For Windows:

```
aws rds modify-db-cluster --db-cluster-identifier "your-instance-name" ^
    ----db-cluster-parameter-group-name "docs-lab-scram-passwords"
```

To resynchronize your Aurora PostgreSQL DB cluster with your custom DB cluster parameter group, reboot the primary and all other instances of the cluster.

### Configuring password encryption to use SCRAM

The password encryption mechanism used by an Aurora PostgreSQL DB cluster is set in the DB cluster parameter group in the `password_encryption` parameter. Allowed values are `unset`, `md5`, or `scram-sha-256`. The default value depends on the Aurora PostgreSQL version, as follows:

- Aurora PostgreSQL 14 – Default is `scram-sha-256`
- Aurora PostgreSQL 13 – Default is `md5`

With a custom DB cluster parameter group attached to your Aurora PostgreSQL DB cluster, you can modify values for the `password_encryption` parameter.

<input type="checkbox"/>	Name	Values ▾	Allowed values	Modifiable ▾	Source ▾	Apply type ▾
<input type="checkbox"/>	password_encryption	scram-sha-256	md5, scram-sha-256	true	system	dynamic
<input type="checkbox"/>	rds.accepted_password_auth_method	md5+scram	md5+scram, scram	true	system	dynamic

### To change password encryption setting to scram-sha-256

- Change the value of the password encryption to scram-sha-256, as shown following. The change can be applied immediately because the parameter is dynamic, so a restart isn't required for the change to take effect.

For Linux, macOS, or Unix:

```
aws rds modify-db-cluster-parameter-group --db-parameter-group-name \
  'docs-lab-scram-passwords' --parameters
  'ParameterName=password_encryption,ParameterValue=scram-sha-256,ApplyMethod=immediate'
```

For Windows:

```
aws rds modify-db-parameter-group --db-parameter-group-name ^
  "docs-lab-scram-passwords" --parameters
  "ParameterName=password_encryption,ParameterValue=scram-sha-256,ApplyMethod=immediate"
```

### Migrating passwords for user roles to SCRAM

You can migrate passwords for user roles to SCRAM as described following.

#### To migrate database user (role) passwords from MD5 to SCRAM

- Log in as the administrator user (default user name, `postgres`) as shown following.

```
psql --host=cluster-name-instance-1.111122223333.aws-region.rds.amazonaws.com --
port=5432 --username=postgres --password
```

- Check the setting of the `password_encryption` parameter on your RDS for PostgreSQL DB instance by using the following command.

```
postgres=> SHOW password_encryption;
 password_encryption
-----
 md5
 (1 row)
```

- Change the value of this parameter to `scram-sha-256`. This is a dynamic parameter, so you don't need to reboot the instance after making this change. Check the value again to make sure that it's now set to `scram-sha-256`, as follows.

```
postgres=> SHOW password_encryption;
 password_encryption
-----
 scram-sha-256
 (1 row)
```

4. Notify all database users to change their passwords. Be sure to also change your own password for account `postgres` (the database user with `rds_superuser` privileges).

```
labdb=> ALTER ROLE postgres WITH LOGIN PASSWORD 'change_me';
ALTER ROLE
```

5. Repeat the process for all databases on your Aurora PostgreSQL DB cluster.

### Changing parameter to require SCRAM

This is the final step in the process. After you make the change in the following procedure, any user accounts (roles) that still use `md5` encryption for passwords can't log in to the Aurora PostgreSQL DB cluster.

The `rds.accepted_password_auth_method` specifies the encryption method that the Aurora PostgreSQL DB cluster accepts for a user password during the login process. The default value is `md5+scram`, meaning that either method is accepted. In the following image, you can find the default setting for this parameter.

<input type="checkbox"/>	Name	Values	Allowed values	Modifiable	Source	Apply type
<input type="checkbox"/>	password_encryption	scram-sha-256	md5, scram-sha-256	true	system	dynamic
<input type="checkbox"/>	rds.accepted_password_auth_method	md5+scram	md5+scram, <b>scram</b>	true	system	dynamic

The allowed values for this parameter are `md5+scram` or `scram` alone. Changing this parameter value to `scram` makes this a requirement.

### To change the parameter value to require SCRAM authentication for passwords

1. Verify that all database user passwords for all databases on your Aurora PostgreSQL DB cluster use `scram-sha-256` for password encryption. To do so, query `rds_tools` for the role (user) and encryption type, as follows.

```
postgres=> SELECT * FROM rds_tools.role_password_encryption_type();
   rolname    | encryption_type
-----+-----
 pg_monitor  |
 pg_read_all_settings |
 pg_read_all_stats  |
 pg_stat_scan_tables |
 pg_signal_backend  |
 lab_tester     | scram-sha-256
 user_465      | scram-sha-256
 postgres       | scram-sha-256
( 3 rows )
```

2. Repeat the query across all DB instances in your Aurora PostgreSQL DB cluster.

If all passwords use `scram-sha-256`, you can proceed.

3. Change the value of the accepted password authentication to `scram-sha-256`, as follows.

For Linux, macOS, or Unix:

```
aws rds modify-db-cluster-parameter-group --db-cluster-parameter-group-name 'docs-lab-scram-passwords' \
```

```
--parameters
'ParameterName=rds.accepted_password_auth_method,ParameterValue=scram,ApplyMethod=immediate'
```

For Windows:

```
aws rds modify-db-cluster-parameter-group --db-cluster-parameter-group-name "docs-lab-
scram-passwords" ^
--parameters
"ParameterName=rds.accepted_password_auth_method,ParameterValue=scram,ApplyMethod=immediate"
```

## 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 \(p. 1030\)](#)
- [Determining the SSL/TLS connection status \(p. 1030\)](#)
- [Configuring cipher suites for connections to Aurora PostgreSQL DB clusters \(p. 1031\)](#)

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 \(p. 1642\)](#).

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 parameter groups \(p. 215\)](#).

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.

## Configuring cipher suites for connections to Aurora PostgreSQL DB clusters

By using configurable cipher suites, you can have more control over the security of your database connections. You can specify a list of cipher suites that you want to allow to secure client SSL/TLS connections to your database. With configurable cipher suites, you can control the connection encryption that your database server accepts. Doing this helps prevent the use of insecure or deprecated ciphers.

Configurable cipher suites is supported in Aurora PostgreSQL versions 11.8 and higher.

To specify the list of permissible ciphers for encrypting connections, modify the `ssl_ciphers` cluster parameter. Set the `ssl_ciphers` parameter in a cluster parameter group using the AWS Management Console, the AWS CLI, or the RDS API. To set cluster parameters, see [Modifying parameters in a DB cluster parameter group \(p. 221\)](#).

Set the `ssl_ciphers` parameter to a string of comma-separated cipher values. The valid ciphers include the following:

- ECDHE-RSA-AES128-SHA
- ECDHE-RSA-AES128-SHA256
- ECDHE-RSA-AES128-GCM-SHA256
- ECDHE-RSA-AES256-SHA
- ECDHE-RSA-AES256-GCM-SHA384

You can also use the `describe-engine-default-cluster-parameters` CLI command to determine which cipher suites are currently supported for a specific parameter group family. The following example shows how to get the allowed values for the `ssl_cipher` cluster parameter for Aurora PostgreSQL 11.

```
aws rds describe-engine-default-cluster-parameters --db-parameter-group-family aurora-
postresql11
```

```
...some output truncated...
{
  "ParameterName": "ssl_ciphers",
  "Description": "Sets the list of allowed TLS ciphers to be used on secure connections.",
  "Source": "engine-default",
  "ApplyType": "dynamic",
  "DataType": "list",
  "AllowedValues": "ECDHE-RSA-AES256-GCM-SHA384,ECDHE-RSA-AES256-SHA384,AES256-SHA,AES128-SHA,DES-CBC3-SHA,ADH-DES-CBC3-SHA,EDH-RSA-DES-CBC3-SHA,EDH-DSS-DES-CBC3-SHA,ADH-AES256-SHA,DHE-RSA-AES256-SHA,DHE-DSS-AES256-SHA,ADH-AES128-SHA,DHE-RSA-AES128-SHA,DHE-DSS-AES128-SHA,HIGH",
  "IsModifiable": true,
  "MinimumEngineVersion": "11.8",
  "SupportedEngineModes": [
    "provisioned"
  ],
},
...some output truncated...
```

The `ssl_ciphers` parameter has no default string of cipher suites. For more information about ciphers, see the [ssl\\_ciphers](#) variable in the PostgreSQL documentation. For more information about cipher suite formats, see the [openssl-ciphers list format](#) and [openssl-ciphers string format](#) documentation on the OpenSSL website.

## 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 \(p. 1644\)](#). For more information about downloading certificates, see [Using SSL/TLS to encrypt a connection to a DB cluster \(p. 1642\)](#). For information about using SSL/TLS with PostgreSQL DB clusters, see [Securing Aurora PostgreSQL data with SSL/TLS \(p. 1029\)](#).

### Topics

- [Determining whether applications are connecting to Aurora PostgreSQL DB clusters using SSL \(p. 1033\)](#)
- [Determining whether a client requires certificate verification in order to connect \(p. 1033\)](#)

- [Updating your application trust store \(p. 1034\)](#)
- [Using SSL/TLS connections for different types of applications \(p. 1034\)](#)

## 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 parameter groups \(p. 215\)](#).

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.

For information about downloading the root certificate, see [Using SSL/TLS to encrypt a connection to a DB cluster \(p. 1642\)](#).

For sample scripts that import certificates, see [Sample script for importing certificates into your trust store \(p. 1651\)](#).

## 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 \(p. 1644\)](#).

## Using Kerberos authentication with Aurora PostgreSQL

You can use Kerberos 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 \(p. 1683\)](#).

### Topics

- [Region and version availability \(p. 1035\)](#)
- [Overview of Kerberos authentication for PostgreSQL DB clusters \(p. 1036\)](#)
- [Setting up Kerberos authentication for PostgreSQL DB clusters \(p. 1037\)](#)
- [Managing a DB cluster in a Domain \(p. 1046\)](#)
- [Connecting to PostgreSQL with Kerberos authentication \(p. 1047\)](#)

## Region and version availability

Kerberos authentication is supported on the following engine versions:

- All PostgreSQL 14 and 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 \(p. 1414\)](#).

Amazon Aurora supports Kerberos authentication for PostgreSQL DB clusters in the following AWS Regions:

Region name	Region
US East (Ohio)	us-east-2
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. 100)
  - [Modifying an Amazon Aurora DB cluster](#) (p. 248)
  - [Restoring from a DB cluster snapshot](#) (p. 375)
  - [Restoring a DB cluster to a specified time](#) (p. 415)

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. 1037)
- [Step 2: \(Optional\) create a trust for an on-premises Active Directory](#) (p. 1041)
- [Step 3: Create an IAM role for Amazon Aurora to access the AWS Directory Service](#) (p. 1042)
- [Step 4: Create and configure users](#) (p. 1043)
- [Step 5: Enable cross-VPC traffic between the directory and the DB instance](#) (p. 1043)
- [Step 6: Create or modify a PostgreSQL DB cluster](#) (p. 1044)
- [Step 7: Create Kerberos authentication PostgreSQL logins](#) (p. 1045)
- [Step 8: Configure a PostgreSQL client](#) (p. 1045)

## 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

1. In the [AWS Directory Service console](#) navigation pane, choose **Directories**, and then choose **Set up directory**.
2. Choose **AWS Managed Microsoft AD**. AWS Managed Microsoft AD is the only option currently supported for use with Amazon Aurora.
3. Choose **Next**.
4. 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 <a href="#">Learn more</a> 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'. At the bottom, there are tabs for 'Application management' (which is active), '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 \(p. 1042\)](#).

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 \(p. 1037\)](#)). 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](#).

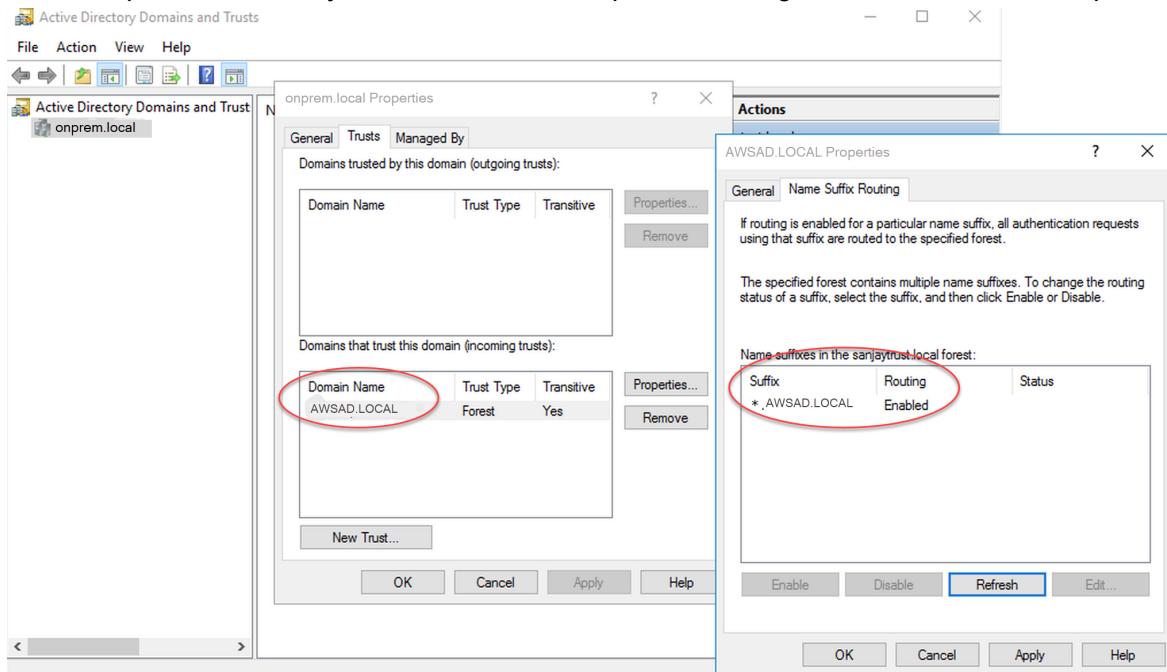
### Note

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 \(p. 1047\)](#).
- Windows clients can't connect with custom endpoints ([p. 38](#)).

- For [global databases \(p. 151\)](#):
  - 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 \(p. 1683\)](#).)

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 \(p. 1044\)](#).

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 \(p. 100\)](#).
- 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 \(p. 248\)](#).
- 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 \(p. 375\)](#).
- 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 \(p. 415\)](#).

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 \(p. 1683\)](#).

## 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 \(p. 212\)](#). For information about obtaining the endpoint, port number, and other details needed for connection, see [Viewing the endpoints for an Aurora cluster \(p. 37\)](#).

### 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 for the Writer instance of your Aurora PostgreSQL DB cluster. An endpoint looks similar to the following:

```
AUR-cluster-instance.111122223333.aws-region.rds.amazonaws.com
```

To connect to an on-premises Microsoft Active Directory from a Windows client, you use the domain name of the AWS Managed Active Directory instead of `rds.amazonaws.com` in the host endpoint. For example, suppose that the domain name for the AWS Managed Active Directory is `corp.example.com`. Then for **Host**, the endpoint would be specified as follows:

```
AUR-cluster-instance.111122223333.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 \(p. 1045\)](#).

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
```

## 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 \(p. 1049\)](#)

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 \(p. 1054\)](#)

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 data from Amazon S3 into Aurora PostgreSQL (p. 1230)**

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 \(p. 67\)](#).
- **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 about creating a VPC, see [Tutorial: Create a VPC for use with a DB cluster \(IPv4 only\) \(p. 1744\)](#).

Otherwise, you can choose to have Amazon RDS create a VPC for you by choosing **Create new VPC**.

- **DB 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 \(p. 207\)](#).

## 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 \(p. 56\)](#) and [Supported DB engines for DB instance classes \(p. 57\)](#).
- `--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, and it's available only if the Region offers a compatible version of Aurora PostgreSQL for your RDS for PostgreSQL DB instance. By *compatible*, we mean 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 \(p. 1054\)](#)
- [Preparing to migrate data by using an Aurora read replica \(p. 1054\)](#)
- [Creating an Aurora read replica \(p. 1055\)](#)
- [Promoting an Aurora read replica \(p. 1061\)](#)

## 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 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*.

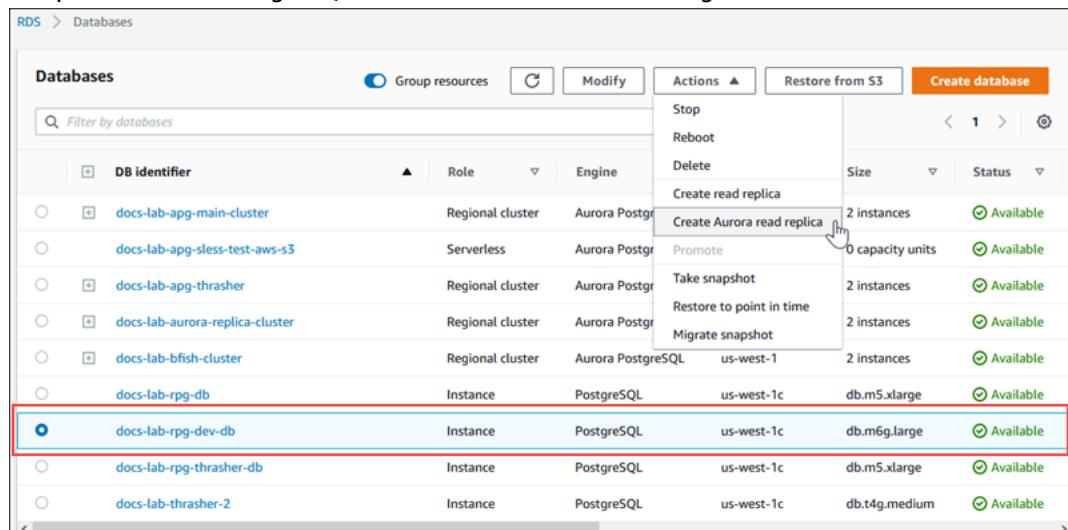
## Creating an Aurora read replica

You can create an Aurora read replica for an RDS for PostgreSQL DB instance by using the AWS Management Console or the AWS CLI. The option to create an Aurora read replica using the AWS Management Console 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
<b>DB instance class</b>	Choose a DB instance class that meets the processing and memory requirements primary instance in the DB cluster. For more information, see <a href="#">Aurora DB instance classes (p. 56)</a> .
<b>Multi-AZ deployment</b>	Not available during the migration
<b>DB instance identifier</b>	<p>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.</p> <p>The DB instance identifier has the following constraints:</p> <ul style="list-style-type: none"> <li>• It must contain 1–63 alphanumeric characters or hyphens.</li> <li>• Its first character must be a letter.</li> <li>• It can't end with a hyphen or contain two consecutive hyphens.</li> <li>• It must be unique for all DB instances for each AWS account, for each AWS Region.</li> </ul>
<b>Virtual Private Cloud (VPC)</b>	Choose the VPC to host the DB cluster. Choose <b>Create new VPC</b> to have Amazon RDS create a VPC for you. For more information, see <a href="#">DB cluster prerequisites (p. 127)</a> .
<b>DB subnet group</b>	Choose the DB subnet group to use for the DB cluster. Choose <b>Create new DB Subnet Group</b> to have Amazon RDS create a DB subnet group for you. For more information, see <a href="#">DB cluster prerequisites (p. 127)</a> .
<b>Public accessibility</b>	Choose <b>Yes</b> to give the DB cluster a public IP address; otherwise, choose <b>No</b> . 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 <a href="#">Hiding a DB cluster in a VPC from the internet (p. 1735)</a> .
<b>Availability zone</b>	Determine if you want to specify a particular Availability Zone. For more information about Availability Zones, see <a href="#">Regions and Availability Zones (p. 11)</a> .
<b>VPC security groups</b>	Choose one or more VPC security groups to secure network access to the DB cluster. Choose <b>Create new VPC security group</b> to have Amazon RDS create a VPC security group for you. For more information, see <a href="#">DB cluster prerequisites (p. 127)</a> .

Option	Description
<b>Database port</b>	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.
<b>DB parameter group</b>	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 <a href="#">Working with parameter groups (p. 215)</a> .
<b>DB cluster parameter group</b>	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 <a href="#">Working with parameter groups (p. 215)</a> .
<b>Encryption</b>	Choose <b>Enable encryption</b> for your new Aurora DB cluster to be encrypted at rest. If you choose <b>Enable encryption</b> , also choose a KMS key as the <b>AWS KMS key</b> value.
<b>Priority</b>	Choose a failover priority for the DB cluster. If you don't choose a value, the default is <b>tier-1</b> . This priority determines the order in which Aurora Replicas are promoted when recovering from a primary instance failure. For more information, see <a href="#">Fault tolerance for an Aurora DB cluster (p. 72)</a> .
<b>Backup retention period</b>	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.
<b>Enhanced monitoring</b>	Choose <b>Enable enhanced monitoring</b> to enable gathering metrics in real time for the operating system that your DB cluster runs on. For more information, see <a href="#">Monitoring OS metrics with Enhanced Monitoring (p. 518)</a> .
<b>Monitoring Role</b>	Only available if you chose <b>Enable enhanced monitoring</b> . The AWS Identity and Access Management (IAM) role to use for Enhanced Monitoring. For more information, see <a href="#">Setting up and enabling Enhanced Monitoring (p. 519)</a> .
<b>Granularity</b>	Only available if you chose <b>Enable enhanced monitoring</b> . Set the interval, in seconds, between when metrics are collected for your DB cluster.

Option	Description
<b>Auto minor version upgrade</b>	<p>Choose <b>Yes</b> to enable your Aurora PostgreSQL DB cluster to receive minor PostgreSQL DB engine version upgrades automatically when they become available.</p> <p>The <b>Auto minor version upgrade</b> 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>
<b>Maintenance window</b>	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 by using the AWS CLI, you first use the [create-db-cluster](#) CLI command to create an empty Aurora DB cluster. Once the DB cluster exists, you then use the [create-db-instance](#) command to create the primary instance for your DB cluster. The primary instance is the first instance that's created in an Aurora DB cluster. In this case, it's created initially as an Aurora read replica of your RDS for PostgreSQL DB instance. When the process concludes, your RDS for PostgreSQL DB instance has effectively been migrated to an Aurora PostgreSQL DB cluster.

You don't need to specify the main user account (typically, `postgres`), its password, or the database name. The Aurora read replica obtains these automatically from the source RDS for PostgreSQL DB instance that you identify when you invoke the AWS CLI commands.

You do need to specify the engine version to use for the Aurora PostgreSQL DB cluster and the DB instance. The version you specify should match the source RDS for PostgreSQL DB instance. If the source RDS for PostgreSQL DB instance is encrypted, you need to also specify encryption for the Aurora PostgreSQL DB cluster primary instance. Migrating an encrypted instance to an unencrypted Aurora DB cluster isn't supported.

The following examples create an Aurora PostgreSQL DB cluster named `my-new-aurora-cluster` that's going to use an unencrypted RDS DB source instance. You first create the Aurora PostgreSQL DB cluster by calling the [create-db-cluster](#) CLI command. The example shows how to use the optional `--storage-encrypted` parameter to specify that the DB cluster should be encrypted. Because the source DB isn't encrypted, the `--kms-key-id` is used to specify the key to use. For more information about required and optional parameters, see the list following the example.

For Linux, macOS, or Unix:

```
aws rds create-db-cluster \
--db-cluster-identifier my-new-aurora-cluster \
--db-subnet-group-name my-db-subnet \
--vpc-security-group-ids sg-11111111 \
--engine aurora-postgresql \
--engine-version same-as-your-rds-instance-version \
--replication-source-identifier arn:aws:rds:aws-region:111122223333:db/rpg-source-db \
--storage-encrypted \
--kms-key-id arn:aws:kms:aws-region:111122223333:key/11111111-2222-3333-444444444444
```

For Windows:

```
aws rds create-db-cluster ^
```

```
--db-cluster-identifier my-new-aurora-cluster ^
--db-subnet-group-name my-db-subnet ^
--vpc-security-group-ids sg-11111111 ^
--engine aurora-postgresql ^
--engine-version same-as-your-rds-instance-version ^
--replication-source-identifier arn:aws:rds:aws-region:111122223333:db/rpg-source-db ^
--storage-encrypted ^
--kms-key-id arn:aws:kms:aws-region:111122223333:key/11111111-2222-3333-444444444444
```

In the following list you can find more information about some of the options shown in the example. Unless otherwise specified, these parameters are required.

- **--db-cluster-identifier** – You need to give your new Aurora PostgreSQL DB cluster a name.
- **--db-subnet-group-name** – Create your Aurora PostgreSQL DB cluster in the same DB subnet as the source DB instance.
- **--vpc-security-group-ids** – Specify the security group for your Aurora PostgreSQL DB cluster.
- **--engine-version** – Specify the version to use for the Aurora PostgreSQL DB cluster. This should be the same as the version used by your source RDS for PostgreSQL DB instance.
- **--replication-source-identifier** – Identify your RDS for PostgreSQL DB instance using its Amazon Resource Name (ARN). For more information about Amazon RDS ARNs, see [Amazon Relational Database Service \(Amazon RDS\)](#) in the *AWS General Reference*. of your DB cluster.
- **--storage-encrypted** – Optional. Use only when needed to specify encryption as follows:
  - Use this parameter when the source DB instance has encrypted storage. The call to [create-db-cluster](#) fails if you don't use this parameter with a source DB instance that has encrypted storage. If you want to use a different key for the Aurora PostgreSQL DB cluster than the key used by the source DB instance, you need to also specify the **--kms-key-id**.
  - Use if the source DB instance's storage is unencrypted but you want the Aurora PostgreSQL DB cluster to use encryption. If so, you also need to identify the encryption key to use with the **--kms-key-id** parameter.
- **--kms-key-id** – Optional. When used, you can specify the key to use for storage encryption (**--storage-encrypted**) by using the key's ARN, ID, alias ARN, or its alias name. This parameter is needed only for the following situations:
  - To choose a different key for the Aurora PostgreSQL DB cluster than that used by the source DB instance.
  - To create an encrypted cluster from an unencrypted source. In this case, you need to specify the key that Aurora PostgreSQL should use for encryption.

After creating the Aurora PostgreSQL DB cluster, you then create the primary instance by using the [create-db-instance](#) CLI command, as shown in the following:

For Linux, macOS, or Unix:

```
aws rds create-db-instance \
--db-cluster-identifier my-new-aurora-cluster \
--db-instance-class db.x2g.16xlarge \
--db-instance-identifier rpg-for-migration \
--engine aurora-postgresql
```

For Windows:

```
aws rds create-db-instance ^
--db-cluster-identifier my-new-aurora-cluster ^
--db-instance-class db.x2g.16xlarge ^
--db-instance-identifier rpg-for-migration ^
--engine aurora-postgresql
```

In the following list, you can find more information about some of the options shown in the example.

- `--db-cluster-identifier` – Specify the name of the Aurora PostgreSQL DB cluster that you created with the [create-db-instance](#) command in the previous steps.
- `--db-instance-class` – The name of the DB instance class to use for your primary instance, such as `db.r4.xlarge`, `db.t4g.medium`, `db.x2g.16xlarge`, and so on. For a list of available DB instance classes, see [DB instance class types \(p. 56\)](#).
- `--db-instance-identifier` – Specify the name to give your primary instance.
- `--engine aurora-postgresql` – Specify `aurora-postgresql` for the engine.

## RDS API

To create an Aurora read replica from a source RDS for PostgreSQL DB instance, first use the RDS API operation [CreateDBCluster](#) to create a new Aurora DB cluster for the Aurora read replica that gets created from your source RDS for PostgreSQL DB instance. When the Aurora PostgreSQL DB cluster is available, you then use the [CreateDBInstance](#) to create the primary instance for the Aurora DB cluster.

You don't need to specify the main user account (typically, `postgres`), its password, or the database name. The Aurora read replica obtains these automatically from the source RDS for PostgreSQL DB instance specified with `ReplicationSourceIdentifier`.

You do need to specify the engine version to use for the Aurora PostgreSQL DB cluster and the DB instance. The version you specify should match the source RDS for PostgreSQL DB instance. If the source RDS for PostgreSQL DB instance is encrypted, you need to also specify encryption for the Aurora PostgreSQL DB cluster primary instance. Migrating an encrypted instance to an unencrypted Aurora DB cluster isn't supported.

To create the Aurora DB cluster for the Aurora read replica, 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*. If `ReplicationSourceIdentifier` identifies an encrypted source, Amazon RDS uses your default KMS key unless you specify a different key using the `KmsKeyId` option.
- `VpcSecurityGroupIds` – The list of Amazon EC2 VPC security groups to associate with this DB cluster.
- `StorageEncrypted` – Indicates that the DB cluster is encrypted. When you use this parameter without also specifying the `ReplicationSourceIdentifier`, Amazon RDS uses your default KMS key.
- `KmsKeyId` – The key for an encrypted cluster. When used, you can specify the key to use for storage encryption by using the key's ARN, ID, alias ARN, or its alias name.

For more information, see [CreateDBCluster](#) in the *Amazon RDS API Reference*.

Once the Aurora DB cluster is available, you can then create a primary instance for it 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.

For more information, see [CreateDBInstance](#) in the *Amazon RDS API Reference*.

## Promoting an Aurora read replica

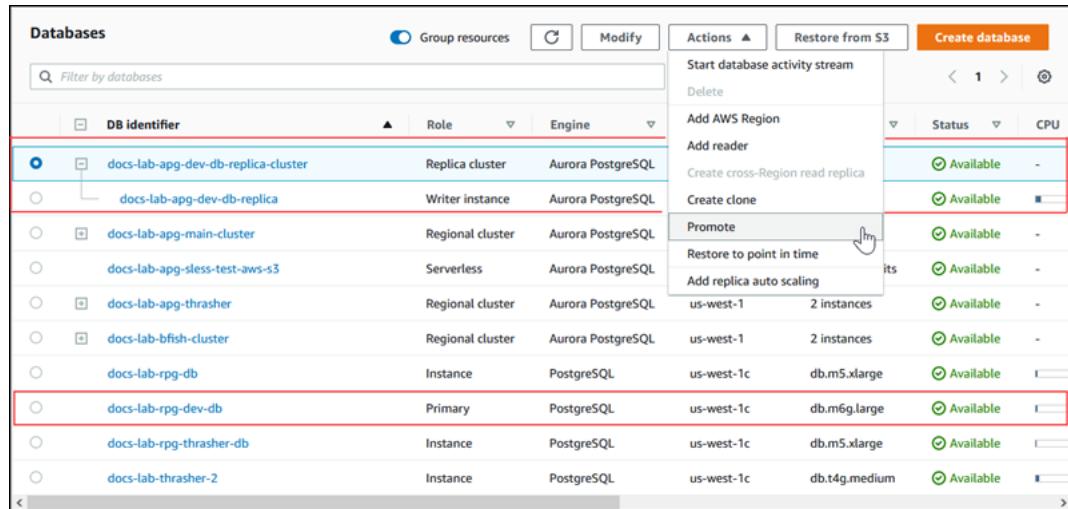
The migration to Aurora PostgreSQL isn't complete until you promote the Replica cluster, so don't delete the RDS for PostgreSQL source DB instance just yet.

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 \(p. 1224\)](#) and [Instance-level metrics for Amazon Aurora \(p. 531\)](#).

### 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.



4. 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

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, 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 \(p. 35\)](#). At this point, you can safely delete the DB instance.

# Using Babelfish for Aurora PostgreSQL

Babelfish for Aurora PostgreSQL extends your Aurora PostgreSQL DB cluster with the ability to accept database connections from SQL Server clients. With Babelfish, applications that were originally built for SQL Server can 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 [Migrating a SQL Server database to Babelfish for Aurora PostgreSQL \(p. 1093\)](#).

**Note**

Babelfish for Aurora PostgreSQL is available with Aurora PostgreSQL 13.4 and higher versions.

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. Client applications that use the Tabular Data Stream (TDS) wire protocol can connect natively to the TDS listener port on Aurora PostgreSQL. To learn more about TDS, see [\[MS-TDS\]: Tabular Data Stream Protocol](#) on the Microsoft website.

**Note**

Babelfish for Aurora PostgreSQL supports TDS versions 7.1 through 7.4.

Babelfish also provides access to data using the native PostgreSQL connection. By default, both SQL dialects supported by Babelfish are available through their native wire protocols at the following ports:

- SQL Server dialect (T-SQL), clients connect to port 1433.
- PostgreSQL dialect (PL/pgSQL), clients connect to port 5432.

Babelfish runs the Transact-SQL (T-SQL) language with some differences. For more information, see [Differences between Babelfish for Aurora PostgreSQL and SQL Server \(p. 1108\)](#).

In the following sections, you can find information about setting up and using a Babelfish for Aurora PostgreSQL DB cluster.

## Topics

- [Understanding Babelfish architecture and configuration \(p. 1064\)](#)
- [Creating a Babelfish for Aurora PostgreSQL DB cluster \(p. 1086\)](#)
- [Migrating a SQL Server database to Babelfish for Aurora PostgreSQL \(p. 1093\)](#)
- [Connecting to a Babelfish DB cluster \(p. 1097\)](#)
- [Working with Babelfish \(p. 1106\)](#)
- [Troubleshooting Babelfish \(p. 1123\)](#)
- [Turning off Babelfish \(p. 1125\)](#)
- [Babelfish version updates \(p. 1126\)](#)
- [Babelfish for Aurora PostgreSQL reference \(p. 1128\)](#)

# Understanding Babelfish architecture and configuration

You manage the Aurora PostgreSQL-Compatible Edition DB cluster running Babelfish much as you would any Aurora DB cluster. That is, you benefit from the scalability, high-availability with failover support, and built-in replication provided by an Aurora DB cluster. To learn more about these capabilities, see [Managing performance and scaling for Aurora DB clusters \(p. 274\)](#), [High availability for Amazon Aurora \(p. 71\)](#), and [Replication with Amazon Aurora \(p. 73\)](#). You also have access to many other AWS tools and utilities, including the following:

- Amazon CloudWatch is a monitoring and observability service that provides you with data and actionable insights. For more information, see [Monitoring Amazon Aurora metrics with Amazon CloudWatch \(p. 452\)](#).
- Performance Insights is a database performance tuning and monitoring feature that helps you quickly assess the load on your database. To learn more, see [Monitoring DB load with Performance Insights on Amazon Aurora \(p. 461\)](#).
- 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. For more information, see [Using Amazon Aurora global databases \(p. 151\)](#).
- Automatic software patching keeps your database up-to-date with the latest security and feature patches when they become available.
- Amazon RDS events notify you by email or SMS message of important database events, such as an automated failover. For more information, see [Monitoring Amazon Aurora events \(p. 568\)](#).

However, not all Aurora features are supported. Currently, Babelfish doesn't support the following:

- AWS Directory Service
- AWS Identity and Access Management
- Database activity streams (DAS)
- Kerberos
- Query plan management (QPM)
- SCRAM (salted challenge response authentication mechanism)
- Zero-downtime patching (ZDP)

Currently, Babelfish isn't a HIPAA eligible service listed in the AWS [HIPAA Eligible Services Reference](#). To learn more, see [Amazon Aurora PostgreSQL in Architecting for HIPAA Security and Compliance on Amazon Web Services](#).

Following, you can learn about Babelfish architecture and how the SQL Server databases that you migrate are handled by Babelfish. When you create your Babelfish DB cluster, you need to make some decisions up front about single database or multiple databases, collations, and other details.

## Topics

- [Babelfish architecture \(p. 1065\)](#)
- [DB cluster parameter group settings for Babelfish \(p. 1068\)](#)
- [Collations supported by Babelfish \(p. 1073\)](#)
- [Managing Babelfish error handling with escape hatches \(p. 1081\)](#)

## 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

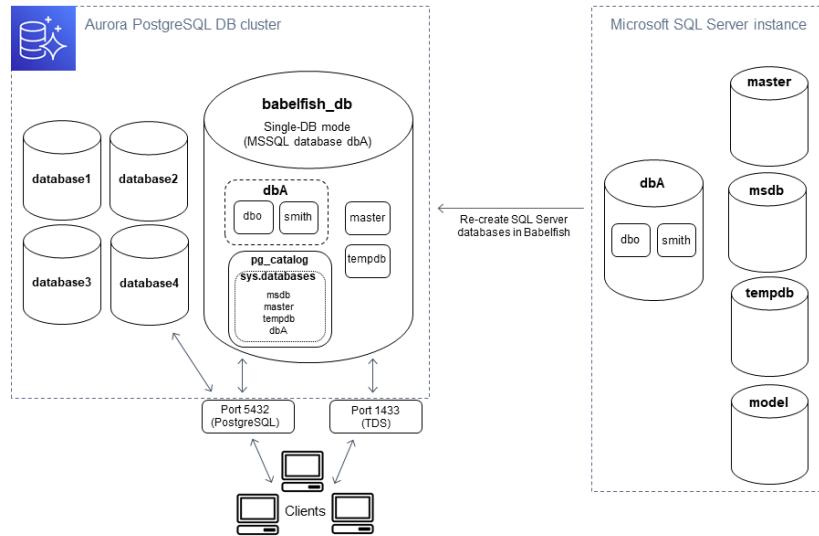
In an Aurora PostgreSQL cluster, the `babelfish_db` database name is reserved for Babelfish. Creating your own "babelfish\_db" database on a Babelfish DB cluster prevents Aurora from successfully provisioning Babelfish.

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`, `msdb`, 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

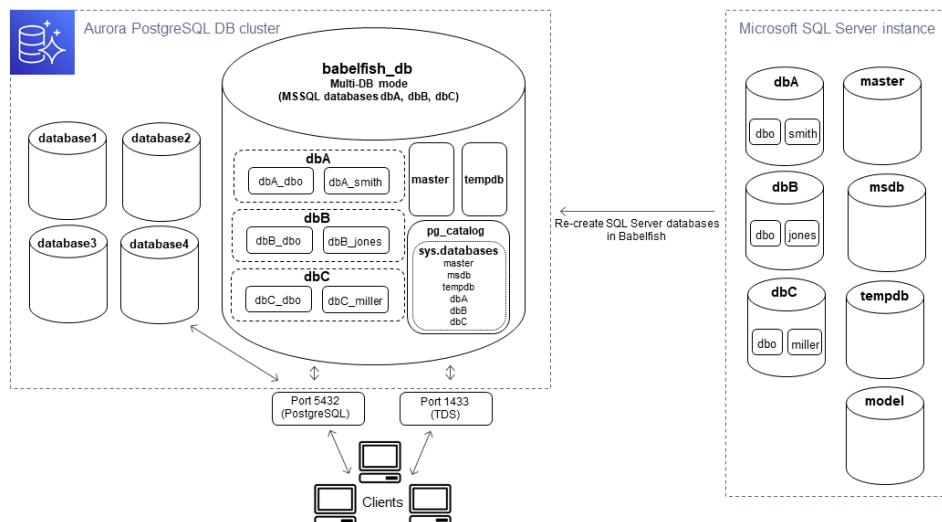
When you create an Aurora PostgreSQL cluster to use with Babelfish, you choose between using a single SQL Server database on its own or multiple SQL Server databases together. Your choice affects how the names of SQL Server schemas inside the `babelfish_db` database appear from Aurora PostgreSQL. The 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.



As shown in the image, multiple-database mode and single-database mode are the same as SQL Server when connecting through the TDS port and using T-SQL. For example, `USE dbA` lists schemas `dbo` and `smith` just as it does 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.

## DB cluster parameter group settings for Babelfish

When you create an Aurora PostgreSQL DB cluster and choose **Turn on Babelfish**, a DB cluster parameter group is created for you automatically if you choose **Create new**. This DB cluster parameter group is based on the Aurora PostgreSQL DB cluster parameter group for the Aurora PostgreSQL version chosen for the install, for example, Aurora PostgreSQL version 14. It's named using the following general pattern:

custom-aurora-postgresql14-babelfish-compat-3

You can change the following settings during the cluster creation process but some of these can't be changed once they're stored in the custom parameter group, so choose carefully:

- Single database or Multiple databases
- Default collation locale
- Collation name
- Babelfish TDS port
- DB parameter group

To use an existing Aurora PostgreSQL DB cluster version 13 or higher parameter group, edit the group and set the `babelfish_status` parameter to `on`. Specify any Babelfish options before creating your Aurora PostgreSQL cluster. To learn more, see [Working with parameter groups \(p. 215\)](#).

The following parameters control Babelfish preferences. Unless otherwise stated in the Description, parameters are modifiable. The default value is included in the description. To see the allowable values for any parameter, do as follows:

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** from the navigation menu.
3. Choose the `default.aurora-postgresql14` DB cluster parameter group from the list.
4. Enter the name of a parameter in the search field. For example, enter `babelfishpg_tsql.default_locale` in the search field to display this parameter and its default value and allowable settings.

Parameter	Description
<code>babelfishpg_tds.tds_default_numeric_scale</code>	Sets the default scale of numeric type to be sent in the TDS column metadata if the engine doesn't specify one. (Default: 8) (Allowable: 0–38)
<code>babelfishpg_tds.tds_default_numeric_precision</code>	An integer that sets the default precision of numeric type to be sent in the TDS column metadata if the engine doesn't specify one. (Default: 38) (Allowable: 1–38)
<code>babelfishpg_tds.tds_default_packet_size</code>	An integer that sets the default packet size for connecting SQL Server clients. (Default: 4096) (Allowable: 512–32767)
<code>babelfishpg_tds.tds_default_protocol_version</code>	An integer that sets a default TDS protocol version for connecting clients. (Default: DEFAULT) (Allowable: TDSv7.0, TDSv7.1, TDSv7.1.1, TDSv7.2, TDSv7.3A, TDSv7.3B, TDSv7.4, DEFAULT)

Parameter	Description
babelfishpg_tds.default_server_name	A string that identifies the default name of the Babelfish server. (Default: Microsoft SQL Server) (Allowable: null)
babelfishpg_tds.enable_tds_debug_log_level	An integer that sets the logging level in TDS; 0 turns off logging. (Default: 1) (Allowable: 0, 1, 2, 3)
babelfishpg_tds.listen_addresses	A string that sets the host name or IP address or addresses to listen for TDS on. This parameter can't be modified after the Babelfish DB cluster is created. (Default: * ) (Allowable: null)
babelfishpg_tds.port	An integer that specifies the TCP port used for requests in SQL Server syntax. (Default: 1433) (Allowable: 1–65535)
babelfishpg_tds.tds_ssl_encrypt	A boolean that turns encryption on (0) or off (1) for data traversing the TDS listener port. For detailed information about using SSL for client connections, see <a href="#">Babelfish SSL settings and client connections (p. 1071)</a> . (Default: 0) (Allowable: 0, 1)
babelfishpg_tds.tds_ssl_max_protocol_version	A string that specifies the highest SSL/TLS protocol version to use for the TDS session. (Default: 'TLSv1.2') (Allowable: 'TLSv1', 'TLSv1.1', 'TLSv1.2')
babelfishpg_tds.tds_ssl_min_protocol_version	A string that specifies the minimum SSL/TLS protocol version to use for the TDS session. (Default: 'TLSv1') (Allowable: 'TLSv1', 'TLSv1.1', 'TLSv1.2')
babelfishpg_tds.unix_socket_directories	A string that identifies the TDS server Unix socket directory. This parameter can't be modified after the Babelfish DB cluster is created. (Default: /tmp) (Allowable: null)
babelfishpg_tds.unix_socket_group	A string that identifies the TDS server Unix socket group. This parameter can't be modified after the Babelfish DB cluster is created. (Default: rdsdb) (Allowable: null)
babelfishpg_tsql.default_locale	A string that specifies the default locale used for Babelfish collations. The default locale is the locale only and doesn't include any qualifiers.  Set this parameter when you provision a Babelfish DB cluster. After the DB cluster is provisioned, changes to this parameter are ignored. (Default: en_US) (Allowable: See <a href="#">tables (p. 1073)</a> )
babelfishpg_tsql.migration_mode	A non-modifiable list that specifies support for single- or multiple user databases. Set this parameter when you provision a Babelfish DB cluster. After the DB cluster is provisioned, you can't modify this parameter's value. (Default: single-db) (Allowable: single-db, multi-db,null)

Parameter	Description
<code>babelfishpg_tsql.server_collation_name</code>	A string that specifies the name of the collation used for server-level actions. Set this parameter when you provision a Babelfish DB cluster. After the DB cluster is provisioned, don't modify the value of this parameter. (Default: <code>bbf_unicode_general_ci_as</code> ) (Allowable: See <a href="#">tables (p. 1073)</a> )
<code>babelfishpg_tsql.version</code>	A string that sets the output of <code>@@VERSION</code> variable. Don't modify this value for Aurora PostgreSQL DB clusters. (Default: <code>null</code> ) (Allowable: default)
<code>rds.babelfish_status</code>	A string that sets the state of Babelfish functionality. When this parameter is set to <code>datatypesonly</code> , Babelfish is turned off but SQL Server data types are still available. (Default: <code>off</code> ) (Allowable: <code>on</code> , <code>off</code> , <code>datatypesonly</code> )
<code>unix_socket_permissions</code>	An integer that sets the TDS server Unix socket permissions. This parameter can't be modified after the Babelfish DB cluster is created. (Default: <code>0700</code> ) (Allowable: <code>0–511</code> )

## Babelfish SSL settings and client connections

When a client connects to the TDS port (default 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

## Client authentication to Babelfish

Babelfish for Aurora PostgreSQL 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 \(p. 1018\)](#).

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 that you do the following:

- Require a complex password that's at least eight (8) characters long.
- Enforce a password expiration policy.

To review a complete list of database users, use the command `SELECT * FROM pg_user;`.

## Collations supported by Babelfish

When you create an Aurora PostgreSQL DB cluster with Babelfish, you choose a collation for your data. A *collation* specifies the sort order and bit patterns that produce the text or characters in a given written human language. A collation includes rules comparing data for a given set of bit patterns. Collation is related to localization. Different locales affect character mapping, sort order, and the like. Collation attributes are reflected in the names of various collations. For information about attributes, see the [Babelfish collation attributes table](#).

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 supports the collations listed in the [Babelfish supported collations table](#). Babelfish maps SQL Server collations to comparable collations provided by Babelfish.

Babelfish uses version 153.80 of the International Components for Unicode (ICU) collation library. For more information about ICU collations, see [Collation](#) in the ICU documentation. To learn more about PostgreSQL and collation, see [Collation Support](#) in the the PostgreSQL documentation.

### Topics

- [DB cluster parameters that control collation and locale \(p. 1073\)](#)
- [Deterministic and nondeterministic collations and Babelfish \(p. 1074\)](#)
- [Collations supported by Babelfish \(p. 1074\)](#)
- [Managing collations \(p. 1077\)](#)
- [Collation limitations and behavior differences \(p. 1078\)](#)

## DB cluster parameters that control collation and locale

The following parameters affect collation behavior.

### **`babelfishpg_tsql.default_locale`**

This parameter specifies the default locale used by the collation. This parameter is used in combination with attributes listed in the [Babelfish collation attributes table](#) to customize collations for a specific language and region. The default value for this parameter is `en-US`.

The default locale applies to all Babelfish collation names that start with "BBF" and to all SQL Server collations that are mapped to Babelfish collations. Changing the setting for this parameter on an existing Babelfish DB cluster doesn't affect the locale of existing collations. For the list of collations, see the [Babelfish supported collations table](#).

### **`babelfishpg_tsql.server_collation_name`**

This parameter specifies the default collation for the server (Aurora PostgreSQL DB cluster instance) and the database. The default value is `sql_latin1_general_cp1_ci_as`. The `server_collation_name` has to be a `CI_AS` collation because in T-SQL, the server collation determines how identifiers are compared.

When you create your Babelfish DB cluster, you choose the **Collation name** from the selectable list. These include the collations listed in the [Babelfish supported collations table](#). Don't modify the `server_collation_name` after the Babelfish database is created.

The settings you choose when you create your Babelfish for Aurora PostgreSQL DB cluster are stored in the DB cluster parameter group associated with the cluster for these parameters and set its collation behavior.

## Deterministic and nondeterministic collations and Babelfish

Babelfish supports deterministic and nondeterministic collations:

- A *deterministic collation* evaluates characters that have identical byte sequences as equal. That means that `x` and `X` aren't equal in a deterministic collation. Deterministic collations can be case-sensitive (CS) and accent-sensitive (AS).
- A *nondeterministic collation* doesn't need an identical match. A nondeterministic collation evaluates `x` and `X` as equal. Nondeterministic collations are case-insensitive (CI) and accent-insensitive (AI).

In the table following, you can find some behavior differences between Babelfish and PostgreSQL when using nondeterministic collations.

Babelfish	PostgreSQL
Supports the LIKE clause for CI_AS collations.	Doesn't support the LIKE clause on nondeterministic collations.
Doesn't support the LIKE clause on AI collations.	
Don't support pattern-matching operations on nondeterministic collations.	

For a list of other limitations and behavior differences for Babelfish compared to SQL Server and PostgreSQL, see [Collation limitations and behavior differences \(p. 1078\)](#).

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.

## Collations supported by Babelfish

Use the following collations as a server collation or an object collation.

Collation ID	Notes
bbf_unicode_general_ci_as	Supports case-insensitive comparison and the LIKE operator.
bbf_unicode_cp1_ci_as	<a href="#">Nondeterministic collation</a> also known as CP1252.
bbf_unicode_CP1250_ci_as	<a href="#">Nondeterministic collation</a> used to represent texts in Central European and Eastern European languages that use Latin script.
bbf_unicode_CP1251_ci_as	<a href="#">Nondeterministic collation</a> for languages that use the Cyrillic script.
bbf_unicode_cp1253_ci_as	<a href="#">Nondeterministic collation</a> used to represent modern Greek.
bbf_unicode_cp1254_ci_as	<a href="#">Nondeterministic collation</a> that supports Turkish.
bbf_unicode_cp1255_ci_as	<a href="#">Nondeterministic collation</a> that supports Hebrew.
bbf_unicode_cp1256_ci_as	<a href="#">Nondeterministic collation</a> used to write languages that use Arabic script.
bbf_unicode_cp1257_ci_as	<a href="#">Nondeterministic collation</a> used to support Estonian, Latvian, and Lithuanian languages.
bbf_unicode_cp1258_ci_as	<a href="#">Nondeterministic collation</a> used to write Vietnamese characters.
bbf_unicode_cp874_ci_as	<a href="#">Nondeterministic collation</a> used to write Thai characters.
sql_latin1_general_cp1250_ci_as	<a href="#">Nondeterministic single-byte character encoding</a> used to represent Latin characters.
sql_latin1_general_cp1251_ci_as	<a href="#">Nondeterministic collation</a> that supports Latin characters.
sql_latin1_general_cp1_ci_as	<a href="#">Nondeterministic collation</a> that supports Latin characters.
sql_latin1_general_cp1253_ci_as	<a href="#">Nondeterministic collation</a> that supports Latin characters.
sql_latin1_general_cp1254_ci_as	<a href="#">Nondeterministic collation</a> that supports Latin characters.
sql_latin1_general_cp1255_ci_as	<a href="#">Nondeterministic collation</a> that supports Latin characters.
sql_latin1_general_cp1256_ci_as	<a href="#">Nondeterministic collation</a> that supports Latin characters.
sql_latin1_general_cp1257_ci_as	<a href="#">Nondeterministic collation</a> that supports Latin characters.
sql_latin1_general_cp1258_ci_as	<a href="#">Nondeterministic collation</a> 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.
japanese_ci_as	Nondeterministic collation that supports Japanese. Supported in Babelfish 2.1.0 and higher releases.
korean_wansung_ci_as	Nondeterministic collation that supports Korean (with dictionary sort).
latin1_general_ci_as	Nondeterministic collation that supports Latin characters.

<b>Collation ID</b>	<b>Notes</b>
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.

<b>Dialect</b>	<b>Deterministic options</b>	<b>Nondeterministic options</b>
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
Japanese (Babelfish 2.1.0 and higher)	Japanese_CS_AS	Japanese_CI_AI, Japanese_CI_AS
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

Dialect	Deterministic options	Nondeterministic options
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. To see if your current database has collations that need refreshing, you can use the following query after connecting using `psql` or `pgAdmin`:

```
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;
```

This query returns output such as the following:

Collation   Object
(0 rows)

In this example, no collations need to be updated.

To get a listing of the predefined collations in your Babelfish database, you can use `psql` or `pgAdmin` with the following query:

```
SELECT * FROM pg_collation;
```

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 get a listing of all collations by using `sqlcmd`, connect to the T-SQL port (1433, by default) and run the following query:

```
1> :setvar SQLCMDMAXVARTYPEWIDTH 40
2> :setvar SQLCMDMAXFIXEDTYPEWIDTH 40
3> SELECT * FROM fn_helpcollations()
4> GO
name                                description
-----                                -----
arabic_cs_as                          Arabic, case-sensitive, accent-sensitive
arabic_ci_ai                          Arabic, case-insensitive, accent-insensi
arabic_ci_as                          Arabic, case-insensitive, accent-sensiti
bbf_unicode_bin2                      Unicode-General, case-sensitive, accent-
bbf_unicode_cp1250_ci_ai              Default locale, code page 1250, case-ins
bbf_unicode_cp1250_ci_as              Default locale, code page 1250, case-ins
bbf_unicode_cp1250_cs_ai              Default locale, code page 1250, case-sen
bbf_unicode_cp1250_cs_as              Default locale, code page 1250, case-sen
bbf_unicode_pref_cp1250_cs_as        Default locale, code page 1250, case-sen
bbf_unicode_cp1251_ci_ai              Default locale, code page 1251, case-ins
bbf_unicode_cp1251_ci_as              Default locale, code page 1251, case-ins
bbf_unicode_cp1254_ci_ai              Default locale, code page 1254, case-ins
...
(124 rows affected)
```

Lines 1 and 2 shown in the example narrow the output for documentation readability purposes only.

```

1> SELECT SERVERPROPERTY('COLLATION')
2> GO
serverproperty
-----
sql_latin1_general_cp1_ci_as

(1 rows affected)
1>

```

## Collation limitations and behavior differences

Babelfish uses the ICU library for collation support. 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. In the following list you can find known limitations and behavior variations of Babelfish collations:

- **Indexes and collation type dependency** – 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 library version changes.
- **COLLATIONPROPERTY function** – Collation properties are implemented only for the supported Babelfish BBF collations. For more information, see the [Babelfish supported collations table](#).
- **Unicode sorting rule differences** – SQL collations for SQL Server sort Unicode-encoded data (`nchar` and `nvarchar`) differently than data that's not Unicode-encoded (`char` and `varchar`). Babelfish databases are always UTF-8 encoded and always apply Unicode sorting rules consistently, regardless of data type, so the sort order for `char` or `varchar` is the same as it is for `nchar` or `nvarchar`.
- **Secondary-equal collations and sorting behavior** – 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, workaround for ORDER BY** – SQL collations, such as `SQL_Latin1_General_Pref_CP1_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 such as with `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_CP1_CI_AS` collation to use the `BBF_SQL_Latin1_General_CP1_CI_AS` collation first. Then add `COLLATE BBF_SQL_Latin1_General_Pref_CP1_CS_AS` to any `ORDER BY` clause for this column.

- **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 for accent-insensitive collations only.

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 SQL collations 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 all characters are converted to UTF-8 and sorted 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 (KS) collations** – A Kana-sensitive (KS) collation is one that treats Hiragana and Katakana Japanese Kana characters differently. 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 (WS) collations** – 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 sensitive (VSS) collations** – Variation-selector sensitive (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 collations** – 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.

- **Non-deterministic collations and CHARINDEX limitation** – For Babelfish releases older than version 2.1.0, you can't use CHARINDEX with non-deterministic collations. By default, Babelfish uses a case-insensitive (non-deterministic) collation. Using CHARINDEX for older versions of Babelfish raises the following runtime error:

```
nondeterministic collations are not supported for substring searches
```

#### Note

This limitation and workaround apply to Babelfish version 1.x only (Aurora PostgreSQL 13.4 through 13.7). Babelfish 2.1.0 and higher releases don't have this issue.

You can work around this issue in one of the following ways:

- Explicitly convert the expression to a case-sensitive collation and case-fold both arguments by applying LOWER or UPPER. For example, `SELECT charindex('x', a) FROM t1` would become the following:

```
SELECT charindex(LOWER('x'), LOWER(a COLLATE sql_latin1_general_cp1_cs_as)) FROM t1
```

- Create a SQL function `f_charindex`, and replace CHARINDEX calls with calls to the following function:

```
CREATE function f_charindex(@s1 varchar(max), @s2 varchar(max)) RETURNS int
AS
BEGIN
declare @i int = 1
WHILE len(@s2) >= len(@s1)
BEGIN
    if LOWER(@s1) = LOWER(substring(@s2,1,len(@s1))) return @i
    set @i += 1
    set @s2 = substring(@s2,2,999999999)
END
return 0
END
go
```

## Managing Babelfish error handling with escape hatches

Babelfish mimics SQL behavior for control flow and transaction state whenever possible. When Babelfish encounters an error, it returns an error code similar to the SQL Server error code. If Babelfish can't map the error to a SQL Server code, it returns a fixed error code (33557097) and takes specific actions based on the type of error, as follows:

- For compile time errors, Babelfish rolls back the transaction.
- For runtime errors, Babelfish ends the batch and rolls back the transaction.
- For protocol error between 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](#).

### Modifying Babelfish escape hatch settings

To handle 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.

To apply changes to the current session and on the cluster level, include the `server` keyword.

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 make the settings permanent on a cluster-wide level, include the `server` keyword, such as shown in the following:

```
EXECUTE sp_babelfish_configure 'escape_hatch_unique_constraint', 'ignore', 'server'
```

To set them for the current session only, don't use `server`.

- To reset all escape hatches to their default values, run `sp_babelfish_configure 'default'` (Babelfish 1.2.0 and higher).

The string identifying the hatch (or hatches) can include SQL wildcards. For example, the following sets all syntax escape hatches to `ignore` for the Aurora PostgreSQL cluster.

```
EXECUTE sp_babelfish_configure '%', 'ignore', 'server'
```

In the following table you can find descriptions and default values for the Babelfish predefined escape hatches.

<b>Escape hatch</b>	<b>Description</b>	<b>Default</b>
<code>escape_hatch_constraint_name_for_default</code>	Controls Babelfish behavior related to default constraint names.	ignore
<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_for_replication</code>	Controls Babelfish behavior related to the [NOT] FOR REPLICATION clause when creating or altering a table.	strict
<code>escape_hatch_fulltext</code>	Controls Babelfish behavior related to FULLTEXT features, such a DEFAULT_FULLTEXT_LANGUAGE in CREATE/ALTER DATABASE, CREATE FULLTEXT INDEX, or sp_fulltext_database.	strict
<code>escape_hatch_ignore_dup_key</code>	Controls Babelfish behavior related to CREATE/ALTER TABLE and CREATE INDEX. When IGNORE_DUP_KEY=ON, raises an error when set to strict (the default) or ignores the error when set to ignore (Babelfish version 1.2.0 and higher).	strict
<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_join_hints</code>	Controls the behavior of keywords in a JOIN operator: LOOP, HASH, MERGE, REMOTE, REDUCE, REDISTRIBUTE, REPLICATE.	ignore
<code>escape_hatch_language_non_english</code>	Controls Babelfish behavior related to languages other than English for onscreen messages. Babelfish currently supports only us_english	strict

<b>Escape hatch</b>	<b>Description</b>	<b>Default</b>
	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.	
<code>escape_hatch_login_hashed_password</code>	When ignored, suppresses the error for the HASHED keyword for CREATE LOGIN and ALTER LOGIN.	strict
<code>escape_hatch_login_misc_options</code>	When ignored, suppresses the error for other keywords besides HASHED, MUST_CHANGE, OLD_PASSWORD, and UNLOCK for CREATE LOGIN and ALTER LOGIN.	strict
<code>escape_hatch_login_old_password</code>	When ignored, suppresses the error for the OLD_PASSWORD keyword for CREATE LOGIN and ALTER LOGIN.	strict
<code>escape_hatch_login_password_must_change</code>	When ignored, suppresses the error for the MUST_CHANGE keyword for CREATE LOGIN and ALTER LOGIN.	strict
<code>escape_hatch_login_password_unlock</code>	When ignored, suppresses the error for the UNLOCK keyword for CREATE LOGIN and ALTER LOGIN.	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_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 SELECT ... OPTION(MERGE JOIN HASH, MAXRECURSION 10)).	ignore
<code>escape_hatch_rowversion</code>	Controls the behavior of the ROWVERSION and TIMESTAMP datatypes. For usage information, see <a href="#">Using Babelfish features with limited implementation (p. 1113)</a> .	strict
<code>escape_hatch_schemabinding_function</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 FUNCTION command.	ignore

<b>Escape hatch</b>	<b>Description</b>	<b>Default</b>
<code>escape_hatch_schemabinding_procedure</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 PROCEDURE command.	ignore
<code>escape_hatch_rowguidcol_column</code>	Controls Babelfish behavior related to the ROWGUIDCOL clause when creating or altering a table.	strict
<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_schemabinding_view</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 VIEW command.	ignore
<code>escape_hatch_session_settings</code>	Controls Babelfish behavior toward unsupported session-level SET statements.	ignore
<code>escape_hatch_storage_on_partition</code>	Controls Babelfish behavior related to the ON partition_scheme column clause when defining partitioning. Babelfish currently doesn't implement partitioning.	strict
<code>escape_hatch_storage_options</code>	Escape hatch on any storage option used in CREATE, 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).	ignore
<code>escape_hatch_table_hints</code>	Controls the behavior of table hints specified using the WITH (...) clause.	ignore

Escape hatch	Description	Default
escape_hatch_unique_constraint	When set to strict, an obscure semantic difference between SQL Server and PostgreSQL in handling NULL values on indexed columns can raise errors. The semantic difference only emerges in unrealistic use cases, so you can set this escape hatch to 'ignore' to avoid seeing the error.	strict

# Creating a Babelfish for Aurora PostgreSQL DB cluster

Babelfish for Aurora PostgreSQL is 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.

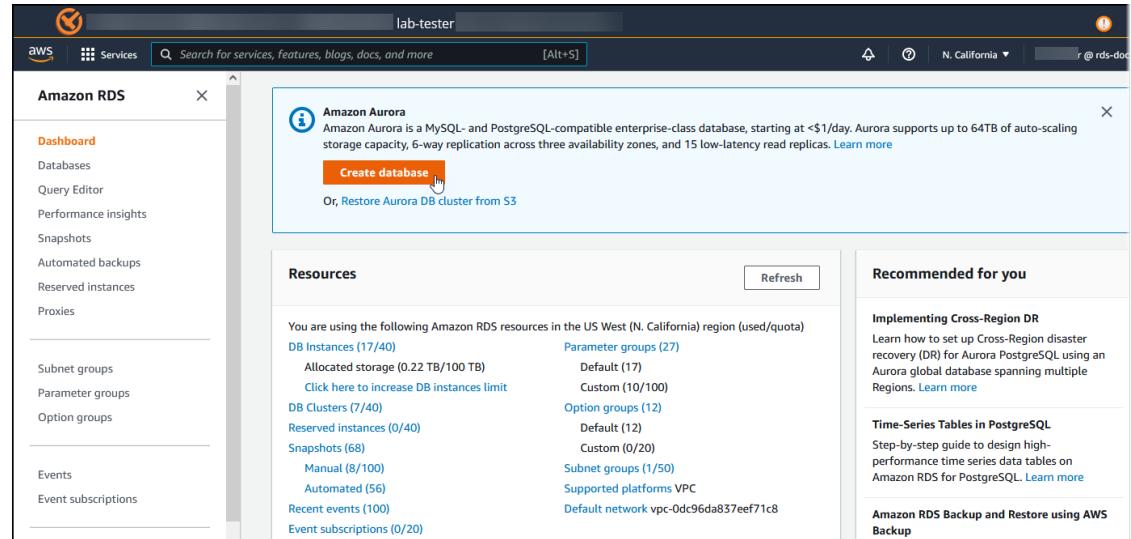
## Note

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**.



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 versions.
6. For **Available versions**, choose an Aurora PostgreSQL version. To get the latest Babelfish features, choose the highest Aurora PostgreSQL major 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 Serverless v2  
Offers instance scaling for even the most demanding workloads.

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 (3/22) [Info](#)  
Aurora PostgreSQL (Compatible with PostgreSQL 13.6)

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. The default value for Aurora PostgreSQL is `postgres`. You can accept the default or choose a different name. For example, to follow the naming convention used on your SQL Server databases, you can enter `sa` (system administrator) for the 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 the `sysadmin` group (role) for the cluster.
10. For **Master password**, create a strong password 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.

**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 [DB cluster parameter group settings for Babelfish \(p. 1068\)](#).

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 \(p. 1065\)](#).

## ▼ 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 [Collations supported by Babelfish \(p. 1073\)](#).
16. For **Collation name** field, enter your default collation. The default is sql\_latin1\_general\_cp1\_ci\_as. For detailed information, see [Collations supported by Babelfish \(p. 1073\)](#).
17. For **Babelfish TDS port**, enter the default port 1433 unless you specified a different port for your Babelfish DB cluster.
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 \(p. 72\)](#).

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.

Databases									<input checked="" type="checkbox"/> Group resources	<a href="#">Modify</a>	<a href="#">Actions</a> ▾	<a href="#">Restore from S3</a>	<a href="#">Create database</a>
<a href="#">Filter databases</a>		DB Identifier	Role	Engine	Region & AZ	Size	Status	CPU	Current activity	Maintenance			
<input type="radio"/>	babelfish-workshop	Regional cluster	Aurora PostgreSQL	us-west-2	1 instance	<span>Available</span>	-	-	-	none			
<input type="radio"/>	babelfish-workshop-instance-1	Writer Instance	Aurora PostgreSQL	-	db.r6g.large	<span>Creating</span>	-	<span>0 Sessions</span>	-	none			

## AWS CLI

When you create an Babelfish for Aurora PostgreSQL; using the AWS CLI, you need to pass the command the name of the DB cluster parameter group to use for the cluster. For more information, see [DB cluster prerequisites \(p. 127\)](#).

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 parameter groups \(p. 215\)](#).
- Modify the parameter group, adding the parameter that turns on Babelfish.

## To create an Aurora PostgreSQL DB cluster with Babelfish using the AWS CLI

The examples that follow use the default Master username, `postgres`. Replace as needed with the username that you created for your DB cluster, such as `sa` or whatever username you chose if you didn't accept the default.

1. Create a parameter group.

For Linux, macOS, or Unix:

```
aws rds create-db-cluster-parameter-group \
--endpoint-url endpoint-url \
--db-cluster-parameter-group-name parameter-group \
--db-parameter-group-family aurora-postgresql14 \
--description "description"
```

For Windows:

```
aws rds create-db-cluster-parameter-group ^
--endpoint-url endpoint-URL ^
--db-cluster-parameter-group-name parameter-group ^
--db-parameter-group-family aurora-postgresql14 ^
--description "description"
```

2. Modify your parameter group to turn on Babelfish.

For Linux, macOS, or Unix:

```
aws rds modify-db-cluster-parameter-group \
--endpoint-url endpoint-url \
--db-cluster-parameter-group-name parameter-group \
--parameters "ParameterName=rds.babelfish_status,ParameterValue=on,ApplyMethod=pending-reboot"
```

For Windows:

```
aws rds modify-db-cluster-parameter-group ^
--endpoint-url endpoint-url ^
--db-cluster-parameter-group-name paramater-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 cluster-name \
--master-username postgres \
--master-user-password password \
--engine aurora-postgresql \
--engine-version 14.3 \
--vpc-security-group-ids security-group \
--db-subnet-group-name subnet-group-name \
--db-cluster-parameter-group-name parameter-group
```

For Windows:

```
aws rds create-db-cluster ^
--db-cluster-identifier cluster-name ^
--master-username postgres ^
--master-user-password password ^
--engine aurora-postgresql ^
--engine-version 14.3 ^
--vpc-security-group-ids security-group ^
--db-subnet-group-name subnet-group ^
--db-cluster-parameter-group-name parameter-group
```

4. Explicitly create the primary instance for your DB cluster. Use the name of the cluster that you created in step 3 for the --db-cluster-identifier argument when you call the [create-db-instance](#) command, as shown following.

For Linux, macOS, or Unix:

```
aws rds create-db-instance \
--db-instance-identifier instance-name \
--db-instance-class db.r6g \
--db-subnet-group-name subnet-group \
--db-cluster-identifier cluster-name \
--engine aurora-postgresql
```

For Windows:

```
aws rds create-db-instance ^
--db-instance-identifier instance-name ^
--db-instance-class db.r6g ^
--db-subnet-group-name subnet-group ^
--db-cluster-identifier cluster-name ^
--engine aurora-postgresql
```

# Migrating a SQL Server database to Babelfish for Aurora PostgreSQL

You can use Babelfish for Aurora PostgreSQL to migrate an SQL Server database to an Amazon Aurora PostgreSQL DB cluster. Before migrating, review [Using Babelfish with a single database or multiple databases \(p. 1065\)](#).

## Topics

- [Overview of the migration process \(p. 1093\)](#)
- [Evaluating and handling differences between SQL Server and Babelfish \(p. 1094\)](#)
- [Import/export tools for migrating from SQL Server to Babelfish \(p. 1094\)](#)

## Overview of the migration process

The following summary overview lists the steps required to successfully migrate your SQL Server application and make it work with Babelfish. For information about the tools you can use for the export and import processes and for more detail, see [Import/export tools for migrating from SQL Server to Babelfish \(p. 1094\)](#).

1. Create a new Aurora PostgreSQL DB cluster with Babelfish turned on. To learn how, see [Creating a Babelfish for Aurora PostgreSQL DB cluster \(p. 1086\)](#).

To import the various SQL artifacts exported from your SQL Server database in the steps that follow, you connect to the Babelfish cluster using a native SQL Server tool such as `sqlcmd`. For more information, see [Using a SQL Server client to connect to your DB cluster \(p. 1100\)](#).

2. On the SQL Server database that you want to migrate, export the data definition language (DDL). 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).

To do so, you can use a SQL Server client tool such as Microsoft SQL Server Management Studio (SMSS). Regardless of the tool, we recommend that you export your SQL Server DDL in two distinct phases:

- Generate the DDL for the table objects alone, without foreign keys, indexes, or other constraints.
- Generate the DDL for other objects, such as views and stored procedures.

For more information, see [Using SQL Server Management Studio \(SSMS\) to migrate to Babelfish \(p. 1095\)](#).

3. 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.
4. Run an assessment tool to evaluate the scope of any changes that you might need to make so that Babelfish can effectively support the application running on SQL Server. For more information, see [Evaluating and handling differences between SQL Server and Babelfish \(p. 1094\)](#).
5. On your new Babelfish DB cluster, run the DDL to create only the schemas and tables with their primary key constraints.
6. To load the data, you can use the SSMS Import/Export Wizard or another tool. For more information, see [Import/export tools for migrating from SQL Server to Babelfish \(p. 1094\)](#). 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 rather than your SQL Server database. For more information, see [Connecting to a Babelfish DB cluster \(p. 1097\)](#).

9. Modify your application as needed and retest. For more information, see [Differences between Babelfish for Aurora PostgreSQL and SQL Server \(p. 1108\)](#).

You still need to assess your client-side SQL queries. The schemas generated from your SQL Server instance convert only the server-side SQL code. We recommend that you take the following steps:

- Capture client-side queries by using the SQL Server Profiler with the TSQL\_Replay predefined template. This template captures T-SQL statement information that can then be replayed for iterative tuning and testing. You can start the profiler within SQL Server Management Studio, from the **Tools** menu. Choose **SQL Server Profiler** to open the profiler and choose the TSQL\_Replay template.

To use for your Babelfish migration, start a trace and then run your application using your functional tests. The profiler captures the T-SQL statements. When you finish testing, stop the trace. Save the result to an XML file with your client-side queries (File > Save as > Trace XML File for Replay).

For more information, see [SQL Server Profiler](#) in the Microsoft documentation. For more information about the TSQL\_Replay template, see [SQL Server Profiler Templates](#).

- For applications with complex client-side SQL queries, we recommend that you use Babelfish Compass to 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 modify as needed.

When you're satisfied with all testing, analysis, and any modifications need for your migrated application, you can start using your Babelfish database for production. To do so, stop the original database and redirect live client applications to use the Babelfish TDS port.

## Evaluating and handling differences between SQL Server and Babelfish

For best results, we recommend that you evaluate the generated DDL/DML and the client query code before actually migrating your SQL Server database application to Babelfish. Depending on the version of Babelfish and the specific features of SQL Server that your application implements, you might need to refactor your application or use alternatives for functionality that isn't yet fully supported in Babelfish.

To assess your SQL Server application code, you can use tools such as the following:

- Use Babelfish Compass on the generated DDL to determine to what extent the T-SQL code is supported by Babelfish. Identify T-SQL code that might need modifications before running on Babelfish. For more information about this tool, see [Babelfish Compass tool](#) on GitHub.
- Use the AWS Schema Conversion Tool to help with your migration. The AWS Schema Conversion Tool supports Babelfish as a virtual target, so you don't need to have a Babelfish DB cluster running to find out how well (or how poorly) your migration would work. To learn more, see [Using virtual targets](#) in the [AWS Schema Conversion Tool User Guide](#).

### Note

Babelfish Compass is an open-source tool. Report any issues with Babelfish Compass through GitHub rather than to AWS Support.

## Import/export tools for migrating from SQL Server to Babelfish

As of Babelfish 2.1.0 (Aurora PostgreSQL 14.3), you can choose one of the following tools to export your SQL code (DDL, DML) from SQL Server and then import into Babelfish.

- Use SQL Server Management Studio (SSMS) to export your DDL and DML from SQL Server. For more information, see [Using SQL Server Management Studio \(SSMS\) to migrate to Babelfish \(p. 1095\)](#).

- Use the SSMS Import/Export Wizard. This tool is available through the SSMS, but it's also available as a standalone tool. For more information, see [Welcome to SQL Server Import and Export Wizard](#) in the Microsoft documentation.
- Use the Microsoft bulk data copy program (bcp). Support for bcp in Babelfish 2.1.0 is experimental. Certain bcp options aren't supported, including `-E`, `-T`, and `-h`. This utility lets you copy data from a Microsoft SQL Server instance to a data file in the format you specify. For more information, see [bcp Utility](#) in the Microsoft documentation.

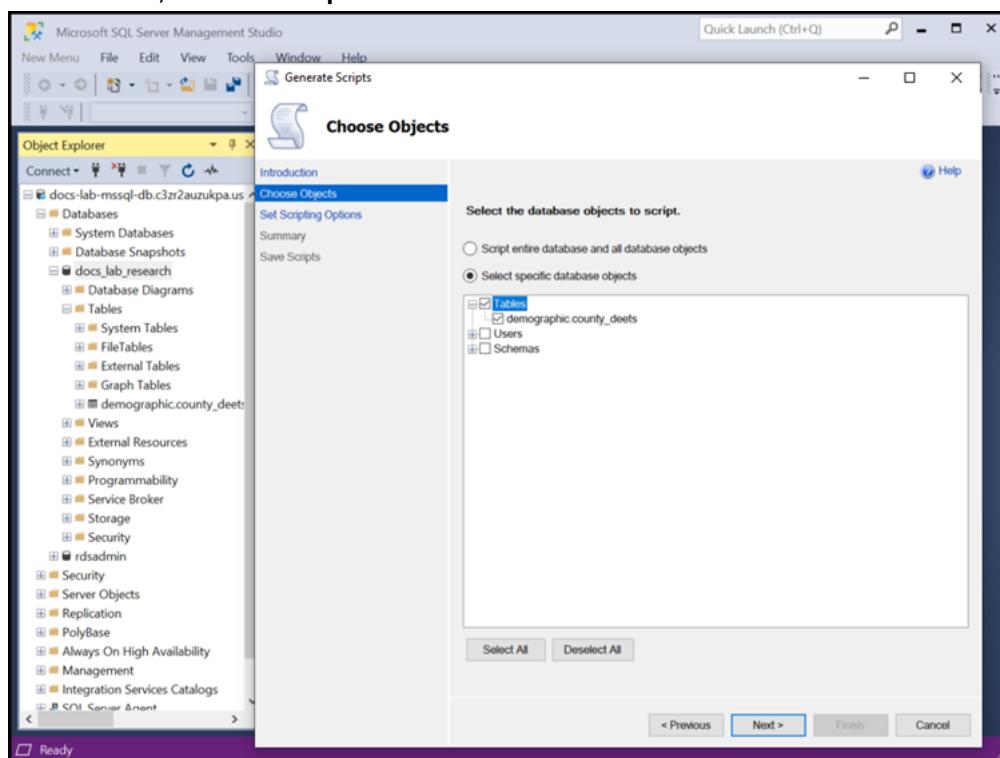
### **Important**

The converse approach isn't currently supported. In other words, you can't use SSMS or another tool to export T-SQL DDL from Babelfish-created objects based on the Babelfish catalogs. The DDL generated in this way isn't reliable for production use.

## Using SQL Server Management Studio (SSMS) to migrate to Babelfish

You can use SQL Server Management Studio (SSMS) to export DDL and DML from your SQL Server instance by following these steps.

1. Connect to your existing SQL Server instance.
2. Open the context menu (right-click) for a database name.
3. Choose **Tasks, Generate Scripts...** from the context menu.



4. On the **Choose Objects** pane, choose **Select specific database objects**. Choose **Tables**, select all tables. Choose **Next** to generate the one of two scripts to create the DDL to use on your Babelfish DB cluster.
5. On the **Set Scripting Options** page, choose **Advanced** to open the **Options** settings. Turn on triggers, logins, owners, and permissions. These are turned off by default in SSMS.
6. Save the script.
7. Repeat the process to generate scripts for the remaining DDL.

To export your DML, follow these steps:

1. Open the context (right-click) menu for a database name.
2. Choose **Tasks, Generate Scripts** from the context menu.
3. On the **Choose Objects** page, select the entire database or specific objects.
4. On the **Set Scripting Options** page, choose **Advanced** and for **Types of data to script**, choose **Data only**.
5. Save the script.

## Connecting to a Babelfish DB cluster

To connect to Babelfish, you 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 through 7.4:

- Open Database Connectivity (ODBC)
- OLE DB Driver/MSOLEDBSQL
- Java Database Connectivity (JDBC) version 8.2.2 (mssql-jdbc-8.2.2) and higher
- Microsoft SqlClient Data Provider for SQL Server
- .NET Data Provider for SQL Server
- SQL Server Native Client 11.0 (deprecated)
- OLE DB 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.

To learn more about connecting to Aurora PostgreSQL in general, see [Connecting to an Amazon Aurora PostgreSQL DB cluster \(p. 212\)](#).

### Topics

- [Finding the writer endpoint and port number \(p. 1097\)](#)
- [Creating C# or JDBC client connections to Babelfish \(p. 1099\)](#)
- [Using a SQL Server client to connect to your DB cluster \(p. 1100\)](#)
- [Using a PostgreSQL client to connect to your DB cluster \(p. 1103\)](#)

## Finding the writer endpoint and port number

To connect to your Babelfish DB cluster, you use the endpoint associated with the DB cluster's writer (primary) instance. The instance must have a status of **Available**. It can take up to 20 minutes for the instances to be available after creating the Babelfish for Aurora PostgreSQL DB cluster.

### To find your database endpoint

1. Open the console for Babelfish.
2. Choose **Databases** from the navigation pane.
3. Choose your Babelfish for Aurora PostgreSQL DB cluster from those listed to see 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 'babelfish-workshop' is highlighted with a red circle. It contains two instances: 'babelfish-workshop-instance-1' (Writer instance) and 'babelfish-workshop-instance-2' (Reader instance). Both instances are listed as 'Available'. In the 'Endpoints' section, there are two entries: 'babelfish-workshop.cluster-ro...' (Reader instance) and 'babelfish-workshop.cluster-...' (Writer instance), both also circled in red.

DB identifier	Role	Engine	Region & AZ	Size	Status
babelfish-workshop	Regional cluster	Aurora PostgreSQL	us-east-1	2 instances	Available
babelfish-workshop-instance-1	Writer instance	Aurora PostgreSQL	us-east-1c	db.r6g.large	Available
babelfish-workshop-instance-2	Reader instance	Aurora PostgreSQL	us-east-1b	db.r6g.large	Available

Endpoint name	Status	Type	Port
babelfish-workshop.cluster-ro...	Available	Reader instance	5432, 1433 (Babelfish)
babelfish-workshop.cluster-...	Available	Writer instance	5432, 1433 (Babelfish)

For more information about Aurora DB cluster details, see [Creating an Amazon Aurora DB cluster \(p. 127\)](#).

## Creating C# or JDBC client connections to Babelfish

In the following you can find some examples of using C# and JDBC classes to connect to an Babelfish for Aurora PostgreSQL.

### 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");

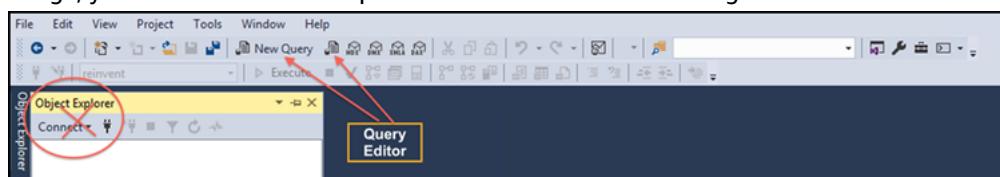
Connection con = ds.getConnection();
```

## 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. As of Babelfish 2.1.0 and higher releases, you can use the SSMS Object Explorer or the SSMS Query Editor to connect to your Babelfish cluster.

### Limitations

- Using `PARSE` to check SQL syntax doesn't work as it should. Rather than checking the syntax without running the query, the `PARSE` command runs the query but doesn't display any results. Using the SSMS `<Ctrl><F5>` key combination to check syntax has the same anomalous behavior, that is, Babelfish unexpectedly runs the query without providing any output.
- Babelfish 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.
- For Babelfish 1.3.0 and older versions, only the Query Editor is supported for SSMS. To use SSMS with Babelfish, be sure to open the Query Editor connection dialog in SSMS, and not the Object Explorer. If the Object Explorer dialog does open, cancel the dialog and re-open the Query Editor. In the following image, you can find the menu options to choose when connecting to Babelfish 1.3.0 or older versions.



For more information about interoperability and behavioral differences between SQL Server and Babelfish, see [Differences between Babelfish for Aurora PostgreSQL and SQL Server \(p. 1108\)](#).

## 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 [Getting information from the Babelfish system catalog \(p. 1106\)](#).

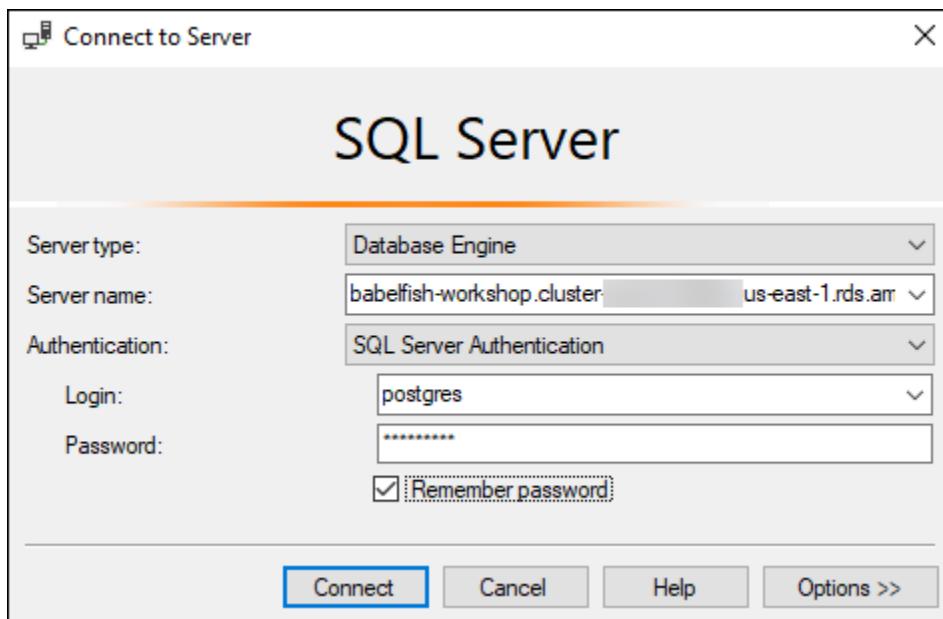
## Using SSMS to connect to the DB cluster

You can connect to an Aurora PostgreSQL DB cluster running Babelfish by using Microsoft SQL Server Management Studio (SSMS). SSMS includes a variety of tools, including the SQL Server Import and Export Wizard discussed in [Migrating a SQL Server database to Babelfish for Aurora PostgreSQL \(p. 1093\)](#). For more information about SSMS, see [Download SQL Server Management Studio \(SSMS\)](#) in the Microsoft documentation.

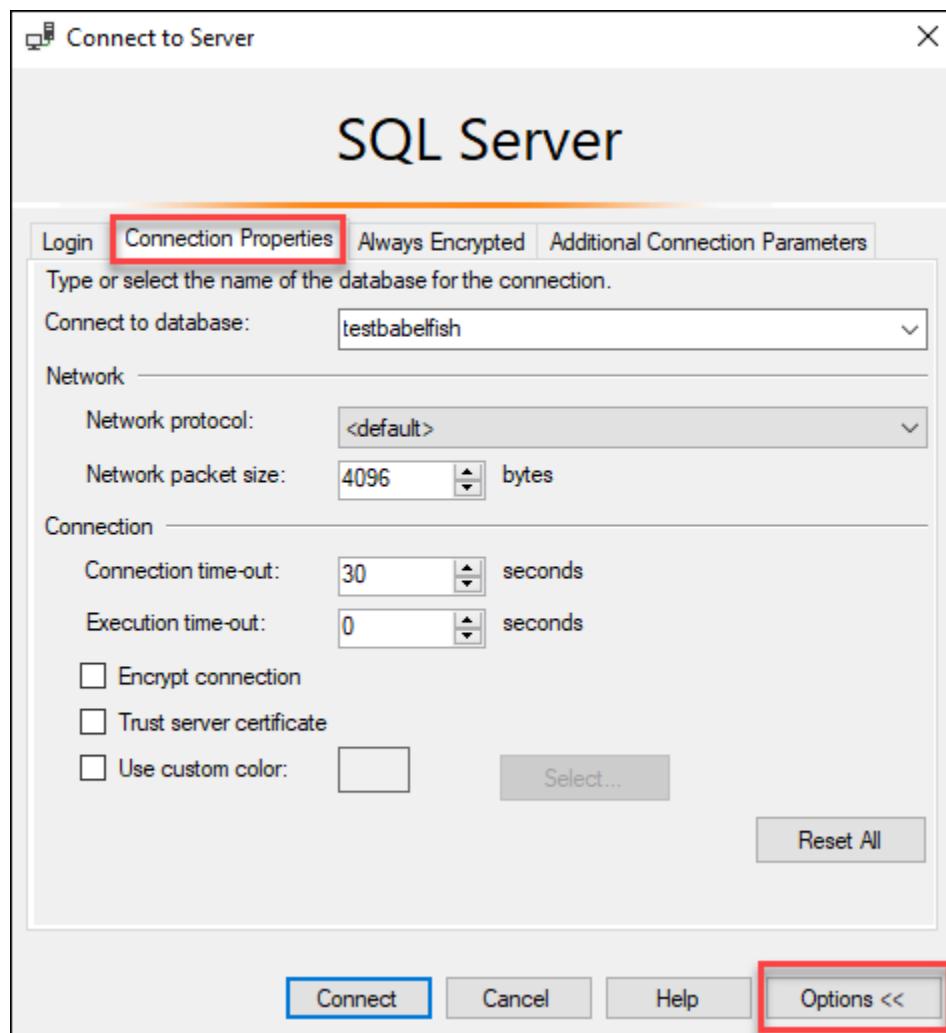
### To connect to your Babelfish database with SSMS

1. Start SSMS.
2. Open the **Connect to Server** dialog box. To continue with the connection, do one of the following:
  - Choose **New Query**.
  - If the Query Editor is open, choose **Query, Connection, Connect**.
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 to Babelfish, see [Connection failure \(p. 1123\)](#).

For more information about SQL Server connection issues, 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 (by default, port 5432). Typically, you don't need to specify the port number unless you changed it from the default. Use the following command to connect to Babelfish from the `psql` client:

```
psql -h bfish-db.cluster-123456789012.aws-region.rds.amazonaws.com
      -p 5432 -U postgres -d babelfish_db -p
```

The parameters are as follows:

- `-h` – The host name of the DB cluster (cluster endpoint) that you want to access.
- `-p` – The PostgreSQL port number used to connect to your DB instance.
- `-d` – `babelfish_db`
- `-U` – The database user account that you want to access. (The example shows the default master username.)
- `-p` – 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`.

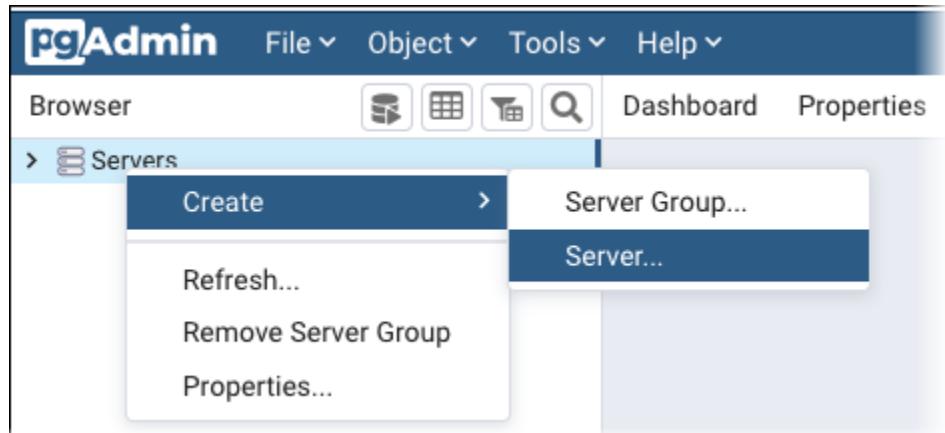
For more information about using the `psql` client to query an Aurora PostgreSQL cluster, see the [PostgreSQL documentation](#).

### 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.

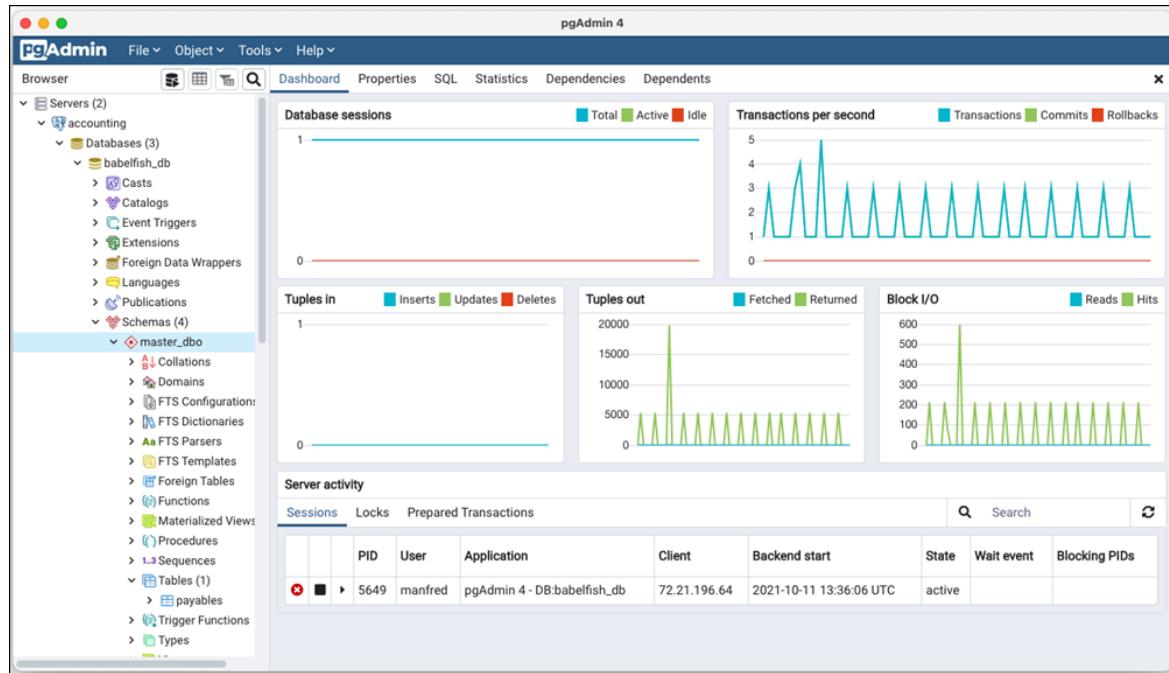
On the **Connection** tab, add the Aurora PostgreSQL cluster address for **Host** and the PostgreSQL port number (by default, 5432) for **Port**. Provide authentication details, and choose **Save**.

The screenshot shows the "Create - Server" dialog box with the "Connection" tab selected. The form fields are as follows:

Setting	Value
Host name/address	babelfish_db.cluster- .us-east-1.rds.ama
Port	5432
Maintenance database	babelfish_db
Username	postgres
Kerberos authentication?	(Toggle switch off)
Password	.....
Save password?	(Toggle switch off)
Role	(Empty field)
Service	(Empty field)

At the bottom of the dialog are buttons for "Close", "Reset", and "Save".

After connecting, you can use pgAdmin functionality to monitor and manage your Aurora PostgreSQL cluster on the PostgreSQL port.



To learn more, see the [pgAdmin web page](#).

## Working with Babelfish

Following, you can find usage information for Babelfish, including some of the differences between working with Babelfish and SQL Server, and between Babelfish and PostgreSQL databases.

### Topics

- [Getting information from the Babelfish system catalog \(p. 1106\)](#)
- [Differences between Babelfish for Aurora PostgreSQL and SQL Server \(p. 1108\)](#)
- [Using Babelfish features with limited implementation \(p. 1113\)](#)
- [Using explain plan to improve Babelfish query performance \(p. 1114\)](#)
- [Using Aurora PostgreSQL extensions with Babelfish \(p. 1117\)](#)

## Getting information from the Babelfish system catalog

You can obtain information about the database objects that are stored in your Babelfish cluster by querying many of the same system views as used in SQL Server. Each new release of Babelfish adds support for more system views. For a list of available views currently available, see the [SQL Server system catalog views table](#).

These system views provide information from the system catalog (`sys.schemas`). In the case of Babelfish, these views contain both SQL Server and PostgreSQL system schemas. To query Babelfish for system catalog information, you can use the TDS port or the PostgreSQL port, as shown in the following examples.

- **Query the T-SQL port using `sqlcmd` or another SQL Server client.**

```
1> SELECT * FROM sys.schemas
2> GO
```

This query returns SQL Server and Aurora PostgreSQL system schemas, as shown in the following.

```
name
-----
demographic_dbo
public
sys
master_dbo
tempdb_dbo
...
```

- **Query the PostgreSQL port using `psql` or `pgAdmin`.** This example uses the `psql` list schemas metacommand (`\dn`):

```
babelfish_db=> \dn
```

The query returns the same result set as that returned by `sqlcmd` on the T-SQL port.

```
List of schemas
Name
-----
demographic_dbo
public
sys
```

```
master_dbo
tempdb_dbo
...
```

## SQL Server system catalogs available in Babelfish

In the following table you can find the SQL Server views currently implemented in Babelfish. For more information about the system catalogs in SQL Server, see [System Catalog Views \(Transact-SQL\)](#) in Microsoft documentation.

<b>View name</b>	<b>Description or Babelfish limitation (if any)</b>
<code>sys.all_columns</code>	All columns in all tables and views
<code>sys.all_objects</code>	All objects in all schemas
<code>sys.all_sql_modules</code>	The union of <code>sys.sql_modules</code> and <code>sys.system_sql_modules</code>
<code>sys.all_views</code>	All views in all schemas
<code>sys.columns</code>	All columns in user-defined tables and views
<code>sys.configurations</code>	Babelfish support limited to a single read-only configuration.
<code>sys.data_spaces</code>	Contains a row for each data space. This can be a filegroup, partition scheme, or FILESTREAM data filegroup.
<code>sys.database_files</code>	A per-database view that contains one row for each file of a database as stored in the database itself.
<code>sys.database_mirroring</code>	For information, see <a href="#">sys.database_mirroring</a> in Microsoft Transact-SQL documentation.
<code>sys.database_principals</code>	For information, see <a href="#">sys.database_principals</a> in Microsoft Transact-SQL documentation.
<code>sys.database_role_members</code>	For information, see <a href="#">sys.database_role_members</a> in Microsoft Transact-SQL documentation.
<code>sys.databases</code>	All databases in all schemas
<code>sys.dm_exec_connections</code>	For information, see <a href="#">sys.dm_exec_connections</a> in Microsoft Transact-SQL documentation.
<code>sys.dm_exec_sessions</code>	For information, see <a href="#">sys.dm_exec_sessions</a> in Microsoft Transact-SQL documentation.
<code>sys.dm_hadr_database_replica_states</code>	For information, see <a href="#">sys.dm_hadr_database_replica_states</a> in Microsoft Transact-SQL documentation.
<code>sys.dm_os_host_info</code>	For information, see <a href="#">sys.dm_os_host_info</a> in Microsoft Transact-SQL documentation.

<b>View name</b>	<b>Description or Babelfish limitation (if any)</b>
<code>sys.endpoints</code>	For information, see <a href="#">sys.endpoints</a> in Microsoft Transact-SQL documentation.
<code>sys.indexes</code>	For information, see <a href="#">sys.indexes</a> in Microsoft Transact-SQL documentation.
<code>sys.languages</code>	For information, see <a href="#">sys.languages</a> in Microsoft Transact-SQL documentation.
<code>sys.schemas</code>	All schemas
<code>sys.server_principals</code>	All logins and roles
<code>sys.sql_modules</code>	For information, see <a href="#">sys.sql_modules</a> in Microsoft Transact-SQL documentation.
<code>sys.sysconfigures</code>	Babelfish support limited to a single read-only configuration.
<code>sys.syscurconfigs</code>	Babelfish support limited to a single read-only configuration.
<code>sys.sysprocesses</code>	For information, see <a href="#">sys.sysprocesses</a> in Microsoft Transact-SQL documentation.
<code>sys.system_sql_modules</code>	For information, see <a href="#">sys.system_sql_modules</a> in Microsoft Transact-SQL documentation.
<code>sys.table_types</code>	For information, see <a href="#">sys.table_types</a> in Microsoft Transact-SQL documentation.
<code>sys.tables</code>	All tables in a schema
<code>sys.xml_schema_collections</code>	For information, see <a href="#">sys.xml_schema_collections</a> in Microsoft Transact-SQL documentation.

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.

## Differences between Babelfish for Aurora PostgreSQL and SQL Server

Babelfish is an evolving Aurora PostgreSQL feature, with new functionality added in each release since the initial offering in Aurora PostgreSQL 13.4. It's designed to provide T-SQL semantics on top of PostgreSQL through the T-SQL dialect using the TDS port. Each new version of Babelfish adds features and functions that better align with T-SQL functionality and behavior, as shown in the [Supported functionality in Babelfish by version \(p. 1137\)](#) table. For best results when working with Babelfish, we recommend that you understand the differences that currently exist between the T-SQL supported by SQL Server and Babelfish for the latest version. To learn more, see [T-SQL differences in Babelfish \(p. 1109\)](#).

In addition to the differences between T-SQL supported by Babelfish and SQL Server, you might also need to consider interoperability issues between Babelfish and PostgreSQL in the context of the Aurora PostgreSQL DB cluster. As mentioned previously, Babelfish supports T-SQL semantics on top of PostgreSQL through the T-SQL dialect using the TDS port. At the same time, the Babelfish database can also be accessed through the standard PostgreSQL port with PostgreSQL SQL statements. If

you're considering using both PostgreSQL and Babelfish functionality in a production deployment, you need to be aware of the potential interoperability issues between schema names, identifiers, permissions, transactional semantics, multiple result sets, default collations, and so on. In simple terms, when PostgreSQL statements or PostgreSQL access occur in the context of Babelfish, interference between PostgreSQL and Babelfish can occur and can potentially affect syntax, semantics, and compatibility when new versions of Babelfish are released. For complete information and guidance about all the considerations, see the [Guidance on Babelfish Interoperability](#) in the Babelfish for PostgreSQL documentation.

**Note**

Before using both PostgreSQL native functionality and Babelfish functionality in the same application context, we strongly recommend that you consider the issues discussed in the [Guidance on Babelfish Interoperability](#) in the Babelfish for PostgreSQL documentation. These interoperability issues (Aurora PostgreSQL and Babelfish) are relevant only if you plan to use the PostgreSQL database instance in the same application context as Babelfish.

## T-SQL differences in Babelfish

Following, you can find a table of T-SQL functionality as supported in the current release of Babelfish with some notes about differences in the behavior from that of SQL Server.

For more information about support in various versions, see [Supported functionality in Babelfish by version \(p. 1137\)](#). For information about features that currently aren't supported, see [Unsupported functionality in Babelfish \(p. 1128\)](#).

Babelfish is available with Aurora PostgreSQL-Compatible Edition. For more information about Babelfish releases, see the [Release Notes for Aurora PostgreSQL](#).

Functionality or syntax	Description of behavior or difference
\ (line continuation character)	The line continuation character (a backslash prior to a newline) for character and hexadecimal strings isn't currently supported. For character strings, the backslash-newline is interpreted as characters in the string. For hexadecimal strings, backslash-newline results in a syntax error.
@@version	The format of the value returned by @@version is slightly different from the value returned by SQL Server. Your code might not work correctly if it depends on the formatting of @@version.
Aggregate functions	Aggregate functions are partially supported (AVG, COUNT, COUNT_BIG, GROUPING, MAX, MIN, STRING_AGG, and SUM are supported). For a list of unsupported aggregate functions, see <a href="#">Functions that aren't supported (p. 1132)</a> .
ALTER TABLE	Supports adding or dropping a single column or constraint only.
BACKUP 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"> <li>SQL Server <code>sqlcmd</code> returns a blank column name.</li> <li>PostgreSQL <code>psql</code> returns a generated column name.</li> </ul>
CHECKSUM function	Babelfish and SQL Server use different hashing algorithms for the CHECKSUM function. As a result, the hash values generated by

Functionality or syntax	Description of behavior or difference
	CHECKSUM function in Babelfish might be different from those generated by CHECKSUM function in SQL Server.
Column default	When creating a column default, the constraint name is ignored. To drop a column default, use the following syntax: ALTER TABLE...ALTER COLUMN..DROP DEFAULT...
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.
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 (ALTER SERVER ROLE sysadmin ADD MEMBER <a href="#">login</a>). 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, 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.
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, 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 are supported except for PASSWORD, DEFAULT_DATABASE, ENABLE, DISABLE.



Functionality or syntax	Description of behavior or difference
Procedure or function parameter limit	Babelfish supports a maximum of 100 parameters for a procedure or function.
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.
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.
Server-level roles	The sysadmin server-level role is supported. Other server-level roles (other than sysadmin) aren't supported.
Database-level roles other than db_owner	The db_owner database-level roles is supported. Other database-level roles (other than db_owner) aren't supported.
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.
sysdatabases.cmptlevel	sysdatabases.cmptlevel are always NULL.
tempdb isn't reinitialized at restart	Permanent objects (like tables and procedures) created in tempdb aren't removed when the database is restarted.
TEXTIMAGE_ON <i>filegroup</i>	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.
Virtual computed columns (non-persistent)	Virtual computed columns are created as persistent.
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.

## Using Babelfish features with limited implementation

Each new version of Babelfish adds support for features that better align with T-SQL functionality and behavior. Still, there are some unsupported features and differences in the current implementation. In the following, you can find information about functional differences between Babelfish and T-SQL, with some workarounds or usage notes.

As of version 1.2.0 of Babelfish, the following features currently have limited implementations:

- **SQL Server catalogs (system views)** – The catalogs `sys.sysconfigures`, `sys.syscurconfigs`, and `sys.configurations` support a single read-only configuration only. The `sp_configure` isn't currently supported. For more information about the other SQL Server views implemented by Babelfish, see [Getting information from the Babelfish system catalog \(p. 1106\)](#).
- **GRANT permissions** – `GRANT...TO PUBLIC` is supported, but `GRANT..TO PUBLIC WITH GRANT OPTION` is not currently supported.
- **SQL Server ownership chain and permission mechanism limitation** – In Babelfish, the SQL Server ownership chain works for views but not for stored procedures. This means that procedures must be granted explicit access to other objects owned by the same owner as the calling procedures. In SQL Server, granting the caller `EXECUTE` permissions on the procedure is sufficient to call other objects owned by same owner. In Babelfish, caller must also be granted permissions on the objects accessed by the procedure.
- **Resolution of unqualified (without schema name) object references** – When a SQL object (procedure, view, function or trigger) references an object without qualifying it with a schema name, SQL Server resolves the object's schema name by using the schema name of the SQL object in which the reference occurs. Currently, Babelfish resolves this differently, by using the default schema of the database user executing the procedure.
- **Default schema changes, sessions, and connections** – If users change their default schema with `ALTER USER...WITH DEFAULT SCHEMA`, the change takes effect immediately in that session. However, for other currently connected sessions belonging to the same user, the timing differs, as follows:
  - For SQL Server: – The change takes effect across all other connections for this user immediately.
  - For Babelfish: – The change takes effect for this user for new connections only.
- **ROWVERSION and TIMESTAMP datatypes implementation and escape hatch setting**
  - The `ROWVERSION` and `TIMESTAMP` datatypes are now supported in Babelfish. To use `ROWVERSION` or `TIMESTAMP` in Babelfish, you must change the setting for the escape hatch `babelfishpg_tsql.escape_hatch_rowversion` from its default (`strict`) to `ignore`. The Babelfish implementation of the `ROWVERSION` and `TIMESTAMP` datatypes is mostly semantically identical to SQL Server, with the following exceptions:
    - The built-in `@@DBTS` function behaves similarly to SQL Server, but with small differences. Rather than returning the last-used value for `SELECT @@DBTS`, Babelfish generates a new timestamp, due to the underlying PostgreSQL database engine and its multi-version concurrency control (MVCC) implementation.
    - In SQL Server, every inserted or updated row gets a unique `ROWVERSION/TIMESTAMP` value. In Babelfish, every inserted row updated by the same statement is assigned the same `ROWVERSION/TIMESTAMP` value.

For example, when an `UPDATE` statement or `INSERT-SELECT` statement affects multiple rows, in SQL Server, the affected rows all have different values in their `ROWVERSION/TIMESTAMP` column. In Babelfish (PostgreSQL), the rows have the same value.

- In SQL Server, when you create a new table with `SELECT-INTO`, you can cast an explicit value (such as `NULL`) to a to-be-created `ROWVERSION/TIMESTAMP` column. When you do the same thing in Babelfish, an actual `ROWVERSION/TIMESTAMP` value is assigned to each row in the new table for you, by Babelfish.

These minor differences in ROWVERSION/TIMESTAMP datatypes shouldn't have an adverse impact on applications running on Babelfish.

**Schema creation, ownership, and permissions** – Permissions to create objects in a schema that was created by the database owner (using `CREATE SCHEMA...AUTHORIZATION DBO`) differ for SQL Server and Babelfish non-DBO users, as shown in the following table:

Database user (non-DBO) can do the following:	SQL Server	Babelfish
Create objects in the schema without additional grants by the DBO?	No	Yes
Reference objects created in the schema by DBO without additional grants?	Yes	No

## Using explain plan to improve Babelfish query performance

Starting with version 2.1.0, Babelfish includes two functions that transparently use the PostgreSQL optimizer to generate estimated and actual query plans for T-SQL queries on the TDS port. These functions are similar to using `SET STATISTICS PROFILE` or `SET SHOWPLAN_ALL` with SQL Server databases to identify and improve slow running queries.

**Note**

Getting query plans from functions, control flows, and cursors isn't currently supported.

In the table you can find a comparison of query plan explain functions across SQL Server, Babelfish, and PostgreSQL.

SQL Server	Babelfish	PostgreSQL
<code>SHOWPLAN_ALL</code>	<code>BABELFISH_SHOWPLAN_ALL</code>	<code>EXPLAIN</code>
<code>STATISTICS PROFILE</code>	<code>BABELFISH_STATISTICS PROFILE</code>	<code>EXPLAIN ANALYZE</code>
Uses the SQL Server optimizer	Uses the PostgreSQL optimizer	Uses the PostgreSQL optimizer
SQL Server input and output format	SQL Server input and PostgreSQL output format	PostgreSQL input and output format
Set for the session	Set for the session	Apply to a specific statement
Supports the following:	Supports the following:	Supports the following:
<ul style="list-style-type: none"> <li>• <code>SELECT</code></li> <li>• <code>INSERT</code></li> <li>• <code>UPDATE</code></li> <li>• <code>DELETE</code></li> <li>• <code>CURSOR</code></li> <li>• <code>CREATE</code></li> <li>• <code>EXECUTE</code></li> <li>• <code>EXEC</code> and functions, including control flow (<code>CASE</code>, <code>WHILE-BREAK-CONTINUE</code>, <code>WAITFOR</code>, <code>BEGIN-END</code>, <code>IF-ELSE</code>, and so on)</li> </ul>	<ul style="list-style-type: none"> <li>• <code>SELECT</code></li> <li>• <code>INSERT</code></li> <li>• <code>UPDATE</code></li> <li>• <code>DELETE</code></li> <li>• <code>CREATE</code></li> <li>• <code>EXECUTE</code></li> <li>• <code>EXEC</code></li> </ul>	<ul style="list-style-type: none"> <li>• <code>SELECT</code></li> <li>• <code>INSERT</code></li> <li>• <code>UPDATE</code></li> <li>• <code>DELETE</code></li> <li>• <code>CURSOR</code></li> <li>• <code>CREATE</code></li> <li>• <code>EXECUTE</code></li> </ul>

Use the Babelfish functions as follows:

- SET BABELFISH\_SHOWPLAN\_ALL [ON|OFF] – Set to ON to generate an estimated query execution plan. This function implements the behavior of the PostgreSQL EXPLAIN command. Use this command to obtain the explain plan for given query.
- SET BABELFISH\_STATISTICS PROFILE [ON|OFF] – Set to ON for actual query execution plans. This function implements the behavior of PostgreSQL's EXPLAIN ANALYZE command.

For more information about PostgreSQL EXPLAIN and EXPLAIN ANALYZE see [EXPLAIN](#) in the PostgreSQL documentation.

For example, the following command sequence turns on query planning and then returns an estimated query execution plan for the SELECT statement without running the query. This example uses the SQL Server sample northwind database using the sqlcmd command-line tool to query the TDS port:

```
1> SET BABELFISH_SHOWPLAN_ALL ON
2> GO
1> SELECT t.territoryid, e.employeeid FROM
2> dbo.employeeteritories e, dbo.territories t
3> WHERE e.territoryid=e.territoryid ORDER BY t.territoryid;
4> GO

QUERY PLAN
-----
Query Text: SELECT t.territoryid, e.employeeid FROM
dbo.employeeteritories e, dbo.territories t
WHERE e.territoryid=e.territoryid ORDER BY t.territoryid
Sort (cost=6231.74..6399.22 rows=66992 width=10)
Sort Key: t.territoryid NULLS FIRST
-> Nested Loop (cost=0.00..861.76 rows=66992 width=10)
    -> Seq Scan on employeeteritories e (cost=0.00..22.70 rows=1264 width=4)
        Filter: ((territoryid)::"varchar" IS NOT NULL)
    -> Materialize (cost=0.00..1.79 rows=53 width=6)
        -> Seq Scan on territories t (cost=0.00..1.53 rows=53 width=6)
```

When you finish reviewing and adjusting your query, you turn off the function as shown following:

```
1> SET BABELFISH_SHOWPLAN_ALL OFF
```

With BABELFISH\_STATISTICS PROFILE set to ON, each executed query returns its regular result set followed by an additional result set that shows actual query execution plans. Babelfish generates the query plan that provides the fastest result set when it invokes the SELECT statement.

```
1> SET BABELFISH_STATISTICS PROFILE ON
1>
2> GO
1> SELECT e.employeeid, t.territoryid FROM
2> dbo.employeeteritories e, dbo.territories t
3> WHERE t.territoryid=e.territoryid ORDER BY t.territoryid;
4> GO
```

The result set and the query plan are returned (this example shows only the query plan).

**QUERY PLAN**

```
-----
Query Text: SELECT e.employeedid, t.territoryid FROM
dbo.employeeterritories e, dbo.territories t
WHERE t.territoryid=e.territoryid ORDER BY t.territoryid
Sort  (cost=42.44..43.28 rows=337 width=10)
      Sort Key: t.territoryid NULLS FIRST

->  Hash Join  (cost=2.19..28.29 rows=337 width=10)
    Hash Cond: ((e.territoryid)::"varchar" = (t.territoryid)::"varchar")
        ->  Seq Scan on employeeterritories e  (cost=0.00..22.70 rows=1270 width=36)
        ->  Hash  (cost=1.53..1.53 rows=53 width=6)
            ->  Seq Scan on territories t  (cost=0.00..1.53 rows=53 width=6)
```

To learn more about how to analyze your queries and the results returned by the PostgreSQL optimizer, see [explain.depesz.com](https://explain.depesz.com). For more information about PostgreSQL EXPLAIN and EXPLAIN ANALYZE, see [EXPLAIN](#) in the PostgreSQL documentation.

## Parameters that control Babelfish explain options

You can use the parameters shown in the following table to control the type of information that's displayed by your query plan.

Parameter	Description
<code>babelfishpg_tsql.explain_buffers</code>	A boolean that turns on (and off) buffer usage information for the optimizer. (Default: off) (Allowable: off, on)
<code>babelfishpg_tsql.explain_costs</code>	A boolean that turns on (and off) estimated startup and total cost information for the optimizer. (Default: on) (Allowable: off, on)
<code>babelfishpg_tsql.explain_format</code>	Specifies the output format for the EXPLAIN plan. (Default: text) (Allowable: text, xml, json, yaml)
<code>babelfishpg_tsql.explain_settings</code>	A boolean that turns on (or off) the inclusion of information about configuration parameters in the EXPLAIN plan output. (Default: off) (Allowable: off, on)
<code>babelfishpg_tsql.explain_summary</code>	A boolean that turns on (or off) summary information such as total time after the query plan. (Default: off) (Allowable: off, on)
<code>babelfishpg_tsql.explain_timing</code>	A boolean that turns on (or off) actual startup time and time spent in each node in the output. (Default: off) (Allowable: off, on)
<code>babelfishpg_tsql.explain_verbose</code>	A boolean that turns on (or off) the most detailed version of an explain plan. (Default: off) (Allowable: off, on)
<code>babelfishpg_tsql.explain_wal</code>	A boolean that turns on (or off) generation of WAL record information as part of an explain plan. (Default: off) (Allowable: off, on)

You can check the values of any Babelfish-related parameters on your system by using either PostgreSQL client or SQL Server client. You can get your current parameter values as follows:

```
1> :setvar SQLCMDMAXVARTYPEWIDTH 35
2> :setvar SQLCMDMAXFIXEDTYPEWIDTH 10
3> SELECT name,
4> setting,
5> short_desc,
6> pending_restart
7> FROM pg_settings
8> WHERE name LIKE '%babelfishpg_tsql.explain%';
9> GO
```

In the following output, you can see that all settings on this particular Babelfish DB cluster are at their default values. Not all output is shown in this example.

name	setting	short_desc
babelfishpg_tsql.explain_buffers	off	Include information on buffer usage
babelfishpg_tsql.explain_costs	on	Include information on estimated st
babelfishpg_tsql.explain_format	text	Specify the output format, which ca
babelfishpg_tsql.explain_settings	off	Include information on configuratio
babelfishpg_tsql.explain_summary	off	Include summary information (e.g.,
babelfishpg_tsql.explain_timing	off	Include actual startup time and tim
babelfishpg_tsql.explain_verbose	off	Display additional information rega
babelfishpg_tsql.explain_wal	off	Include information on WAL record g

(8 rows affected)

You can change the setting for these parameters using a `SELECT` statement with the PostgreSQL `set_config` function, as shown in the following example.

```
SELECT set_config('babelfishpg_tsql.explain_verbose', 'on', false);
```

For more information about `set_config`, see [Configuration Settings Functions](#) in the PostgreSQL documentation.

## 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 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 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 \(p. 1103\)](#). You can also use pgAdmin. For details, see [Using pgAdmin to connect to the DB cluster \(p. 1103\)](#).

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 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.444455556666-us-east-1.rds.amazonaws.com \
-U postgres \
-d babelfish_db \
-p 5432
```

For Windows:

```
psql -h your-Babelfish.cluster.444455556666-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 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 \(p. 1233\)](#).
  - To export data from your Babelfish 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 \(p. 1246\)](#).

You can now use Amazon S3 with the aws\_s3 extension with your Babelfish 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 [Importing data from Amazon S3 to your Aurora PostgreSQL DB cluster \(p. 1238\)](#).
  - For examples of using aws\_s3 functions to export data, see [Exporting query data using the aws\\_s3.query\\_export\\_to\\_s3 function \(p. 1249\)](#).
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 data from Amazon S3 into an Aurora PostgreSQL DB cluster \(p. 1230\)](#) and [Exporting data from an Aurora PostgreSQL DB cluster to Amazon S3 \(p. 1243\)](#).

## 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 and 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",  
            "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 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 \(p. 1255\)](#).

## 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 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 \(p. 1103\)](#).
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).

- a. 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)	

- 
- b. 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 \(p. 1100\)](#). 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 \(p. 1254\)](#). For more information about working with Lambda functions, see [Getting started with Lambda](#) in the *AWS Lambda Developer Guide*.

## Troubleshooting Babelfish

Following, you can find troubleshooting ideas and workarounds for some Babelfish DB cluster issues.

### Topics

- [Connection failure \(p. 1123\)](#)
- [Using pg\\_dump and pg\\_restore requires extra setup \(p. 1123\)](#)

## 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*.
- **Mismatching SSL configurations** – If the `rds.force_ssl` parameter is turned on (set to 1) on Aurora PostgreSQL, then clients must connect to Babelfish over SSL. If your client isn't set up correctly, you see an error message such as the following:

```
Cannot connect to your-Babelfish-DB-cluster, 1433
-----
ADDITIONAL INFORMATION:
no pg_hba.conf entry for host "256.256.256.256", user "your-user-name",
"database babelfish_db", SSL off (Microsoft SQL Server, Error: 33557097)
...
```

This error indicates a possible SSL configuration issue between your local client and the Babelfish DB cluster, and that the cluster requires clients to use SSL (its `rds.force_ssl` parameter is set to 1). For more information about configuring SSL, see [Using SSL with a PostgreSQL DB instance](#) in the *Amazon RDS User Guide*.

If you are using SQL Server Management Studio (SSMS) to connect to Babelfish and you see this error, you can choose **Encrypt connection** and **Trust server certificate** connection options on the Connection Properties pane and try again. These settings handle the SSL connection requirement for SSMS.

For more information about troubleshooting Aurora connection issues, see [Can't connect to Amazon RDS DB instance \(p. 1761\)](#).

## 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 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 work around this issue, first create the same logical database on the target cluster that is on the source. After 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. Connect to your SQL Server endpoint using SQL Server Management Studio (SSMS).

2. Create the same logical database on your target that is on your source. To restore multiple databases, make sure to create those in advance.

```
CREATE DATABASE your-DB-name
```

3. 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](#) in the PostgreSQL documentation.

## 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 parameter groups \(p. 215\)](#). For information about Babelfish-specific parameters, see [DB cluster parameter group settings for Babelfish \(p. 1068\)](#).

## Babelfish version 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. For more information, see the [Release Notes for Aurora PostgreSQL](#).

For a list of supported functionality across different Babelfish releases, see [Supported functionality in Babelfish by version \(p. 1137\)](#).

For a list of currently unsupported functionality, see [Unsupported functionality in Babelfish \(p. 1128\)](#).

**Note**

Currently, you can't upgrade Babelfish DB clusters running on Aurora PostgreSQL 13.7 or older versions to Aurora PostgreSQL 14.3 with Babelfish 2.1.0.

## Identifying your version of Babelfish

You can query Babelfish to find details about the Babelfish version, the Aurora PostgreSQL version, and the compatible Microsoft SQL Server version. You can use the TDS port or the PostgreSQL port.

### To use the TDS port to query for version information

1. Use `sqlcmd` or `ssms` to connect to the endpoint for your Babelfish DB cluster.

```
sqlcmd -S bfish_db.cluster-123456789012.aws-region.rds.amazonaws.com,1433 -U  
        login-id -P password -d db_name
```

2. To identify the Babelfish version, run the following query:

```
1> SELECT CAST(serverproperty('babelfishversion') AS VARCHAR)  
2> GO
```

The query returns results similar to the following:

```
serverproperty  
-----  
2.1.0  
  
(1 rows affected)
```

3. To identify the version of the Aurora PostgreSQL DB cluster, run the following query:

```
1> SELECT aurora_version() AS aurora_version  
2> GO
```

The query returns results similar to the following:

```
aurora_version  
  
-----  
14.3.0  
  
(1 rows affected)
```

4. To identify the compatible Microsoft SQL Server version, run the following query:

```
1> SELECT @@VERSION AS version
```

```
2> GO
```

The query returns results similar to the following:

```
Babelfish for Aurora PostgreSQL with SQL Server Compatibility - 12.0.2000.8
May 23 2022 19:17:14
Copyright (c) Amazon Web Services
PostgreSQL 14.3 on x86_64-pc-linux-gnu

(1 rows affected)
```

As an example that shows one minor difference between Babelfish and Microsoft SQL Server, you can run the following query. On Babelfish, the query returns 1, while on Microsoft SQL Server, the query returns NULL.

```
SELECT CAST(serverproperty('babelfish') AS VARCHAR) AS runs_on_babelfish
```

You can also use the PostgreSQL port to obtain version information, as shown in the following procedure.

### To use the PostgreSQL port to query for version information

1. Use psql or pgAdmin to connect to the endpoint for your Babelfish DB cluster.

```
psql host=bfish_db.cluster-123456789012.aws-region.rds.amazonaws.com
port=5432 dbname=babelfish_db user=sa
```

2. Turn on the extended feature (\x) of psql for more readable output.

```
babelfish_db=> \x
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;
```

The query returns output similar to the following:

```
-[ RECORD 1 ]-----
aurora_version      | 14.3.0
postgresql_version   | PostgreSQL 14.3 on x86_64-pc-linux-gnu, compiled by x86_64-
pc-linux-gnu-gcc (GCC) 7.4.0, 64-bit
babelfish_compatibility | Babelfish for Aurora Postgres with SQL Server Compatibility -
12.0.2000.8
                           +
                           | May 23 2022 19:17:14
                           +
                           | Copyright (c) Amazon Web Services
                           +
                           | PostgreSQL 14.3 on x86_64-pc-linux-gnu
babelfish_version     | 2.1.0
```

# Babelfish for Aurora PostgreSQL reference

## Topics

- [Unsupported functionality in Babelfish \(p. 1128\)](#)
- [Supported functionality in Babelfish by version \(p. 1137\)](#)

## Unsupported functionality in Babelfish

In the following table and lists, you can find functionality that isn't currently supported in Babelfish. Updates to Babelfish are included in Aurora PostgreSQL versions. For more information, see the [Release Notes for Aurora PostgreSQL](#).

## Topics

- [Functionality that isn't currently supported \(p. 1128\)](#)
- [Settings that aren't supported \(p. 1131\)](#)
- [Commands that aren't supported \(p. 1131\)](#)
- [Column names or attributes that aren't supported \(p. 1131\)](#)
- [Object types that aren't supported \(p. 1131\)](#)
- [Functions that aren't supported \(p. 1132\)](#)
- [Syntax that isn't supported \(p. 1133\)](#)

## Functionality that isn't currently supported

In the table you can find information about certain functionality that isn't currently supported.

Functionality or syntax	Description
Assembly modules and SQL Common Language Runtime (CLR) routines	Functionality related to assembly modules and CLR routines isn't supported.
Column attributes	ROWGUIDCOL, SPARSE, FILESTREAM, and MASKED aren't supported.
Contained databases	Contained databases with logins authenticated at the database level rather than at the server level aren't supported.
CROSS APPLY	Lateral joins aren't supported.
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 for output parameters (an error is raised).
Cursor options	SCROLL, KEYSET, DYNAMIC, FAST_FORWARD, SCROLL_LOCKS, OPTIMISTIC, TYPE_WARNING, and FOR UPDATE
Data encryption	Data encryption isn't supported.

Functionality or syntax	Description
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.
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.
EXECUTE AS USER clause	This clause isn't supported in functions, procedures, or triggers.
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.
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.
Global temporary tables (tables with names that start with ##)	Global temporary tables aren't supported.
Graph functionality	All SQL graph functionality isn't supported.
Hints	Hints aren't supported for joins, queries, or tables.
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.
Inline indexes	Inline indexes aren't supported.
Invoking a procedure whose name is in a variable	Using a variable as a procedure name isn't supported.
JSON_MODIFY	Updating JSON string using JSON_MODIFY isn't supported.
FOR JSON clause	Formatting query results as JSON isn't supported.
Materialized views	Materialized views aren't supported.
NOT FOR REPLICATION clause	This syntax is accepted and ignored.

Functionality or syntax	Description
ODBC escape functions	ODBC escape functions aren't supported.
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.
Remote object references	Objects with four-part names aren't supported.. For more information, see: <a href="#">DB cluster parameter group settings for Babelfish (p. 1068)</a> .
Row-level security	Row-level security with CREATE SECURITY POLICY and inline table-valued functions isn't supported.
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.
SP_CONFIGURE	This system stored procedure isn't supported.
SQL keyword SPARSE	The keyword SPARSE is accepted and ignored.
Table value constructor syntax (FROM clause)	The unsupported syntax is for a derived table constructed with the FROM clause.
Temporal tables	Temporal tables aren't supported.
Temporary procedures aren't dropped automatically	This functionality isn't supported.
Transaction isolation levels	READUNCOMMITTED is treated the same as READCOMMITTED. REPEATABLEREAD, and SERIALIZABLE aren't supported.
Triggers, externally defined	These triggers aren't supported, including SQL Common Language Runtime (CLR).
Triggers, INSTEAD-OF on views	INSTEAD-OF triggers on views aren't supported. INSTEAD-OF triggers are supported (Babelfish 1.2.0 and higher releases).
Unquoted string values in stored procedure calls and default values	String parameters to stored procedure calls, and defaults for string parameters in CREATE PROCEDURE, aren't supported.

Functionality or syntax	Description
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.

## 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
- SET PARSEONLY ON (command doesn't work as expected)

## Commands that aren'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 or attributes that aren't supported

The following column names aren't supported:

- \$IDENTITY
- \$ROWGUID
- IDENTITYCOL

## 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

## Functions that aren't supported

The following built-in functions aren't supported:

### Aggregate functions

- APPROX\_COUNT\_DISTINCT
- CHECKSUM\_AGG
- GROUPING\_ID
- ROWCOUNT\_BIG
- STDEV
- STDEVP
- VAR
- VARP

### Cryptographic functions

- CERTENCODED function
- CERTID function
- CERTPROPERTY function

### Metadata functions

- COLUMNPROPERTY
- OBJECTPROPERTY
- OBJECTPROPERTYEX
- TYPEPROPERTY
- SERVERPROPERTY function – The following properties aren't supported:
  - BuildClrVersion
  - ComparisonStyle
  - ComputerNamePhysicalNetBIOS
  - HadrManagerStatus
  - InstanceDefaultDataPath

- InstanceDefaultLogPath
- InstanceName
- IsClustered
- IsHadrEnabled
- LCID
- MachineName
- NumLicenses
- ProcessID
- ProductBuild
- ProductBuildType
- ProductLevel
- ProductUpdateLevel
- ProductUpdateReference
- ResourceLastUpdateDateTime
- ResourceVersion
- ServerName
- SqlCharSet
- SqlCharSetName
- SqlSortOrder
- SqlSortOrderName
- FilestreamShareName
- FilestreamConfiguredLevel
- FilestreamEffectiveLevel

## Security functions

- CERTPRIVATEKEY
- LOGINPROPERTY

## Statements, operators, other functions

- EVENTDATA function
- GET\_TRANSMISSION\_STATUS
- OPENXML

## 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 SERVICE, BACKUP/RESTORE SERVICE MASTER KEY clause
- ALTER VIEW
- BEGIN CONVERSATION TIMER
- BEGIN DISTRIBUTED TRANSACTION
- BEGIN DIALOG CONVERSATION
- BULK INSERT
- CREATE COLUMNSTORE INDEX
- CREATE EXTERNAL FILE FORMAT
- CREATE EXTERNAL TABLE
- CREATE, ALTER, DROP APPLICATION ROLE
- CREATE, ALTER, DROP ASSEMBLY
- CREATE, ALTER, DROP ASYMMETRIC KEY
- CREATE, ALTER, DROP CREDENTIAL
- CREATE, ALTER, DROP CRYPTOGRAPHIC PROVIDER
- CREATE, ALTER, DROP ENDPOINT
- CREATE, ALTER, DROP EVENT SESSION
- CREATE, ALTER, DROP EXTERNAL LANGUAGE
- CREATE, ALTER, DROP EXTERNAL RESOURCE POOL
- CREATE, ALTER, DROP FULLTEXT CATALOG
- CREATE, ALTER, DROP FULLTEXT INDEX
- CREATE, ALTER, DROP FULLTEXT STOPLIST
- CREATE, ALTER, DROP MESSAGE TYPE
- CREATE, ALTER, DROP OPEN/CLOSE, BACKUP/RESTORE MASTER KEY
- CREATE, ALTER, DROP PARTITION FUNCTION
- CREATE, ALTER, DROP PARTITION SCHEME
- CREATE, ALTER, DROP QUEUE
- CREATE, ALTER, DROP RESOURCE GOVERNOR
- CREATE, ALTER, DROP RESOURCE POOL
- CREATE, ALTER, DROP ROUTE
- CREATE, ALTER, DROP SEARCH PROPERTY LIST
- CREATE, ALTER, DROP SECURITY POLICY
- CREATE, ALTER, DROP SELECTIVE XML INDEX clause
- CREATE, ALTER, DROP SERVICE
- CREATE, ALTER, DROP SPATIAL INDEX
- CREATE, ALTER, DROP TYPE
- CREATE, ALTER, DROP XML INDEX
- CREATE, ALTER, DROP XML SCHEMA COLLECTION
- CREATE/DROP RULE
- CREATE, DROP WORKLOAD CLASSIFIER
- CREATE, ALTER, DROP WORKLOAD GROUP
- CREATE/ALTER/ENABLE/DISABLE TRIGGER
- CREATE TABLE... GRANT clause
- CREATE TABLE... IDENTITY clause

- 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. For more information, see [T-SQL differences in Babelfish \(p. 1109\)](#).
- 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 BROWSE
- SELECT... FOR XML AUTO
- SELECT... FOR XML EXPLICIT
- SEND
- SET CONTEXT\_INFO
- SET DATEFORMAT
- SET DEADLOCK\_PRIORITY
- SET FMTONLY
- SET FORCEPLAN
- SET NO\_BROWSETABLE
- 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
- UPDATERETEXT
- 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

## Supported functionality in Babelfish by version

In the following table you can find T-SQL functionality supported by different Babelfish versions. For lists of unsupported functionality, see [Unsupported functionality in Babelfish \(p. 1128\)](#). For information about various Babelfish releases, see the [Release Notes for Aurora PostgreSQL](#).

Functionality or syntax	2.1.0	1.2.1	1.1.0	1.0.0
ALTER ROLE	✓	–	–	–
Connecting with the SSMS object explorer connection dialog	✓	–	–	–
CREATE ROLE	✓	–	–	–
Create unique indexes	✓	✓	✓	–
Cross-database references	✓	–	–	–
SELECT,SELECT..INTO, INSERT, UPDATE, DELETE				
Cursor-typed parameters for input parameters only (not output)	✓	✓	✓	✓
Data migration using the bcp client utility	✓	–	–	–
Data migration with the SSMS Import/ Export Wizard	✓	–	–	–
Datatypes TIMESTAMP, ROWVERSION (for usage information, see Features with limited implementation)	✓	✓	–	–
DROP IF EXISTS (for SCHEMA, DATABASE, and USER objects)	✓	✓	–	–
DROP ROLE	✓	–	–	–
Identifiers with leading dot character	✓	✓	✓	–
INSTEAD OF triggers on tables (tables only, not views)	✓	✓	–	–

Functionality or syntax	2.1.0	1.2.1	1.1.0	1.0.0
Partial support for the SSMS object explorer	✓	-	-	-
SET BABELFISH_SHOWPLAN_ALL ON (and OFF)	✓	-	-	-
SET BABELFISH_STATISTICS PROFILE ON (OFF)	✓	-	-	-
SET LOCK_TIMEOUT	✓	✓	-	-
Triggers with multiple DML actions can reference transition tables	✓	✓	✓	-
<b>Built-in functions:</b>				
CHARINDEX	✓	-	-	-
CHOOSE	✓	✓	✓	-
COLUMNS_UPDATED	✓	✓	-	-
COLUMNPROPERTY (CharMaxLen, AllowsNull only)	✓	✓	✓	-
CONCAT_WS	✓	✓	✓	-
CURSOR_STATUS	✓	✓	✓	-
DATEFROMPARTS	✓	✓	✓	-
DATETIMEFROMPARTS	✓	✓	✓	-
FULLTEXTSERVICEPROPERTY	✓	-	-	-
HAS_DBACCESS	✓	✓	-	-
HAS_PERMS_BY_NAME	✓	-	-	-
IS_MEMBER	✓	-	-	-
IS_ROLEMEMBER	✓	-	-	-
IS_SRVROLEMEMBER	✓	✓	-	-
ISJSON	✓	✓	-	-
JSON_QUERY	✓	✓	-	-
JSON_VALUE	✓	✓	-	-
OPENJSON	✓	-	-	-
ORIGINAL_LOGIN	✓	✓	✓	-

Functionality or syntax	2.1.0	1.2.1	1.1.0	1.0.0
PATINDEX	✓	-	-	-
SCHEMA_NAME	✓	✓	✓	-
SESSION_USER	✓	✓	✓	-
SQUARE	✓	✓	✓	-
STRING_SPLIT	✓	-	-	-
SUSER_SID	✓	✓	-	-
SUSER_SNAME	✓	✓	-	-
TRIGGER_NESTLEVEL	✓ (without arguments only)	✓	✓	-
UPDATE	✓	✓	-	-
<b>INFORMATION_SCHEMA catalogs</b>				
COLUMNS	✓	✓	-	-
DOMAINS	✓	✓	-	-
TABLES	✓	✓	-	-
TABLE_CONSTRAINTS	✓	✓	-	-
<b>System-defined @@ variables:</b>				
@@CURSOR_ROWS	✓	✓	✓	-
@@DATEFIRST	✓	✓	✓	✓
@@DBTS	✓	✓	-	-
@@ERROR	✓	✓	✓	✓
@@ERROR=213	✓	-	-	-
@@FETCH_STATUS	✓	✓	✓	✓
@@IDENTITY	✓	✓	✓	✓
@@LOCK_TIMEOUT	✓	✓	✓	-
@@LOCK_TIMEOUT	✓	✓	-	-
@@MAX_CONNECTIONS		✓	✓	-
@@MAX_PRECISION	✓	✓	✓	✓
@@MICROSOFTVERSION		✓	✓	-
@@NESTLEVEL	✓	✓	✓	-
@@PROCID	✓	✓	✓	-
@@ROWCOUNT	✓	✓	✓	✓

Functionality or syntax	2.1.0	1.2.1	1.1.0	1.0.0
@@SERVERNAME	✓	✓	✓	✓
@@SERVICENAME	✓	✓	–	–
@@SPID	✓	✓	✓	✓
@@TRANCOUNT	✓	✓	✓	✓
@@VERSION (note format difference as described in <a href="#">T-SQL differences in Babelfish (p. 1109)</a> .)	✓	✓	✓	✓
<b>System stored procedures:</b>				
sp_column_privileges	✓	✓	–	–
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_resultset	✓	✓	✓	–
sp_describe_declared_parameters	✓	–	–	–
sp_fkeys	✓	✓	–	–
sp_getapplock	✓	✓	✓	✓
sp_helpdb	✓	✓	✓	✓

Functionality or syntax	2.1.0	1.2.1	1.1.0	1.0.0
sp_helprole	✓	–	–	–
sp_helprolemember	✓	–	–	–
sp_helpuser	✓	✓	–	–
sp_oledb_ro_username	✓	✓	✓	–
sp_pkeys	✓	✓	✓	–
sp_pkeys	✓	✓	–	–
sp_prepare	✓	✓	✓	–
sp_releaseapplock	✓	✓	✓	✓
sp_special_columns	✓	✓	–	–
sp_sproc_columns	✓	–	–	–
sp_sproc_columns_100	✓	–	–	–
sp_statistics	✓	✓	✓	–
sp_statistics_100	✓	✓	✓	–
sp_stored_procedures	✓	✓	–	–
sp_table_privileges	✓	✓	–	–
sp_tablecollations_100	✓	✓	✓	–
sp_tables	✓	✓	✓	–
sp_unprepare	✓	✓	✓	–
xp_qv	✓	✓	–	–
<b>Properties supported on the CONNECTIONPROPERTY system function</b>				
auth_scheme	✓	✓	✓	✓
client_net_address	✓	✓	–	–
local_net_address	✓	✓	✓	–
local_tcp_port	✓	✓	✓	✓
net_transport	✓	✓	✓	✓
protocol_type	✓	✓	✓	✓
physical_net_transport	✓	✓	–	–
<b>Properties supported on the SERVERPROPERTY function</b>				
Babelfish	✓	✓	✓	✓
Collation	✓	✓	✓	✓
CollationID	✓	✓	✓	✓

Functionality or syntax	2.1.0	1.2.1	1.1.0	1.0.0
Edition	✓	✓	✓	✓
EditionID	✓	✓	–	–
EngineEdition	✓	✓	–	–
IsAdvancedAnalyticsInstalled		✓	–	–
IsBigDataCluster	✓	✓	–	–
IsFullTextInstalled	✓	✓	–	–
IsIntegratedSecurityOnly		✓	–	–
IsLocalDB	✓	✓	–	–
IsPolyBaseInstalled	✓	✓	–	–
IsSingleUser	✓	✓	✓	✓
IsXTPSupported	✓	✓	–	–
Japanese_CI_AI	✓	–	–	–
Japanese_CI_AS	✓	–	–	–
Japanese_CS_AS	✓	–	–	–
LicenseType	✓	✓	–	–
ProductMajorVersion	✓	✓	–	–
ProductMinorVersion	✓	✓	–	–
ProductVersion	✓	✓	–	–
ServerName	✓	✓	✓	✓
<b>SQL Server views supported by Babelfish</b>				
sys.all_columns	✓	✓	✓	✓
sys.all_objects	✓	✓	✓	✓
sys.all_sql_modules	✓	–	–	–
sys.all_views	✓	✓	✓	✓
sys.columns	✓	✓	✓	✓
sys.configurations	✓	✓	–	–
sys.data_spaces	✓	–	–	–
sys.database_files	✓	–	–	–
sys.database_mirroring	✓	–	–	–
sys.database_principals	✓	✓	–	–
sys.database_role_members		–	–	–

Functionality or syntax	2.1.0	1.2.1	1.1.0	1.0.0
sys.databases	✓	✓	✓	✓
sys.dm_exec_connections		✓	–	–
sys.dm_exec_sessions	✓	✓	–	–
sys.dm_hadr_database_replica_states		–	–	–
sys.dm_os_host_info	✓	✓	–	–
sys.endpoints	✓	✓	–	–
sys.indexes	✓	–	–	–
sys.schemas	✓	✓	✓	✓
sys.server_principals	✓	✓	✓	✓
sys.sql_modules	✓	–	–	–
sys.sysconfigures	✓	✓	–	–
sys.sysconfigs	✓	✓	–	–
sys.sysprocesses	✓	✓	–	–
sys.table_types	✓	✓	–	–
sys.tables	✓	✓	✓	✓
sys.xml_schema_collections		–	–	–
syslanguages	✓	–	–	–

## 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 \(p. 1143\)](#)
- [Maximum connections to an Aurora PostgreSQL DB instance \(p. 1144\)](#)
- [Temporary storage limits for Aurora PostgreSQL \(p. 1146\)](#)
- [Testing Amazon Aurora PostgreSQL by using fault injection queries \(p. 1147\)](#)
- [Displaying volume status for an Aurora PostgreSQL DB cluster \(p. 1151\)](#)
- [Specifying the RAM disk for the stats\\_temp\\_directory \(p. 1152\)](#)

## 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 \(p. 278\)](#).

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 \(p. 73\)](#) and [Fast failover with Amazon Aurora PostgreSQL \(p. 1201\)](#).

For detailed specifications of the DB instance classes supported by Aurora PostgreSQL, see [Supported DB engines for DB instance classes \(p. 57\)](#).

## Maximum connections to an Aurora PostgreSQL DB instance

An Aurora PostgreSQL DB cluster allocates resources based on the DB instance class and its available memory. Each connection to the DB cluster consumes incremental amounts of these resources, such as memory and CPU. Memory consumed per connection varies based on query type, count, and whether temporary tables are used. Even an idle connection consumes memory and CPU. That's because when queries run on a connection, more memory is allocated for each query and it's not released completely, even when processing stops. Thus, we recommend that you make sure your applications aren't holding on to idle connections: each one of these wastes resources and affects performance negatively. For more information, see [Resources consumed by idle PostgreSQL connections](#).

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. 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. Before modifying the `max_connections` parameter setting, we recommend that you consider the following:

- 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 this happens, attempts to connect using `psql` raise error messages such as the following:

```
psql: FATAL: remaining connection slots are reserved for non-replication superuser connections
```

- If the `max_connections` value exceeds the number of connections that are actually needed, the unused connections can cause performance to degrade.

The value of `max_connections` is derived from the following Aurora PostgreSQL `LEAST` function:

```
LEAST({DBInstanceClassMemory/9531392}, 5000)
```

If you want to change the value for `max_connections`, you need to create a custom DB cluster parameter group and change its value there. After applying your custom DB parameter group to your cluster, be sure to reboot the primary instance so the new value takes effect. For more information, see [Amazon Aurora PostgreSQL parameters \(p. 1369\)](#) and [Creating a DB cluster parameter group \(p. 219\)](#).

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. 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 \(p. 57\)](#). For the amount of memory for each DB instance class, [Hardware specifications for DB instance classes for Aurora \(p. 64\)](#).
- Pool connections by using RDS Proxy with your Aurora PostgreSQL DB cluster. For more information, see [Using Amazon RDS Proxy \(p. 1430\)](#).

For details about how Aurora Serverless v2 instances handle this parameter, see [Parameters that Aurora computes based on Aurora Serverless v2 maximum capacity \(p. 1537\)](#).

Instance class	Default max_connections 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
db.r5.large	1600
db.r4.16xlarge	5000

Instance class	Default max_connections value
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

## 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 data sets during query processing or for index build operations. These local storage volumes are backed by Amazon Elastic Block Store and can be extended. For more information about storage, see [Amazon Aurora storage and reliability \(p. 67\)](#).

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-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 instance class	Maximum temporary storage available (GiB)
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 \(p. 525\)](#). (This doesn't apply to Aurora Serverless v2.)

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 let you crash the instance so that you can test failover and recovery. You can also simulate Aurora Replica failure, disk failure, and disk congestion. Fault injection queries are supported by all available Aurora PostgreSQL versions, as follows.

- Aurora PostgreSQL versions 12, 13, 14, and higher
- Aurora PostgreSQL version 11.7 and higher
- Aurora PostgreSQL version 10.11 and higher

## Topics

- [Testing an instance crash \(p. 1148\)](#)
- [Testing an Aurora Replica failure \(p. 1148\)](#)
- [Testing a disk failure \(p. 1149\)](#)
- [Testing disk congestion \(p. 1150\)](#)

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 \(p. 35\)](#).

## 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 \(p. 1151\)](#).

*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 \(p. 1151\)](#).

*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. To learn more about Aurora cluster volumes in general, see [Amazon Aurora storage and reliability \(p. 67\)](#).

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 \(p. 1147\)](#).

**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 \(p. 1414\)](#).

### 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 [Run-time Statistics](#) in 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. This parameter denotes MB, so you must use an integer value. Expressions, formulas, and functions aren't valid for the `rds.pg_stat_ramdisk_size` parameter. Be sure to restart the DB cluster so that the change takes effect. For information about setting parameters, see [Working with parameter groups \(p. 215\)](#). For more information about restarting the DB cluster, see [Rebooting an Amazon Aurora DB cluster or Amazon Aurora DB instance \(p. 329\)](#).

As an 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)
```

## Tuning with wait events for Aurora PostgreSQL

Wait events are an important tuning tool for Aurora PostgreSQL. When you can find out why sessions are waiting for resources and what they are doing, you're better able to reduce bottlenecks. You can use the information in this section to find possible causes and corrective actions. Before delving into this section, we strongly recommend that you understand basic Aurora concepts, especially the following topics:

- [Amazon Aurora storage and reliability \(p. 67\)](#)
- [Managing performance and scaling for Aurora DB clusters \(p. 274\)](#)

### Important

The wait events in this section are specific to Aurora PostgreSQL. Use the information in this section to tune Amazon Aurora only, 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 \(p. 1153\)](#)
- [Aurora PostgreSQL wait events \(p. 1156\)](#)
- [Client:ClientRead \(p. 1158\)](#)
- [Client:ClientWrite \(p. 1160\)](#)
- [CPU \(p. 1161\)](#)
- [IO:BufFileRead and IO:BufFileWrite \(p. 1166\)](#)
- [IO:DataFileRead \(p. 1171\)](#)
- [IO:XactSync \(p. 1177\)](#)
- [ipc:damrecordtxack \(p. 1179\)](#)
- [Lock:advisory \(p. 1180\)](#)
- [Lock:extend \(p. 1182\)](#)
- [Lock:Relation \(p. 1184\)](#)
- [Lock:transactionid \(p. 1187\)](#)
- [Lock:tuple \(p. 1189\)](#)
- [lwlock:buffer\\_content \(BufferContent\) \(p. 1192\)](#)
- [LWLock:buffer\\_mapping \(p. 1193\)](#)
- [LWLock:BufferIO \(p. 1195\)](#)
- [LWLock:lock\\_manager \(p. 1196\)](#)
- [Timeout:PgSleep \(p. 1199\)](#)

## 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 \(p. 1153\)](#)
- [Aurora PostgreSQL memory \(p. 1154\)](#)
- [Aurora PostgreSQL processes \(p. 1155\)](#)

## 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 \(p. 1154\)](#)
- [Local memory in Aurora PostgreSQL \(p. 1155\)](#)

## 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 \(p. 1154\)](#)
- [Write ahead log \(WAL\) buffers \(p. 1154\)](#)

### 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 \(p. 1155\)](#)
- [Maintenance work memory area \(p. 1155\)](#)
- [Temporary buffer area \(p. 1155\)](#)

### 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 \(p. 1155\)](#)
- [Backend processes \(p. 1156\)](#)
- [Background processes \(p. 1156\)](#)

### 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 \(p. 1395\)](#).

Wait event	Definition
<a href="#">Client:ClientRead (p. 1158)</a>	This event occurs when Aurora PostgreSQL is waiting to receive data from the client.

Wait event	Definition
<a href="#">Client:ClientWrite (p. 1160)</a>	This event occurs when Aurora PostgreSQL is waiting to write data to the client.
<a href="#">CPU (p. 1161)</a>	This event occurs when a thread is active in CPU or is waiting for CPU.
<a href="#">IO:BufFileRead and IO:BufFileWrite (p. 1166)</a>	These events occur when Aurora PostgreSQL creates temporary files.
<a href="#">IO:DataFileRead (p. 1171)</a>	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.
<a href="#">IO:XactSync (p. 1177)</a>	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.
<a href="#">ipc:damrecordtxack (p. 1179)</a>	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.
<a href="#">Lock:advisory (p. 1180)</a>	This event occurs when a PostgreSQL application uses a lock to coordinate activity across multiple sessions.
<a href="#">Lock:extend (p. 1182)</a>	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.
<a href="#">Lock:Relation (p. 1184)</a>	This event occurs when a query is waiting to acquire a lock on a table or view that's currently locked by another transaction.
<a href="#">Lock:transactionid (p. 1187)</a>	This event occurs when a transaction is waiting for a row-level lock.
<a href="#">Lock:tuple (p. 1189)</a>	This event occurs when a backend process is waiting to acquire a lock on a tuple.
<a href="#">lwlock:buffer_content (BufferContent) (p. 1192)</a>	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.
<a href="#">LWLock:buffer_mapping (p. 1193)</a>	This event occurs when a session is waiting to associate a data block with a buffer in the shared buffer pool.
<a href="#">LWLock:BufferIO (p. 1195)</a>	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.
<a href="#">LWLock:lock_manager (p. 1196)</a>	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.
<a href="#">Timeout:PgSleep (p. 1199)</a>	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 \(p. 1158\)](#)
- [Context \(p. 1158\)](#)
- [Likely causes of increased waits \(p. 1158\)](#)
- [Actions \(p. 1159\)](#)

## 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 \(p. 1159\)](#)
- [Scale your client \(p. 1159\)](#)
- [Use current generation instances \(p. 1159\)](#)
- [Increase network bandwidth \(p. 1159\)](#)
- [Monitor maximums for network performance \(p. 1159\)](#)
- [Monitor for transactions in the "idle in transaction" state \(p. 1160\)](#)

### 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 \(p. 56\)](#). 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 \(p. 525\)](#).

### 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 \(p. 1160\)](#)
- [Context \(p. 1160\)](#)
- [Likely causes of increased waits \(p. 1160\)](#)
- [Actions \(p. 1161\)](#)

## 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 \(p. 1161\)](#)
- [Use current generation instances \(p. 1161\)](#)
- [Reduce the amount of data sent to the client \(p. 1161\)](#)
- [Scale your client \(p. 1161\)](#)

### 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 \(p. 56\)](#). 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 \(p. 1162\)](#)
- [Context \(p. 1162\)](#)
- [Likely causes of increased waits \(p. 1163\)](#)
- [Actions \(p. 1164\)](#)

## 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 \(p. 1162\)](#)
- [DBLoadCPU metric \(p. 1162\)](#)
- [os.cpuUtilization metrics \(p. 1162\)](#)
- [Likely cause of CPU scheduling \(p. 1163\)](#)

### 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 \(p. 1163\)](#)
- [Likely causes of long-term high frequency \(p. 1163\)](#)
- [Corner cases \(p. 1163\)](#)

### 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 \(p. 1164\)](#)
- [Determine whether the number of connections increased \(p. 1164\)](#)
- [Respond to workload changes \(p. 1165\)](#)

### 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 \(p. 1430\)](#).
- 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 `blk_hit` metrics in Performance Insights. Look for a correlation between an increase in `blk_hit` and CPU usage. The following scenarios are possible:

- CPU usage and `blk_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 `blk_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:BuffFileRead and IO:BuffFileWrite

The `IO:BuffFileRead` and `IO:BuffFileWrite` 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 \(p. 1166\)](#)
- [Context \(p. 1166\)](#)
- [Likely causes of increased waits \(p. 1166\)](#)
- [Actions \(p. 1167\)](#)

### 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 \(p. 1155\)](#) and [Maintenance work memory area \(p. 1155\)](#).

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 \(p. 1167\)](#)
- [Examine your join queries \(p. 1167\)](#)
- [Examine your ORDER BY and GROUP BY queries \(p. 1168\)](#)
- [Avoid using the DISTINCT operation \(p. 1169\)](#)
- [Consider using window functions instead of GROUP BY functions \(p. 1169\)](#)
- [Investigate materialized views and CTAS statements \(p. 1170\)](#)
- [Use pg\\_repack when you create indexes \(p. 1170\)](#)
- [Increase maintenance\\_work\\_mem when you cluster tables \(p. 1170\)](#)
- [Tune memory to prevent IO:BufFileRead and IO:BufFileWrite \(p. 1170\)](#)

### Identify the problem

Assume a situation in which Performance Insights isn't turned on and you suspect that IO:BufFileRead and IO:BufFileWrite 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 uses 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 \(p. 1155\)](#).

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:BufFileWrite and IO:BufFileRead, 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:BufFileRead and IO:BufFileWrite

In some situation, you need to tune memory. Your goal is to balance the following requirements:

- The `work_mem` value (see [Work memory area \(p. 1155\)](#))
- The memory remaining after discounting the `shared_buffers` value (see [Buffer pool \(p. 748\)](#))
- 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 \(p. 1155\)](#).

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 \(p. 748\)](#).

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:BuffFileRead` and `IO:BuffFileWrite` 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 \(p. 1171\)](#)
- [Context \(p. 1171\)](#)
- [Likely causes of increased waits \(p. 1171\)](#)
- [Actions \(p. 1172\)](#)

## 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 \(p. 748\)](#).

## 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 \(p. 1172\)](#)
- [Minimize the effect of maintenance operations \(p. 1173\)](#)
- [Respond to high numbers of connections \(p. 1177\)](#)

## 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 \(p. 1173\)](#)
- [Find indexes consuming unnecessary space \(p. 1175\)](#)
- [Find tables that are eligible to be autovacuumed \(p. 1176\)](#)

## 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=(\d+)'::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.rehasoids 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.rehasoids
      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 * (relopages-est_pages)::float / relopages AS extra_ratio,
fillfactor, bs*(relopages-est_pages_ff) AS bloat_size,
100 * (relopages-est_pages_ff)::float / relopages AS bloat_ratio,
is_na
-- , 100-(sub.pst).avg_leaf_density, est_pages, index_tuple_hdr_bm,
-- maxalign, pagehdr, nulldatawidth, nulldatahdrwidth, sub.reltuples, sub.relopages
-- (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, nspname, table_oid, tblname, idxname, relopages, fillfactor, is_na
    -- ,stattuple.pgstatindex(quote_ident(nspname)||'.'||quote_ident(idxname)) AS pst,
    -- index_tuple_hdr_bm, maxalign, pagehdr, nulldatawidth, nulldatahdrwidth, reltuples
    -- (DEBUG INFO)
FROM (
    SELECT maxalign, bs, nspname, tblname, idxname, reltuples, relopages, 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.nspname, i.tblname, i.idxname, i.reltuples, i.relopages, 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 nullwidth,
      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 nspname, 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.tblname 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 cvsf ON cvsf.param = 'autovacuum_vacuum_scale_factor' AND c.oid =
    cvsf.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 \(p. 1430\)](#).

- 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 \(p. 1178\)](#)
- [Context \(p. 1178\)](#)
- [Likely causes of increased waits \(p. 1178\)](#)
- [Actions \(p. 1178\)](#)

## 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 \(p. 1178\)](#)
- [Scale up the CPU \(p. 1179\)](#)
- [Increase network bandwidth \(p. 1179\)](#)
- [Reduce the number of commits \(p. 1179\)](#)

### 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 \(p. 531\)](#).

## 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 \(p. 64\)](#).

## 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 \(p. 64\)](#).

## 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 \(p. 1179\)](#)
- [Context \(p. 1179\)](#)
- [Causes \(p. 1179\)](#)
- [Actions \(p. 1179\)](#)

## 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 \(p. 1180\)](#)
- [Context \(p. 1180\)](#)
- [Causes \(p. 1180\)](#)
- [Actions \(p. 1181\)](#)

## 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 \(p. 1182\)](#)
- [Context \(p. 1182\)](#)
- [Likely causes of increased waits \(p. 1182\)](#)
- [Actions \(p. 1182\)](#)

## 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 \(p. 1183\)](#)
- [Increase network bandwidth \(p. 1183\)](#)

## 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.username,
    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_query          | 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 \(p. 1333\)](#).
- 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 \(p. 525\)](#). For information about network performance for each DB instance class, see [Hardware specifications for DB instance classes for Aurora \(p. 64\)](#).

## 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 \(p. 1184\)](#)
- [Context \(p. 1184\)](#)
- [Likely causes of increased waits \(p. 1185\)](#)
- [Actions \(p. 1185\)](#)

## 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 occurs more frequently than normal, it can indicate a performance issue. 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 a conflict between the relation locks held by the writer and readers. Currently, only `ACCESS EXCLUSIVE` relation locks are replicated to reader instances. However, the `ACCESS EXCLUSIVE` relation lock will conflict with any `ACCESS SHARE` relation locks held by the reader. This can cause an increase in lock relation wait events on the reader.

## Actions

We recommend different actions depending on the causes of your wait event.

### Topics

- [Reduce the impact of blocking SQL statements \(p. 1185\)](#)
- [Minimize the effect of maintenance operations \(p. 1185\)](#)
- [Check for reader locks \(p. 1186\)](#)

### 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 by running queries that return the lock type and relation. In the table you can find a sequence of queries to two such concurrent sessions, a writer session (left-hand column) and a reader session (right-hand column).

The replay process waits for the duration of `max_standby_streaming_delay` before cancelling the reader query. As shown in the example, the lock timeout of 100ms is well below the default `max_standby_streaming_delay` of 30 seconds. The lock times out before it's an issue.

Writer session	Reader session
<pre>export WRITER=aurorapg1.12345678910.us-west-1.rds.amazonaws.com psql -h \$WRITER psql (15devel, server 10.14) Type "help" for help.</pre>	<pre>export READER=aurorapg2.12345678910.us-west-1.rds.amazonaws.com psql -h \$READER psql (15devel, server 10.14) Type "help" for help.</pre>

The writer session creates table `t1` on the writer instance. The ACCESS EXCLUSIVE lock is acquired on the writer immediately, assuming that there are no conflicting queries on the writer.

```
postgres=> CREATE TABLE t1(b integer);
CREATE TABLE
```

The reader session sets a lock timeout interval of 100 milliseconds.

```
postgres=> SET lock_timeout=100;
SET
```

The reader session tries to read data from table `t1` on the reader instance.

```
postgres=> SELECT * FROM t1;
 b
 ---
 (0 rows)
```

The writer session drops `t1`.

```
postgres=> BEGIN;
BEGIN
postgres=> DROP TABLE t1;
DROP TABLE
postgres=>
```

The query times out and is canceled on the reader.

```
postgres=> SELECT * FROM t1;
ERROR: canceling statement due to lock timeout
```

**Writer session**

**Reader session**

```
LINE 1: SELECT * FROM t1;  
^
```

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.

```
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)
```

## Lock:transactionid

The Lock:transactionid event occurs when a transaction is waiting for a row-level lock.

### Topics

- [Supported engine versions \(p. 1187\)](#)
- [Context \(p. 1187\)](#)
- [Likely causes of increased waits \(p. 1188\)](#)
- [Actions \(p. 1188\)](#)

## 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 \(p. 1188\)](#)
- [Idle in transaction \(p. 1188\)](#)
- [Long-running transactions \(p. 1188\)](#)

### 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 \(p. 1188\)](#)
- [Respond to idle transactions \(p. 1189\)](#)
- [Respond to long-running transactions \(p. 1189\)](#)

### 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 \(p. 1430\)](#).
- 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 \(p. 1189\)](#)
- [Context \(p. 1189\)](#)
- [Likely causes of increased waits \(p. 1190\)](#)
- [Actions \(p. 1190\)](#)

## 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.		

Time	Session 1	Session 2	Session 3
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 \(p. 1190\)](#)
- [Find the blocker session \(p. 1191\)](#)
- [Reduce concurrency when it is high \(p. 1191\)](#)
- [Troubleshoot bottlenecks \(p. 1192\)](#)

### 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 username<>'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 | username | 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 | 1000018
 11999 | sys      | psql      | /*session4*/ update tab set col=1 where col=1; | Lock
         | tuple     | active     | Lock       | tuple        | 1 | 1000024
```

## 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 \(p. 1192\)](#)
- [Context \(p. 1192\)](#)
- [Likely causes of increased waits \(p. 1192\)](#)
- [Actions \(p. 1193\)](#)

## 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 `llock:buffer_content` (`BufferContent`) events by using Amazon RDS Performance Insights or by querying the view `pg_stat_activity`.

### Topics

- [Improve in-memory efficiency \(p. 1193\)](#)
- [Reduce usage of foreign key constraints \(p. 1193\)](#)
- [Remove unused indexes \(p. 1193\)](#)

### 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 \(p. 56\)](#).

### Reduce usage of foreign key constraints

Investigate workloads experiencing high numbers of `llock: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 `llock: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 \(p. 1194\)](#)
- [Context \(p. 1194\)](#)
- [Causes \(p. 1194\)](#)
- [Actions \(p. 1194\)](#)

## 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 \(p. 1154\)](#).

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 \(p. 1194\)](#)
- [Assess your indexing strategy \(p. 1195\)](#)
- [Reduce the number of buffers that must be allocated quickly \(p. 1195\)](#)

#### 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 \(p. 1195\)](#)
- [Context \(p. 1195\)](#)
- [Causes \(p. 1196\)](#)
- [Actions \(p. 1196\)](#)

## 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 \(p. 1171\)](#) 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 \(p. 1197\)](#)
- [Context \(p. 1197\)](#)
- [Likely causes of increased waits \(p. 1197\)](#)
- [Actions \(p. 1198\)](#)

## 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 \(p. 1198\)](#)
- [Remove unnecessary indexes \(p. 1198\)](#)
- [Tune your queries for fast path locking \(p. 1198\)](#)
- [Tune for other wait events \(p. 1199\)](#)
- [Reduce hardware bottlenecks \(p. 1199\)](#)
- [Use a connection pooler \(p. 1199\)](#)
- [Upgrade your Aurora PostgreSQL version \(p. 1199\)](#)

## 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 \(p. 1430\)](#)
- [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 \(p. 1413\)](#).

## 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 \(p. 1200\)](#)
- [Likely causes of increased waits \(p. 1200\)](#)

- [Actions \(p. 1200\)](#)

## 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

Following, you can find several best practices for managing your Amazon Aurora PostgreSQL DB cluster. Be sure to also review basic maintenance tasks. For more information, see [Managing Amazon Aurora PostgreSQL \(p. 1143\)](#).

### Topics

- [Troubleshooting storage issues \(p. 1200\)](#)
- [Fast failover with Amazon Aurora PostgreSQL \(p. 1201\)](#)
- [Fast recovery after failover with cluster cache management for Aurora PostgreSQL \(p. 1208\)](#)
- [Managing Aurora PostgreSQL connection churn with pooling \(p. 1212\)](#)
- [Tuning memory parameters for Aurora PostgreSQL \(p. 1218\)](#)

## 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 that it uses for storing error and message logs.

If your sorts or index creation functions exceed the memory available, you can run out of local storage. If you experience issues with Aurora PostgreSQL running out of storage space, you have a couple of options. 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 \(p. 610\)](#).

If your Aurora cluster is larger than 40 TB, don't use db.t2, db.t3, or db.t4g instance classes.

## Fast failover with Amazon Aurora PostgreSQL

Following, you can learn how to make sure that failover occurs as fast as possible. To recover quickly after failover, you can use cluster cache management for your Aurora PostgreSQL DB cluster. For more information, see [Fast recovery after failover with cluster cache management for Aurora PostgreSQL \(p. 1208\)](#).

Some of the steps that you can take to make failover perform fast include the following:

- Set Transmission Control Protocol (TCP) keepalives with short time frames, to stop longer running queries before the read timeout expires if there's a failure.
- Set timeouts for Java Domain Name System (DNS) caching aggressively. Doing this helps ensure the Aurora read-only endpoint can properly cycle through read-only nodes on later 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 read and write Aurora endpoints that are provided to connect to the cluster.
- Use RDS API operations to test application response on server-side failures. Also, 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](#).

These are covered in more detail following.

### Topics

- [Setting TCP keepalives parameters \(p. 1201\)](#)
- [Configuring your application for fast failover \(p. 1202\)](#)
- [Testing failover \(p. 1205\)](#)
- [Fast failover example in Java \(p. 1206\)](#)

## Setting TCP keepalives parameters

When you set up a TCP connection, a set of timers is associated with the connection. When the keepalive timer reaches zero, a keepalive probe packet is sent to the connection endpoint. If the probe receives a reply, you can assume that the connection is still up and running.

Turning on TCP keepalive parameters and setting them aggressively ensures that if your client can't connect to the database, any active connections are quickly closed. The application can then connect to a new endpoint.

Make sure 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 aren't 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 this time by 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. If keepalive packets are often dropped within the application's network, you can set a higher `tcp_keepalive_probes` value. Doing this allows for more buffer in less reliable networks, although it increases the time that it takes to detect an actual failure.

### To set TCP keepalive parameters on Linux

1. Test how to configure your TCP keepalive parameters.

We recommend doing so by using the command line with the following commands. This suggested configuration is system-wide. In other words, it also 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
```

2. 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
```

## Configuring your application for fast failover

Following, you can find a discussion of several configuration changes for Aurora PostgreSQL that you can make for fast failover. To learn more about PostgreSQL JDBC driver setup and configuration, see the [PostgreSQL JDBC Driver](#) documentation.

### Topics

- [Reducing DNS cache timeouts \(p. 1202\)](#)
- [Setting an Aurora PostgreSQL connection string for fast failover \(p. 1202\)](#)
- [Other options for obtaining the host string \(p. 1204\)](#)

### 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. You can find it by using the Aurora read-only endpoint before DNS updates have fully propagated. Setting the java DNS time to live (TTL) to a low value, such as under 30 seconds, helps cycle between reader nodes on later 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 use Aurora PostgreSQL fast failover, make sure that your application's connection string has a list of hosts instead of just a single host. Following is an example connection string that you can use to connect to an Aurora PostgreSQL cluster. In this example, the hosts are in bold.

```
jdbc:postgresql://myauroracluster.cluster-c9bfei4hjlr.us-east-1-
beta.rds.amazonaws.com:5432,
myauroracluster.cluster-ro-c9bfei4hjlr.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 best availability and to avoid a dependency on the RDS API, we recommend that you maintain a file to connect with. This file contains a host string that your application reads from when you establish a connection to the database. This host string has all the Aurora endpoints available for the cluster. For more information about Aurora endpoints, see [Amazon Aurora connection management \(p. 35\)](#).

For example, you might store your endpoints in a local file as shown following.

```
myauroracluster.cluster-c9bfei4hjlr.us-east-1-beta.rds.amazonaws.com:5432,
myauroracluster.cluster-ro-c9bfei4hjlr.us-east-1-beta.rds.amazonaws.com:5432
```

Your application reads from this file to populate the host section of the JDBC connection string. Renaming the DB cluster causes these endpoints to change. Make sure that your application handles this event if it occurs.

Another option is to use a list of DB instance nodes, as follows.

```
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 loops through all nodes on this list to find a valid connection. In contrast, when you use the Aurora endpoints only two nodes are tried in each connection attempt. However, there's a downside to using DB instance nodes. If you add or remove nodes from your cluster and the list of instance endpoints becomes stale, the connection driver might never find the correct host to connect to.

To help ensure that your application doesn't wait too long to connect to any one host, set the following parameters aggressively:

- `targetServerType` – Controls whether the driver connects to a write or read node. To ensure that your applications 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. It connects to the writer if no reader connection can be established.
- `loginTimeout` – Controls how long your application waits to log in 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 might want to send a "best effort" cancel signal on a query that has timed out. If this cancel signal is in your failover path, 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 two connection handlers. One connection handler runs short-lived queries and sets this value lower. Another connection handler, for long-running queries, has this value set much

higher. With this approach, you can rely on TCP keepalive parameters to stop long-running queries if the server goes down.

- `tcpKeepAlive` – Turn on 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.

In many cases, you need to determine who the writer of the cluster is or to find other reader nodes in the cluster. To do this, your application can connect to any DB instance in the DB cluster and query the `aurora_replica_status` function. You can use this function to reduce the amount of time it takes to find a host to connect to. However, in certain network failure scenarios the `aurora_replica_status` function might show out-of-date or incomplete information.

A good way to ensure that your application can find a node to connect to is to try to connect to the cluster writer endpoint and then the cluster reader endpoint. You do this until you can establish a readable connection. These endpoints don't change unless you rename your DB cluster. Thus, you can generally leave them as static members of your application or store them in a resource file that your application reads from.

After you establish a connection using one of these endpoints, you can get information about the rest of the cluster. To do this, call the `aurora_replica_status` function. For example, the following command retrieves information with `aurora_replica_status`.

```
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)
```

For example, the hosts section of your connection string might start with both the writer and reader cluster endpoints, as shown following.

```
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 attempts 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. To do this, query for the result of the command `SHOW transaction_read_only`.

If the return value of the query is `OFF`, then you successfully connected to the primary node. However, suppose that the return value is `ON` and your application requires a read/write connection. In this case, you can 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 with the `endpointPostfix` described following.

Make sure that you're aware when you connect to a replica that has stale data. When this happens, the `aurora_replica_status` function might show out-of-date information. You can set a threshold for staleness at the application level. To check this, you can look at the difference between the server time and the `last_update_timestamp` value. 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 following the data returned by `aurora_replica_status`.

### [Listing instances using the `DescribeDBClusters` API operation, example in Java](#)

You can programmatically find the list of instances by using the [AWS SDK for Java](#), specifically the `DescribeDBClusters` API operation.

Following is 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(","));
```

Here, `pgJDBCEndpointStr` contains a formatted list of endpoints, as shown following.

```
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 your application can get it by querying the `DescribeDBInstances` API operation for a single instance in your cluster. This value remains constant within an AWS Region and for an individual customer. So it saves an API call to simply keep this constant in a resource file that your application reads from. In the example preceding, it's set to the following.

```
.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 isn't responding or takes too long to respond. The endpoints are guaranteed to be up to date within the time it takes to update the DNS record. Updating the DNS record with an endpoint typically takes less than 30 seconds. You can store the endpoint 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 API operations can cause an outage that can be used to test how your applications responds:

- [`FailoverDBCluster`](#) – This operation attempts to promote a new DB instance in your DB cluster to writer.

The following code example 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 isn't guaranteed with this API operation. It shuts down the database on the writer, however. You can use it to test how your application responds to connections dropping. The `ForceFailover` parameter doesn't apply for Aurora engines. Instead, use the `FailoverDBCluster` API operation.
- [ModifyDBCluster](#) – Modifying the `Port` parameter causes an outage when the nodes in the cluster begin listening on a new port. In general, your application can respond to this failure first by ensuring that only your application controls port changes. Also, ensure that it can appropriately update the endpoints it depends on. You can do this by having someone manually update the port when they make modifications at the API level. Or you can do this by using the RDS API in your application to determine if the port has changed.
- [ModifyDBInstance](#) – Modifying the `DBInstanceClass` parameter causes an outage.
- [DeleteDBInstance](#) – Deleting the primary (writer) causes a new DB instance to be promoted to writer in your DB cluster.

From the application or client side, if you use Linux, you can test how the application responds to sudden packet drops. You can do this based on whether port, host, or if TCP keepalive packets are sent or received by using the `iptables` command.

## Fast failover example in Java

The following code example shows how an application might set up an Aurora PostgreSQL driver manager.

The application calls the `getConnection` function when it needs a connection. A call to `getConnection` can fail to find a valid host. An example is when no writer is found but the `targetServerType` parameter is set to `primary`. In this case, the calling application should simply retry calling the function.

To avoid pushing the retry behavior onto the application, you can wrap this retry call into a connection pooler. With most connection poolers, you can specify a JDBC connection string. So your application can call into `getJdbcConnectionString` and pass that to the connection pooler. Doing this means you can use faster failover with 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:///%s"
    + "/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-c9bfei4hj1rd.us-east-1-
beta.rds.amazonaws.com:5432");
        newEndpointList.add("myauroracluster.cluster-ro-c9bfei4hj1rd.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(","));
    }
}
```

## 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 \(p. 1209\)](#)
  - [Enabling cluster cache management \(p. 1209\)](#)
  - [Setting the promotion tier priority for the writer DB instance \(p. 1210\)](#)
  - [Setting the promotion tier priority for a reader DB instance \(p. 1210\)](#)

- Monitoring the buffer cache (p. 1211)

## Configuring cluster cache management

To configure cluster cache management, do the following processes in order.

### Topics

- Enabling cluster cache management (p. 1209)
- Setting the promotion tier priority for the writer DB instance (p. 1210)
- Setting the promotion tier priority for a reader DB instance (p. 1210)

### 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 \(p. 72\)](#).

### 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();
```

## Managing Aurora PostgreSQL connection churn with pooling

When client applications connect and disconnect so often that Aurora PostgreSQL DB cluster response time slows, the cluster is said to be experiencing *connection churn*. Each new connection to the Aurora PostgreSQL DB cluster endpoint consumes resources, thus reducing the resources that can be used to process the actual workload. Connection churn is an issue that we recommend that you manage by following some of the best practices discussed following.

For starters, you can improve response times on Aurora PostgreSQL DB clusters that have high rates of connection churn. To do this, you can use a connection pooler, such as RDS Proxy. A *connection pooler* provides a cache of ready to use connections for clients. Almost all versions of Aurora PostgreSQL support RDS Proxy. For more information, see [Amazon RDS Proxy with Aurora PostgreSQL \(p. 29\)](#).

If your specific version of Aurora PostgreSQL doesn't support RDS Proxy, you can use another PostgreSQL-compatible connection pooler, such as PgBouncer. To learn more, see the [PgBouncer website](#).

To see if your Aurora PostgreSQL DB cluster can benefit from connection pooling, you can check the postgresql.log file for connections and disconnections. You can also use Performance Insights to find out how much connection churn your Aurora PostgreSQL DB cluster is experiencing. Following, you can find information about both topics.

## Logging connections and disconnections

The PostgreSQL log\_connections and log\_disconnections parameters can capture connections and disconnections to the writer instance of the Aurora PostgreSQL DB cluster. By default, these

parameters are turned off. To turn these parameters on, use a custom parameter group and turn on by changing the value to 1. For more information about custom parameter groups, see [Working with DB cluster parameter groups \(p. 217\)](#). To check the settings, connect to your DB cluster endpoint for Aurora PostgreSQL by using psql and query as follows.

```
labdb=> SELECT setting FROM pg_settings
  WHERE name = 'log_connections';
setting
-----
on
(1 row)
labdb=> SELECT setting FROM pg_settings
  WHERE name = 'log_disconnections';
setting
-----
on
(1 row)
```

With both of these parameters turned on, the log captures all new connections and disconnections. You see the user and database for each new authorized connection. At disconnection time, the session duration is also logged, as shown in the following example.

```
2022-03-07 21:44:53.978 UTC [16641] LOG: connection authorized: user=labtek database=labdb
application_name=psql
2022-03-07 21:44:55.718 UTC [16641] LOG: disconnection: session time: 0:00:01.740
user=labtek database=labdb host=[local]
```

To check your application for connection churn, turn on these parameters if they're not on already. Then gather data in the PostgreSQL log for analysis by running your application with a realistic workload and time period. You can view the log file in the RDS console. Choose the writer instance of your Aurora PostgreSQL DB cluster, and then choose the **Logs & events** tab. For more information, see [Viewing and listing database log files \(p. 597\)](#).

Or you can download the log file from the console and use the following command sequence. This sequence finds the total number of connections authorized and dropped per minute.

```
grep "connection authorized\|disconnection: session time:" postgresql.log.2022-03-21-16 | \
awk '{print $1,$2}' | \
sort | \
uniq -c | \
sort -n -k1
```

In the example output, you can see a spike in authorized connections followed by disconnections starting at 16:12:10.

```
.....
.....
.....
5 2022-03-21 16:11:55 connection authorized:
9 2022-03-21 16:11:55 disconnection: session
5 2022-03-21 16:11:56 connection authorized:
5 2022-03-21 16:11:57 connection authorized:
5 2022-03-21 16:11:57 disconnection: session
32 2022-03-21 16:12:10 connection authorized:
30 2022-03-21 16:12:10 disconnection: session
31 2022-03-21 16:12:11 connection authorized:
27 2022-03-21 16:12:11 disconnection: session
27 2022-03-21 16:12:12 connection authorized:
27 2022-03-21 16:12:12 disconnection: session
41 2022-03-21 16:12:13 connection authorized:
47 2022-03-21 16:12:13 disconnection: session
```

```

46 2022-03-21 16:12:14 connection authorized:
41 2022-03-21 16:12:14 disconnection: session
24 2022-03-21 16:12:15 connection authorized:
29 2022-03-21 16:12:15 disconnection: session
28 2022-03-21 16:12:16 connection authorized:
24 2022-03-21 16:12:16 disconnection: session
40 2022-03-21 16:12:17 connection authorized:
42 2022-03-21 16:12:17 disconnection: session
40 2022-03-21 16:12:18 connection authorized:
40 2022-03-21 16:12:18 disconnection: session
.....
.....
.....
1 2022-03-21 16:14:10 connection authorized:
1 2022-03-21 16:14:10 disconnection: session
1 2022-03-21 16:15:00 connection authorized:
1 2022-03-21 16:16:00 connection authorized:

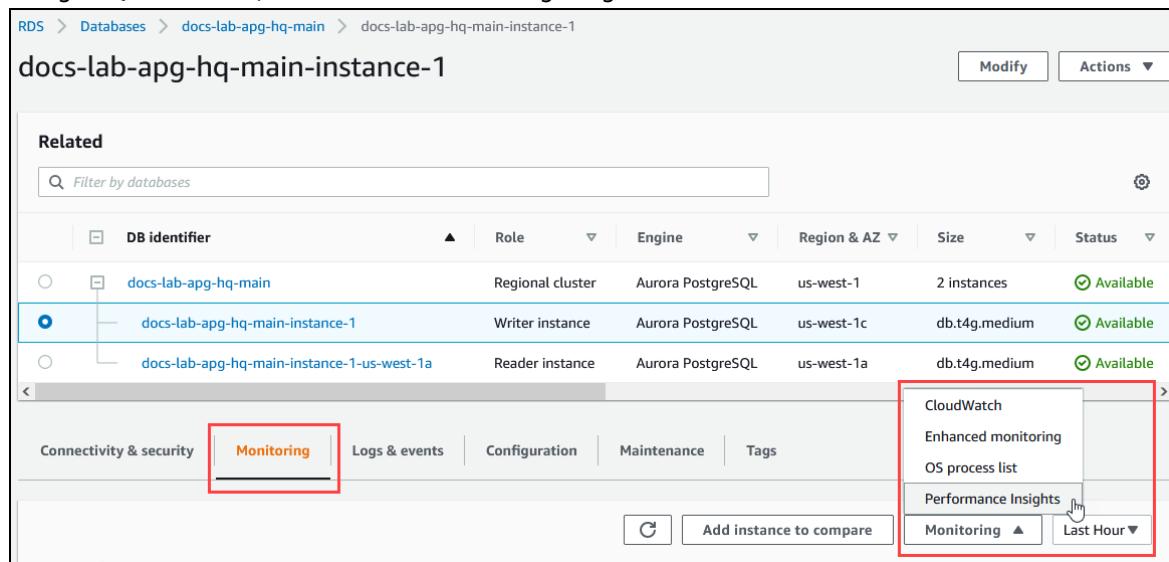
```

With this information, you can decide if your workload can benefit from a connection pooler. For more detailed analysis, you can use Performance Insights.

## Detecting connection churn with Performance Insights

You can use Performance Insights to assess the amount of connection churn on your Aurora PostgreSQL-Compatible Edition DB cluster. When you create an Aurora PostgreSQL DB cluster, the setting for Performance Insights is turned on by default. If you cleared this choice when you created your DB cluster, modify your cluster to turn on the feature. For more information, see [Modifying an Amazon Aurora DB cluster \(p. 248\)](#).

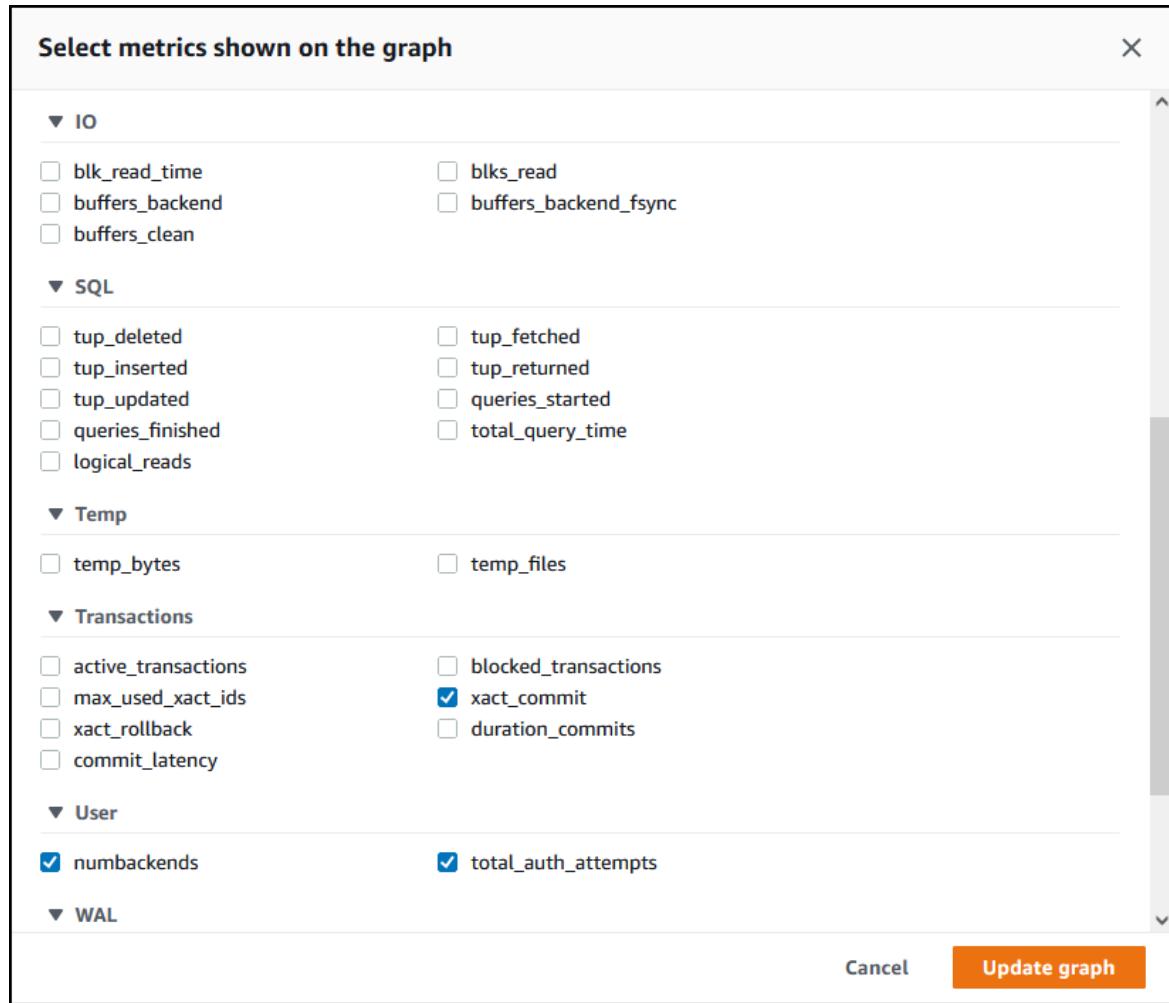
With Performance Insights running on your Aurora PostgreSQL DB cluster, you can choose the metrics that you want to monitor. You can access Performance Insights from the navigation pane in the console. You can also access Performance Insights from the **Monitoring** tab of the writer instance for your Aurora PostgreSQL DB cluster, as shown in the following image.



From the Performance Insights console, choose **Manage metrics**. To analyze your Aurora PostgreSQL DB cluster's connection and disconnection activity, choose the following metrics. These are all metrics from PostgreSQL.

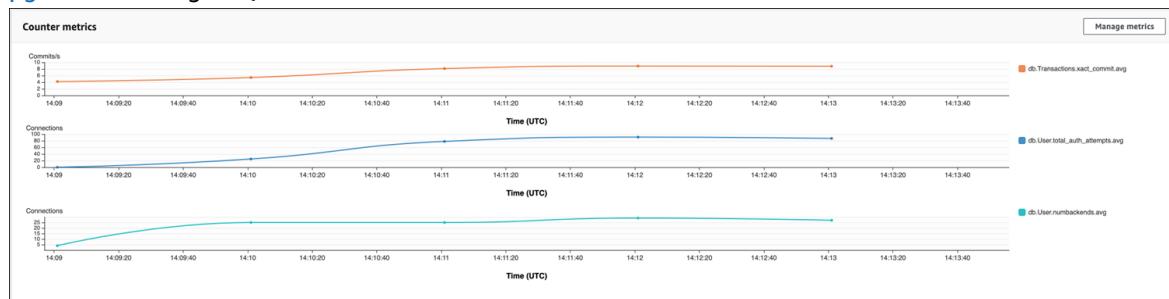
- **xact\_commit** – The number of committed transactions.
- **total\_auth\_attempts** – The number of attempted authenticated user connections.

- numbackends – The number of backends currently connected to the database.



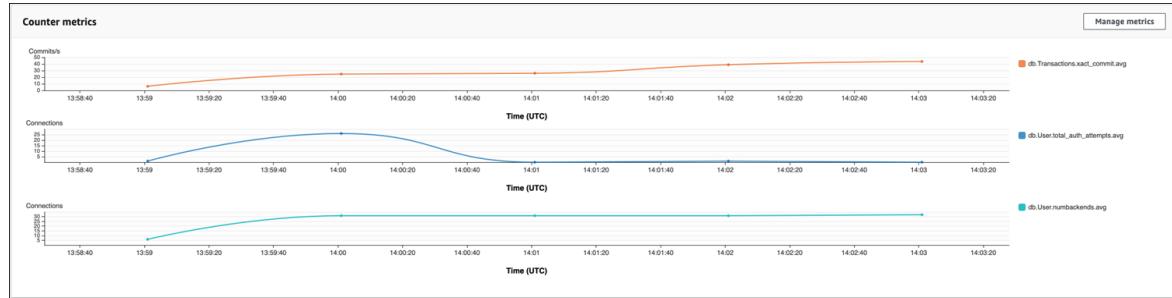
To save the settings and display connection activity, choose **Update graph**.

In the following image, you can see the impact of running pgbench with 100 users. The line showing connections is on a consistent upward slope. To learn more about pgbench and how to use it, see [pgbench](#) in PostgreSQL documentation.



The image shows that running a workload with as few as 100 users without a connection pooler can cause a significant increase in the number of `total_auth_attempts` throughout the duration of workload processing.

With RDS Proxy connection pooling, the connection attempts increase at the start of the workload. After setting up the connection pool, the average declines. The resources used by transactions and backend use stays consistent throughout workload processing.



For more information about using Performance Insights with your Aurora PostgreSQL DB cluster, see [Monitoring DB load with Performance Insights on Amazon Aurora \(p. 461\)](#). To analyze the metrics, see [Analyzing metrics with the Performance Insights dashboard \(p. 476\)](#).

## Demonstrating the benefits of connection pooling

As mentioned previously, if you determine that your Aurora PostgreSQL DB cluster has a connection churn problem, you can use RDS Proxy for improved performance. Following, you can find an example that shows the differences in processing a workload when connections are pooled and when they're not. The example uses pgbench to model a transaction workload.

As is psql, pgbench is a PostgreSQL client application that you can install and run from your local client machine. You can also install and run it from the Amazon EC2 instance that you use for managing your Aurora PostgreSQL DB cluster. For more information, see [pgbench](#) in the PostgreSQL documentation.

To step through this example, you first create the pgbench environment in your database. The following command is the basic template for initializing the pgbench tables in the specified database. This example uses the default main user account, `postgres`, for the login. Change it as needed for your Aurora PostgreSQL DB cluster. You create the pgbench environment in a database on the writer instance of your cluster.

### Note

The pgbench initialization process drops and recreates tables named `pgbench_accounts`, `pgbench_branches`, `pgbench_history`, and `pgbench_tellers`. Be sure that the database that you choose for `dbname` when you initialize pgbench doesn't use these names.

```
pgbench -U postgres -h db-cluster-instance-1.111122223333.aws-region.rds.amazonaws.com -p 5432 -d -i -s 50 dbname
```

For pgbench, specify the following parameters.

**-d**

Outputs a debugging report as pgbench runs.

**-h**

Specifies the endpoint of the Aurora PostgreSQL DB cluster's writer instance.

**-i**

Initializes the pgbench environment in the database for the benchmark tests.

**-p**

Identifies the port used for database connections. The default for Aurora PostgreSQL is typically 5432 or 5433.

**-s**

Specifies the scaling factor to use for populating the tables with rows. The default scaling factor is 1, which generates 1 row in the pgbench\_branches table, 10 rows in the pgbench\_tellers table, and 100000 rows in the pgbench\_accounts table.

**-U**

Specifies the user account for the Aurora PostgreSQL DB cluster's writer instance.

After the pgbench environment is set up, you can then run benchmarking tests with and without connection pooling. The default test consists of a series of five SELECT, UPDATE, and INSERT commands per transaction that run repeatedly for the time specified. You can specify scaling factor, number of clients, and other details to model your own use cases.

As an example, the command that follows runs the benchmark for 60 seconds (-T option, for time) with 20 concurrent connections (the -c option). The -C option makes the test run using a new connection each time, rather than once per client session. This setting gives you an indication of the connection overhead.

```
pgbench -h docs-lab-apg-133-test-instance-1.c3zr2auzukpa.us-west-1.rds.amazonaws.com -U
postgres -p 5432 -T 60 -c 20 -C labdb
Password:*****
pgbench (14.3, server 13.3)
      starting vacuum...end.
      transaction type: <builtin: TPC-B (sort of)>
      scaling factor: 50
      query mode: simple
      number of clients: 20
      number of threads: 1
      duration: 60 s
      number of transactions actually processed: 495
      latency average = 2430.798 ms
      average connection time = 120.330 ms
      tps = 8.227750 (including reconnection times)
```

Running pgbench on the writer instance of an Aurora PostgreSQL DB cluster without reusing connections shows that only about 8 transactions are processed each second. This gives a total of 495 transactions during the 1-minute test.

If you reuse connections, the response from Aurora PostgreSQL DB cluster for the number of users is almost 20 times faster. With reuse, a total of 9,042 transactions is processed compared to 495 in the same amount of time and for the same number of user connections. The difference is that in the following, each connection is being reused.

```
pgbench -h docs-lab-apg-133-test-instance-1.c3zr2auzukpa.us-west-1.rds.amazonaws.com -U
postgres -p 5432 -T 60 -c 20 labdb
Password:*****
pgbench (14.3, server 13.3)
      starting vacuum...end.
      transaction type: <builtin: TPC-B (sort of)>
      scaling factor: 50
      query mode: simple
      number of clients: 20
      number of threads: 1
      duration: 60 s
      number of transactions actually processed: 9042
      latency average = 127.880 ms
      initial connection time = 2311.188 ms
      tps = 156.396765 (without initial connection time)
```

This example shows you that pooling connections can significantly improve response times. For information about setting up RDS Proxy for your Aurora PostgreSQL DB cluster, see [Using Amazon RDS Proxy \(p. 1430\)](#).

## Tuning memory parameters for Aurora PostgreSQL

In Amazon Aurora PostgreSQL, you can use several parameters that control the amount of memory used for various processing tasks. If a task takes more memory than the amount set for a given parameter, Aurora PostgreSQL uses other resources for processing, such as by writing to disk. This can cause your Aurora PostgreSQL DB cluster to slow or potentially halt, with an out-of-memory error.

The default setting for each memory parameter can usually handle its intended processing tasks. However, you can also tune your Aurora PostgreSQL DB cluster's memory-related parameters. You do this tuning to ensure that enough memory is allocated for processing your specific workload.

Following, you can find information about parameters that control memory management. You can also learn how to assess memory utilization.

### Checking and setting parameter values

The parameters that you can set to manage memory and assess your Aurora PostgreSQL DB cluster's memory usage include the following:

- `work_mem` – Specifies the amount of memory that the Aurora PostgreSQL DB cluster uses for internal sort operations and hash tables before it writes to temporary disk files.
- `log_temp_files` – Logs temporary file creation, file names, and sizes. When this parameter is turned on, a log entry is stored for each temporary file that gets created. Turn this on to see how frequently your Aurora PostgreSQL DB cluster needs to write to disk. Turn it off again after you've gathered information about your Aurora PostgreSQL DB cluster's temporary file generation, to avoid excessive logging.
- `logical_decoding_work_mem` – Specifies the amount of memory (in megabytes) to use for logical decoding. *Logical decoding* is the process used to create a replica. This process is done by converting data from the write-ahead log (WAL) file to the logical streaming output needed by the target.

The value of this parameter creates a single buffer of the size specified for each replication connection. By default, it's 64 megabytes. After this buffer is filled, the excess is written to disk as a file. To minimize disk activity, you can set the value of this parameter to a much higher value than that of `work_mem`.

These are all dynamic parameters, so you can change them for the current session. To do this, connect to the Aurora PostgreSQL DB cluster with `psql` and using the `SET` statement, as shown following.

```
SET parameter_name TO parameter_value;
```

Session settings last for the duration of the session only. When the session ends, the parameter reverts to its setting in the DB cluster parameter group. Before changing any parameters, first check the current values by querying the `pg_settings` table, as follows.

```
SELECT unit, setting, max_val
  FROM pg_settings WHERE name='parameter_name';
```

For example, to find the value of the `work_mem` parameter, connect to the Aurora PostgreSQL DB cluster's writer instance and run the following query.

```
SELECT unit, setting, max_val, pg_size.pretty(max_val::numeric)
  FROM pg_settings WHERE name='work_mem';
unit | setting | max_val | pg_size.pretty
-----+-----+-----+
kB   | 1024   | 2147483647| 2048 MB
(1 row)
```

Changing parameter settings so that they persist requires using a custom DB cluster parameter group. After exercising your Aurora PostgreSQL DB cluster with different values for these parameters using the `SET` statement, you can create a custom parameter group and apply to your Aurora PostgreSQL DB cluster. For more information, see [Working with parameter groups \(p. 215\)](#).

## Understanding the working memory parameter

The working memory parameter (`work_mem`) specifies the maximum amount of memory that Aurora PostgreSQL can use to process complex queries. Complex queries include those that involve sorting or grouping operations—in other words, queries that use the following clauses:

- ORDER BY
- DISTINCT
- GROUP BY
- JOIN (MERGE and HASH)

The query planner indirectly affects how your Aurora PostgreSQL DB cluster uses working memory. The query planner generates execution plans for processing SQL statements. A given plan might break up a complex query into multiple units of work that can be run in parallel. When possible, Aurora PostgreSQL uses the amount of memory specified in the `work_mem` parameter for each session before writing to disk for each parallel process.

Multiple database users running multiple operations concurrently and generating multiple units of work in parallel can exhaust your Aurora PostgreSQL DB cluster's allocated working memory. This can lead to excessive temporary file creation and disk I/O, or worse, it can lead to an out-of-memory error.

## Identifying temporary file use

Whenever the memory required to process queries exceeds the value specified in the `work_mem` parameter, the working data is offloaded to disk in a temporary file. You can see how often this occurs by turning on the `log_temp_files` parameter. By default, this parameter is off (it's set to -1). To capture all temporary file information, set this parameter to 0. Set `log_temp_files` to any other positive integer to capture temporary file information for files equal to or greater than that amount of data (in kilobytes). In the following image, you can see an example from AWS Management Console.

	Name	Values	Allowed values	Modifiable	Source
	log_temp_files	1024	-1-2147483647	true	user

After configuring temporary file logging, you can test with your own workload to see if your working memory setting is sufficient. You can also simulate a workload by using pgbench, a simple benchmarking application from the PostgreSQL community.

The following example initializes (-i) pgbench by creating the necessary tables and rows for running the tests. In this example, the scaling factor (-s 50) creates 50 rows in the pgbench\_branches table, 500 rows in pgbench\_tellers, and 5,000,000 rows in the pgbench\_accounts table in the labdb database.

```
pgbench -U postgres -h your-cluster-instance-1.111122223333.aws-regionrds.amazonaws.com -p 5432 -i -s 50 labdb
Password:
dropping old tables...
NOTICE: table "pgbench_accounts" does not exist, skipping
NOTICE: table "pgbench_branches" does not exist, skipping
NOTICE: table "pgbench_history" does not exist, skipping
NOTICE: table "pgbench_tellers" does not exist, skipping
creating tables...
generating data (client-side)...
5000000 of 5000000 tuples (100%) done (elapsed 15.46 s, remaining 0.00 s)
vacuuming...
creating primary keys...
done in 61.13 s (drop tables 0.08 s, create tables 0.39 s, client-side generate 54.85 s,
vacuum 2.30 s, primary keys 3.51 s)
```

After initializing the environment, you can run the benchmark for a specific time (-T) and the number of clients (-c). This example also uses the -d option to output debugging information as the transactions are processed by the Aurora PostgreSQL DB cluster.

```
pgbench -h -U postgres your-cluster-instance-1.111122223333.aws-regionrds.amazonaws.com -p 5432 -d -T 60 -c 10 labdb
Password:*****
pgbench (14.3)
starting vacuum...end.
transaction type: <builtin: TPC-B (sort of)>
scaling factor: 50
query mode: simple
number of clients: 10
number of threads: 1
duration: 60 s
```

```
number of transactions actually processed: 1408
latency average = 398.467 ms
initial connection time = 4280.846 ms
tps = 25.096201 (without initial connection time)
```

For more information about pgbench, see [pgbench](#) in the PostgreSQL documentation.

You can use the psql metacommand command (\d) to list the relations such as tables, views, and indexes created by pgbench.

```
labdb=> \d pgbench_accounts
Table "public.pgbench_accounts"
 Column | Type | Collation | Nullable | Default
-----+-----+-----+-----+-----+
 aid | integer | | not null |
 bid | integer | | |
 abalance | integer | | |
 filler | character(84) | | |
Indexes:
"pgbench_accounts_pkey" PRIMARY KEY, btree (aid)
```

As shown in the output, the pgbench\_accounts table is indexed on the aid column. To ensure that this next query uses working memory, query any nonindexed column, such as that shown in the following example.

```
postgres=> SELECT * FROM pgbench_accounts ORDER BY bid;
```

Check the log for the temporary files. To do so, open the AWS Management Console, choose the Aurora PostgreSQL DB cluster instance, and then choose the **Logs & Events** tab. View the logs in the console or download for further analysis. As shown in the following image, the size of the temporary files needed to process the query indicates that you should consider increasing the amount specified for the `work_mem` parameter.

```
2022-07-07 23:00:02 UTC:[local]:[unknown]@[unknown]:[9698]:LOG: connection received: host=[local]
2022-07-07 23:02:02 UTC:[local]:[unknown]@[unknown]:[15780]:LOG: connection received: host=[local]
2022-07-07 23:04:02 UTC:[local]:[unknown]@[unknown]:[21216]:LOG: connection received: host=[local]
2022-07-07 23:04:16 UTC:@:[18585]:LOG: temporary file: path "base/pgsql_tmp/pgsql_tmp18585.0", size
2022-07-07 23:04:16 UTC:@:[18585]:STATEMENT: SELECT * from pgbench_accounts ORDER by bid;
2022-07-07 23:04:16 UTC:@:[18586]:LOG: temporary file: path "base/pgsql_tmp/pgsql_tmp18586.0", size
2022-07-07 23:04:16 UTC:@:[18586]:STATEMENT: SELECT * from pgbench_accounts ORDER by bid;
2022-07-07 23:04:16 UTC:54.240.198.34(12096):postgres@labdb:[5700]:LOG: temporary file: path "base/pg
2022-07-07 23:04:16 UTC:54.240.198.34(12096):postgres@labdb:[5700]:STATEMENT: SELECT * from pgbench_a
2022-07-07 23:06:02 UTC:[local]:[unknown]@[unknown]:[26796]:LOG: connection received: host=[local]
2022-07-07 23:08:02 UTC:[local]:[unknown]@[unknown]:[331]:LOG: connection received: host=[local]
2022-07-07 23:10:02 UTC:[local]:[unknown]@[unknown]:[5938]:LOG: connection received: host=[local]
2022-07-07 23:12:02 UTC:[local]:[unknown]@[unknown]:[11851]:LOG: connection received: host=[local]
2022-07-07 23:14:02 UTC:[local]:[unknown]@[unknown]:[17375]:LOG: connection received: host=[local]
2022-07-07 23:16:02 UTC:[local]:[unknown]@[unknown]:[22962]:LOG: connection received: host=[local]
2022-07-07 23:18:02 UTC:[local]:[unknown]@[unknown]:[28804]:LOG: connection received: host=[local]
2022-07-07 23:20:02 UTC:[local]:[unknown]@[unknown]:[2012]:LOG: connection received: host=[local]
2022-07-07 23:22:02 UTC:[local]:[unknown]@[unknown]:[180001]:LOG: connection received: host=[local]
```

You can configure this parameter differently for individuals and groups, based on your operational needs. For example, you can set the `work_mem` parameter to 8 GB for the role named `dev_team`.

```
postgres=> ALTER ROLE dev_team SET work_mem='8GB';
```

With this setting for `work_mem`, any role that's a member of the `dev_team` role is allotted up to 8 GB of working memory.

## Using indexes for faster response time

If your queries are taking too long to return results, you can verify that your indexes are being used as expected. First, turn on `\timing`, the `psql` metacommand, as follows.

```
postgres=> \timing on
```

After turning on timing, use a simple `SELECT` statement.

```
postgres=> SELECT COUNT(*) FROM
  (SELECT * FROM pgbench_accounts
   ORDER BY bid)
   AS accounts;
count
-----
5000000
(1 row)
Time: 3119.049 ms (00:03.119)
```

As shown in the output, this query took just over 3 seconds to complete. To improve the response time, create an index on `pgbench_accounts`, as follows.

```
postgres=> CREATE INDEX ON pgbench_accounts(bid);
CREATE INDEX
```

Rerun the query, and notice the faster response time. In this example, the query completed about 5 times faster, in about half a second.

```
postgres=> SELECT COUNT(*) FROM (SELECT * FROM pgbench_accounts ORDER BY bid) AS accounts;
count
-----
5000000
(1 row)
Time: 567.095 ms
```

## Adjusting working memory for logical decoding

Logical replication has been available in all versions of Aurora PostgreSQL since its introduction in PostgreSQL version 10. When you configure logical replication, you can also set the `logical_decoding_work_mem` parameter to specify the amount of memory that the logical decoding process can use for the decoding and streaming process.

During logical decoding, write-ahead log (WAL) records are converted to SQL statements that are then sent to another target for logical replication or another task. When a transaction is written to the WAL and then converted, the entire transaction must fit into the value specified for `logical_decoding_work_mem`. By default, this parameter is set to 64 MB. Any overflow is written to disk. This means that it must be reread from the disk before it can be sent to its destination, thus slowing the overall process.

You can assess the amount of transaction overflow in your current workload at a specific point in time by using the `aurora_stat_file` function as shown in the following example.

```
SELECT split_part (filename, '/', 2)
  AS slot_name, count(1) AS num_spill_files,
  sum(used_bytes) AS slot_total_bytes,
```

```

pg_size.pretty(sum(used_bytes)) AS slot_total_size
FROM aurora_stat_file()
WHERE filename like '%spill%'
GROUP BY 1;
slot_name | num_spill_files | slot_total_bytes | slot_total_size
-----+-----+-----+-----+
slot_name |      590      | 411600000 | 393 MB
(1 row)

```

This query returns the count and size of spill files on your Aurora PostgreSQL DB cluster when the query is invoked. Longer running workloads might not have any spill files on disk yet. To profile long-running workloads, we recommend that you create a table to capture the spill file information as the workload runs. You can create the table as follows.

```

CREATE TABLE spill_file_tracking AS
SELECT now() AS spill_time,*
FROM aurora_stat_file()
WHERE filename LIKE '%spill%';

```

To see how spill files are used during logical replication, set up a publisher and subscriber and then start a simple replication. For more information, see [Configuring logical replication \(p. 1225\)](#). With replication under way, you can create a job that captures the result set from the `aurora_stat_file()` spill file function, as follows.

```

INSERT INTO spill_file_tracking
SELECT now(),*
FROM aurora_stat_file()
WHERE filename LIKE '%spill%';

```

Use the following `psql` command to run the job once per second.

```
\watch 0.5
```

As the job is running, connect to the writer instance from another `psql` session. Use the following series of statements to run a workload that exceeds the memory configuration and causes Aurora PostgreSQL to create a spill file.

```

labdb=> CREATE TABLE my_table (a int PRIMARY KEY, b int);
CREATE TABLE
labdb=> INSERT INTO my_table SELECT x,x FROM generate_series(0,10000000) x;
INSERT 0 10000001
labdb=> UPDATE my_table SET b=b+1;
UPDATE 10000001

```

These statements take several minutes to complete. When finished, press the Ctrl key and the C key together to stop the monitoring function. Then use the following command to create a table to hold the information about the Aurora PostgreSQL DB cluster's spill file usage.

```

SELECT spill_time, split_part (filename, '/', 2)
AS slot_name, count(1)
AS spills, sum(used_bytes)
AS slot_total_bytes, pg_size.pretty(sum(used_bytes))
AS slot_total_size FROM spill_file_tracking
GROUP BY 1,2 ORDER BY 1;
spill_time | slot_name | spills | slot_total_bytes |
slot_total_size
-----+-----+-----+-----+
2022-04-15 13:42:52.528272+00 | replication_slot_name | 1 | 142352280 | 136 MB

```

2022-04-15 14:11:33.962216+00	replication_slot_name   4	467637996	446 MB
2022-04-15 14:12:00.997636+00	replication_slot_name   4	569409176	543 MB
2022-04-15 14:12:03.030245+00	replication_slot_name   4	569409176	543 MB
2022-04-15 14:12:05.059761+00	replication_slot_name   5	618410996	590 MB
2022-04-15 14:12:07.22905+00	replication_slot_name   5	640585316	611 MB
(6 rows)			

The output shows that running the example created five spill files that used 611 MB of memory. To avoid writing to disk, we recommend setting the `logical_decoding_work_mem` parameter to the next highest memory size, 1024.

## 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 \(p. 1224\)](#)
- [Monitoring Aurora PostgreSQL replication \(p. 1224\)](#)
- [Using PostgreSQL logical replication with Aurora \(p. 1225\)](#)

## 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 \(p. 73\)](#).

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 \(p. 427\)](#).

## 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 \(p. 1225\)](#)
- [Example of logical replication of a database table \(p. 1227\)](#)
- [Logical replication using the AWS Database Migration Service \(p. 1228\)](#)
- [Stopping logical replication \(p. 1229\)](#)

## 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. The replication process uses a *replication slot* to track the progress of a subscription.

### Note

Following are requirements for logical replication:

- To set up logical replication, you need to have `rds_superuser` permissions.
- The source instance (publisher node) and the receiving instance (subscriber node) must each have automated backups turned on. For more information, 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 \(p. 219\)](#). 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 \(p. 221\)](#). 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 \(p. 221\)](#).

Follow these guidelines for setting the parameters:

- `max_replication_slots` – A *replication slot* tracks the progress of a subscription. Set the value of the `max_replication_slots` parameter to the total number of subscriptions that you plan to create. If you are using AWS DMS, set this parameter to the number of AWS DMS tasks that you plan to use for change data capture from this DB cluster.
- `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 the 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 \(p. 248\)](#).
    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 \(p. 131\)](#).
    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 \(p. 1225\)](#).
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 \(p. 1227\)](#).

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 \(p. 221\)](#). 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 \(p. 461\)](#).
- 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 \(p. 1265\)](#).
- 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 data from Amazon S3 into an Aurora PostgreSQL DB cluster \(p. 1230\)](#) and [Exporting data from an Aurora PostgreSQL DB cluster to Amazon S3 \(p. 1243\)](#).
- 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 \(p. 1272\)](#).
- 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 \(p. 1254\)](#).
- 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 data from Amazon S3 into an Aurora PostgreSQL DB cluster

You can import data that's been stored using Amazon Simple Storage Service into a table on an Aurora PostgreSQL DB cluster instance. To do this, you first install the Aurora PostgreSQL `aws_s3` extension. This extension provides the functions that you use to import data from an Amazon S3 bucket. A *bucket*

is an Amazon S3 container for objects and files. The data can be in a comma-separated value (CSV) file, a text file, or a compressed (gzip) file. Following, you can learn how to install the extension and how to import data from Amazon S3 into a table.

Your database must be running PostgreSQL version 10.7 or higher to import from Amazon S3 into Aurora PostgreSQL.

If you don't have data stored on Amazon S3, you need to first create a bucket and store the data. For more information, see the following topics in the *Amazon Simple Storage Service User Guide*.

- [Create a bucket](#)
- [Add an object to a bucket](#)

**Note**

Importing data from Amazon S3 isn't supported for Aurora Serverless v1. It is supported for Aurora Serverless v2.

**Topics**

- [Installing the aws\\_s3 extension \(p. 1231\)](#)
- [Overview of importing data from Amazon S3 data \(p. 1232\)](#)
- [Setting up access to an Amazon S3 bucket \(p. 1233\)](#)
- [Importing data from Amazon S3 to your Aurora PostgreSQL DB cluster \(p. 1238\)](#)
- [Function reference \(p. 1240\)](#)

## Installing the aws\_s3 extension

Before you can use Amazon S3 with your Aurora PostgreSQL DB cluster, you need to install the `aws_s3` extension. This extension provides functions for importing data from an Amazon S3. It also provides functions for exporting data from an instance of an Aurora PostgreSQL DB cluster to an Amazon S3 bucket. For more information, see [Exporting data from an Aurora PostgreSQL DB cluster to Amazon S3 \(p. 1243\)](#). The `aws_s3` extension depends on some of the helper functions in the `aws_commons` extension, which is installed automatically when needed.

### To install the aws\_s3 extension

1. Use `psql` (or `pgAdmin`) to connect to the writer instance of your Aurora PostgreSQL DB cluster as a user that has `rds_superuser` privileges. If you kept the default name during the setup process, you connect as `postgres`.

```
psql --host=111122223333.aws-region.rds.amazonaws.com --port=5432 --username=postgres  
--password
```

2. To install the extension, run the following command.

```
postgres=> CREATE EXTENSION aws_s3 CASCADE;  
NOTICE: installing required extension "aws_commons"  
CREATE EXTENSION
```

3. To verify that the extension is installed, you can use the `\dx` metacommand.

```
postgres=> \dx  
      List of installed extensions  
 Name | Version | Schema | Description  
-----+-----+-----+  
aws_commons | 1.2 | public | Common data types across AWS services
```

```
aws_s3      | 1.1      | public      | AWS S3 extension for importing data from S3
plpgsql     | 1.0      | pg_catalog  | PL/pgSQL procedural language
(3 rows)
```

The functions for importing data from Amazon S3 and exporting data to Amazon S3 are now available to use.

## Overview of importing data from Amazon S3 data

### To import S3 data into Aurora PostgreSQL

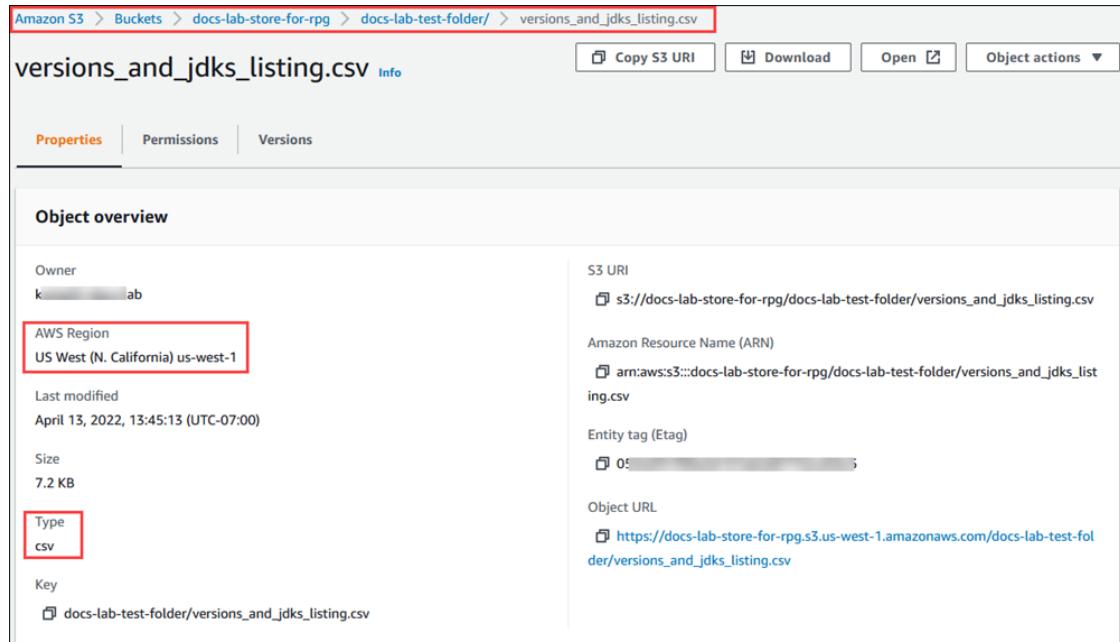
First, gather the details that you need to supply to the function. These include the name of the table on your Aurora PostgreSQL DB cluster's instance, and the bucket name, file path, file type, and AWS Region where the Amazon S3 data is stored. For more information, see [View an object](#) in the *Amazon Simple Storage Service User Guide*.

1. Get the name of the table into which the `aws_s3.table_import_from_s3` function is to import the data. As an example, the following creates a database `t1` that can be used in later steps.

```
postgres=> CREATE TABLE t1
  (col1 varchar(80),
   col2 varchar(80),
   col3 varchar(80));
```

2. Get the details about the Amazon S3 bucket and the data to import. To do this, open the Amazon S3 console at <https://console.aws.amazon.com/s3/>, and choose **Buckets**. Find the bucket containing your data in the list. Choose the bucket, open its Object overview page, and then choose Properties.

Make a note of the bucket name, path, the AWS Region, and file type. You need the Amazon Resource Name (ARN) later, to set up access to Amazon S3 through an IAM role. For more information, see [Setting up access to an Amazon S3 bucket \(p. 1233\)](#). The image following shows an example.



3. You can verify the path to the data on the Amazon S3 bucket 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 ./
```

4. Set up permissions on your Aurora PostgreSQL DB cluster to allow access to the file on the Amazon S3 bucket. To do so, 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 \(p. 1233\)](#).
5. Supply the path and other Amazon S3 object details gathered (see step 2) to the `create_s3_uri` function to construct an Amazon S3 URI object. To learn more about this function, see [aws\\_commons.create\\_s3\\_uri \(p. 1242\)](#). The following is an example of constructing this object during a `psql` session.

```
postgres=> SELECT aws_commons.create_s3_uri(  
    'docs-lab-store-for-rpg',  
    'versions_and_jdks_listing.csv',  
    'us-west-1'  
) AS s3_uri \gset
```

In the next step, you pass this object (`aws_commons._s3_uri_1`) to the `aws_s3.table_import_from_s3` function to import the data to the table.

6. Invoke the `aws_s3.table_import_from_s3` function to import the data from Amazon S3 into your table. For reference information, see [aws\\_s3.table\\_import\\_from\\_s3 \(p. 1240\)](#). For examples, see [Importing data from Amazon S3 to your Aurora PostgreSQL DB cluster \(p. 1238\)](#).

## 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 \(p. 1233\)](#)
- [Using security credentials to access an Amazon S3 bucket \(p. 1237\)](#)
- [Troubleshooting access to Amazon S3 \(p. 1238\)](#)

## 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. 1240)` 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.

### Note

You can't associate an IAM role with an Aurora Serverless v1 DB cluster, so the following steps don't apply.

### 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 \(p. 1686\)](#). 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**

Make a note of the Amazon Resource Name (ARN) of the policy returned by this command. You need the ARN in a subsequent step when you attach the policy to an IAM role.

### 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/*"
            ]
        }
    ]
}'
```

```
    "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.

### 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. You do so by specifying the `credentials` parameter in the [aws\\_s3.table\\_import\\_from\\_s3](#) (p. 1240) 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. 1243) function to set the access key and secret key in an `aws_commons._aws_credentials_1` structure, as shown following.

```
postgres=> 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. 1240) function with the `credentials` parameter to import the data, as shown following.

```
postgres=> SELECT aws_s3.table_import_from_s3(
  't', '', '(format csv)',
  :'s3_uri',
  :'creds'
);
```

Or you can include the [aws\\_commons.create\\_aws\\_credentials](#) (p. 1243) function call inline within the `aws_s3.table_import_from_s3` function call.

```
postgres=> 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 data from Amazon S3, see the following for recommendations:

- [Troubleshooting Amazon Aurora identity and access \(p. 1710\)](#)
- [Troubleshooting Amazon S3 in the Amazon Simple Storage Service User Guide](#)
- [Troubleshooting Amazon S3 and IAM in the IAM User Guide](#)

## Importing data from Amazon S3 to your Aurora PostgreSQL DB cluster

You import data from your Amazon S3 bucket by using the `table_import_from_s3` function of the `aws_s3` extension. For reference information, see [aws\\_s3.table\\_import\\_from\\_s3 \(p. 1240\)](#).

### Note

The following examples use the IAM role method to allow 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 example.

```
postgres=> 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 \(p. 1239\)](#).
- `(format csv)` – PostgreSQL COPY arguments. The copy process uses the arguments and format of the [PostgreSQL COPY](#) command to import the data. Choices for format include comma-separated value (CSV) as shown in this example, text, and binary. The default is text.
- `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. 1242)` function to create an `s3_uri` structure, see [Overview of importing data from Amazon S3 data \(p. 1232\)](#).

For more information about this function, see [aws\\_s3.table\\_import\\_from\\_s3 \(p. 1240\)](#).

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 \(p. 1239\)](#)
- [Importing an Amazon S3 compressed \(gzip\) file \(p. 1239\)](#)
- [Importing an encoded Amazon S3 file \(p. 1240\)](#)

## 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 \(p. 1240\)](#) 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.

```
postgres=> 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 \(p. 1240\)](#) function to import data from the Amazon S3 file.

You can include the [aws\\_commons.create\\_s3\\_uri \(p. 1242\)](#) function call inline within the `aws_s3.table_import_from_s3` function call to specify the file.

```
postgres=> 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.

```
postgres=> 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.

```
postgres=> CREATE TABLE test_gzip(id int, a text, b text, c text, d text);
postgres=> 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.

```
postgres=> 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 \(p. 1240\)](#)
- [aws\\_commons.create\\_s3\\_uri \(p. 1242\)](#)
- [aws\\_commons.create\\_aws\\_credentials \(p. 1243\)](#)

### 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 \(p. 1239\)](#).

### *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 \(p. 1243\)](#).

## 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 identifying the AWS Region location of the file. 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 \(p. 1240\)](#) 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 \(p. 1240\)](#) 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 first install the Aurora PostgreSQL `aws_s3` extension. This extension provides you with the functions that you use to export the results of queries to Amazon S3. Following, you can find out how to install the extension and how to export data to Amazon S3.

You can export from a provisioned or an Aurora Serverless v2 DB instance. These steps aren't supported for Aurora Serverless v1.

All currently available versions of Aurora PostgreSQL support exporting data to Amazon Simple Storage Service. For detailed version information, see [Amazon Aurora PostgreSQL updates in the Release Notes for Aurora PostgreSQL](#).

If you don't have a bucket set up for your export, see the following topics the *Amazon Simple Storage Service User Guide*.

- [Setting up Amazon S3](#)
- [Create a bucket](#)

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 cluster snapshot data to Amazon S3 \(p. 396\)](#).

**Topics**

- [Installing the aws\\_s3 extension \(p. 1244\)](#)
- [Overview of exporting data to Amazon S3 \(p. 1245\)](#)
- [Specifying the Amazon S3 file path to export to \(p. 1245\)](#)
- [Setting up access to an Amazon S3 bucket \(p. 1246\)](#)
- [Exporting query data using the aws\\_s3.query\\_export\\_to\\_s3 function \(p. 1249\)](#)
- [Troubleshooting access to Amazon S3 \(p. 1251\)](#)
- [Function reference \(p. 1252\)](#)

## Installing the aws\_s3 extension

Before you can use Amazon Simple Storage Service with your Aurora PostgreSQL DB cluster, you need to install the aws\_s3 extension. This extension provides functions for exporting data from the writer instance of an Aurora PostgreSQL DB cluster to an Amazon S3 bucket. It also provides functions for importing data from an Amazon S3. For more information, see [Importing data from Amazon S3 into an Aurora PostgreSQL DB cluster \(p. 1230\)](#). The aws\_s3 extension depends on some of the helper functions in the aws\_commons extension, which is installed automatically when needed.

### To install the aws\_s3 extension

1. Use psql (or pgAdmin) to connect to the writer instance of your Aurora PostgreSQL DB cluster as a user that has rds\_superuser privileges. If you kept the default name during the setup process, you connect as postgres.

```
psql --host=111122223333.aws-region.rds.amazonaws.com --port=5432 --username=postgres  
--password
```

2. To install the extension, run the following command.

```
postgres=> CREATE EXTENSION aws_s3 CASCADE;  
NOTICE: installing required extension "aws_commons"  
CREATE EXTENSION
```

3. To verify that the extension is installed, you can use the psql \dx metacommand.

```
postgres=> \dx
```

List of installed extensions			
Name	Version	Schema	Description
aws_commons	1.2	public	Common data types across AWS services
aws_s3	1.1	public	AWS S3 extension for importing data from S3
plpgsql	1.0	pg_catalog	PL/pgSQL procedural language
(3 rows)			

The functions for importing data from Amazon S3 and exporting data to Amazon S3 are now available to use.

## Verify that your Aurora PostgreSQL version supports exports to Amazon S3

You can verify that your Aurora PostgreSQL version supports export to Amazon S3 by using the `describe-db-engine-versions` command. The following example checks to see if version 10.14 can export to Amazon S3.

```
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.

## 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. 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 \(p. 1245\)](#).
2. 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 \(p. 1246\)](#).

3. 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 \(p. 1252\)](#) 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 \(p. 1249\)](#).

## 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 \(p. 1254\)](#) function to create an `aws_commons._s3_uri_1` composite structure as follows.

```
psql=> SELECT aws_commons.create_s3_uri(
    'sample-bucket',
    'sample-filepath',
    '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 \(p. 1252\)](#) function. For examples, see [Exporting query data using the aws\\_s3.query\\_export\\_to\\_s3 function \(p. 1249\)](#).

## 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 \(p. 1686\)](#). 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. 1252) function.

### Topics

- [Prerequisites](#) (p. 1250)
- [Calling aws\\_s3.query\\_export\\_to\\_s3](#) (p. 1250)
- [Exporting to a CSV file that uses a custom delimiter](#) (p. 1251)
- [Exporting to a binary file with encoding](#) (p. 1251)

## 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 \(p. 1245\)](#).
- Determine where to export your data to Amazon S3 as described in [Specifying the Amazon S3 file path to export to \(p. 1245\)](#).
- Make sure that the DB cluster has export access to Amazon S3 as described in [Setting up access to an Amazon S3 bucket \(p. 1246\)](#).

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 \(p. 1252\)](#) 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 \(p. 1254\)](#) function to create the structure.

```
psql=> SELECT aws_commons.create_s3_uri(
    'sample-bucket',
    'sample-filepath',
    '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', 'sample-filepath', '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 `_partXX` 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 \(p. 1252\)](#). 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 \(p. 1252\)](#) 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 \(p. 1252\)](#) 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, the security group must have a rule that allows the DB instance to send TCP traffic to port 443 and to any IPv4 addresses (0.0.0.0/0). For more information, see [Provide access to the DB cluster in the VPC by creating a security group \(p. 90\)](#).

See also the following for recommendations:

- [Troubleshooting Amazon Aurora identity and access \(p. 1710\)](#)
- [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 \(p. 1252\)](#)
- [aws\\_commons.create\\_s3\\_uri \(p. 1254\)](#)

### [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 \(p. 1249\)](#).

#### 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 \(p. 1254\)](#) 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', 'samplefilepath');
psql=> SELECT * from aws_s3.query_export_to_s3('select * from sample_table', 'sample-
bucket', 'samplefilepath','us-west-2');
psql=> SELECT * from aws_s3.query_export_to_s3('select * from sample_table', 'sample-
bucket', 'samplefilepath','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 \(p. 1252\)](#) function. For an example of using the `aws_commons.create_s3_uri` function, see [Specifying the Amazon S3 file path to export to \(p. 1245\)](#).

### 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 \(p. 11\)](#).

## 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 \(p. 1255\)](#)
- [Step 2: Configure IAM for your Aurora PostgreSQL DB cluster and AWS Lambda \(p. 1255\)](#)
- [Step 3: Install the aws\\_lambda extension for an Aurora PostgreSQL DB cluster \(p. 1257\)](#)
- [Step 4: Use Lambda helper functions with your Aurora PostgreSQL DB cluster \(Optional\) \(p. 1257\)](#)
- [Step 5: Invoke a Lambda function from your Aurora PostgreSQL DB cluster \(p. 1258\)](#)
- [Step 6: Grant other users permission to invoke Lambda functions \(p. 1259\)](#)
- [Examples: Invoking Lambda functions from your Aurora PostgreSQL DB cluster \(p. 1259\)](#)
- [Lambda function error messages \(p. 1261\)](#)
- [AWS Lambda function reference \(p. 1262\)](#)

## Step 1: Configure your Aurora PostgreSQL DB cluster for outbound connections to AWS Lambda

Lambda functions always run inside an Amazon VPC that's 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 sends 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.

- **Public Aurora PostgreSQL DB cluster** – A 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`. To verify that the instance is in the public subnet of your VPC, you can use the AWS Management Console or the AWS CLI.

To set up access to Lambda, you use the AWS Management Console or the AWS CLI to create an outbound rule on your VPC's security group. The outbound rule specifies that TCP can use port 443 to send packets to any IPv4 addresses (0.0.0.0/0).

- **Private Aurora PostgreSQL DB cluster** – In this case, the instance's "PubliclyAccessible" property is `false` or it's in a private subnet. To allow the instance to work with Lambda, you can use a Network Address Translation (NAT) gateway. For more information, see [NAT gateways](#). Or, you can configure your VPC with a VPC endpoint for Lambda. For more information, see [VPC endpoints](#) in the *Amazon VPC User Guide*. The endpoint responds to calls made by your Aurora PostgreSQL DB cluster to your Lambda functions.

Your VPC can now interact with the AWS Lambda VPC at the network level. Next, you 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.444455556666.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 \(p. 1255\)](#) 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. 1264)` 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(
    '111122223333:function:my-function',
```

```
'aws-region'  
) AS lambda_partial_arn_1 \gset
```

```
SELECT aws_commons.create_lambda_function_arn(  
    'arn:aws:lambda:aws-region:111122223333:function:my-function'  
) AS lambda_arn_1 \gset
```

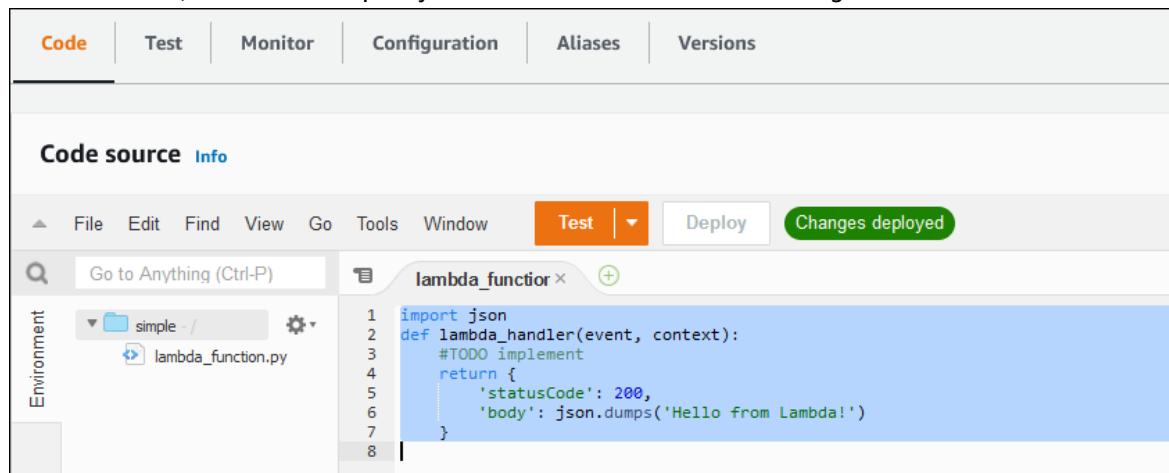
Any of these values can be used in calls to the [aws\\_lambda.invoke \(p. 1262\)](#) function. For examples, see [Step 5: Invoke a Lambda function from your Aurora PostgreSQL DB cluster \(p. 1258\)](#).

## 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.



### 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 \(p. 1261\)](#).

## Step 6: Grant other users permission to invoke Lambda functions

At this point in the procedures, only you as `rds_superuser` can invoke your Lambda functions. To allow other users to invoke any functions that you create, you need to grant them permission.

### To grant others permission to invoke Lambda functions

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. Run the following SQL commands:

```
postgres=> GRANT USAGE ON SCHEMA aws_lambda TO db_username;
GRANT EXECUTE ON ALL FUNCTIONS IN SCHEMA aws_lambda TO db_username;
```

## Examples: Invoking Lambda functions from your Aurora PostgreSQL DB cluster

Following, you can find several examples of calling the `aws_lambda.invoke` (p. 1262) 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 \(Optional\) \(p. 1257\)](#) to simplify passing the function details. For an example of asynchronous invocation, see [Example: Asynchronous \(Event\) invocation of Lambda functions \(p. 1260\)](#). 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 \(p. 1264\)](#).

### Examples list

- [Example: Synchronous \(RequestResponse\) invocation of Lambda functions \(p. 1260\)](#)
- [Example: Asynchronous \(Event\) invocation of Lambda functions \(p. 1260\)](#)
- [Example: Capturing the Lambda execution log in a function response \(p. 1260\)](#)
- [Example: Including client context in a Lambda function \(p. 1261\)](#)
- [Example: Invoking a specific version of a Lambda function \(p. 1261\)](#)

## 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 \(Optional\) \(p. 1257\)](#), 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 \(p. 1262\)](#) 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 \(p. 1262\)](#).

## 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 \(p. 1262\)](#) function's `context` parameter.

For more information about the `context` parameter, see the [aws\\_lambda.invoke \(p. 1262\)](#) 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 \(p. 1262\)](#) reference.

## Lambda function error messages

In the following list you can find information about error messages, with possible causes and solutions.

- **VPC configuration issues**

VPC configuration issues can raise the following error messages when trying to connect:

```
ERROR: invoke API failed
DETAIL: AWS Lambda client returned 'Unable to connect to endpoint'.
CONTEXT: SQL function "invoke" statement 1
```

A common cause for this error is improperly configured VPC security group. Make sure you have an outbound rule for TCP open on port 443 of your VPC security group so that your VPC can connect to the Lambda VPC.

- **Lack of permissions needed to invoke Lambda functions**

If you see either of the following error messages, the user (role) invoking the function doesn't have proper permissions.

```
ERROR: permission denied for schema aws_lambda
```

```
ERROR: permission denied for function invoke
```

A user (role) must be given specific grants to invoke Lambda functions. For more information, see [Step 6: Grant other users permission to invoke Lambda functions \(p. 1259\)](#).

- **Improper handling of errors in your Lambda functions**

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!"}');  
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.

## AWS Lambda function reference

Following is the reference for the functions to use for invoking Lambda functions with Aurora PostgreSQL .

### Functions

- [aws\\_lambda.invoke \(p. 1262\)](#)
- [aws\\_commons.create\\_lambda\\_function\\_arn \(p. 1264\)](#)

### 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. 1262](#)) 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 \(p. 11\)](#).

# Publishing Aurora PostgreSQL logs to Amazon CloudWatch Logs

You can configure your Aurora PostgreSQL DB cluster to export log data to Amazon CloudWatch Logs on a regular basis. When you do so, events from your Aurora PostgreSQL DB cluster's PostgreSQL log are automatically *published* to Amazon CloudWatch, as Amazon CloudWatch Logs. In CloudWatch, you can find the exported log data in a *Log group* for your Aurora PostgreSQL DB cluster. The log group contains one or more *log streams* that contain the events from the PostgreSQL log from each instance in the cluster.

Publishing the logs to CloudWatch Logs allows you to keep your cluster's PostgreSQL log records in highly durable storage. With the log data available in CloudWatch Logs, you can evaluate and improve your cluster's operations. You can also use CloudWatch to create alarms and view metrics. To learn more, see [Monitoring log events in Amazon CloudWatch \(p. 1268\)](#).

## Note

Publishing your PostgreSQL logs to CloudWatch Logs consumes storage, and you incur charges for that storage. Be sure to delete any CloudWatch Logs that you no longer need.

Turning the export log option off for an existing Aurora PostgreSQL DB cluster doesn't affect any data that's already held in CloudWatch Logs. Existing logs remain available in CloudWatch Logs based on your log retention settings. To learn more about CloudWatch Logs, see [What is Amazon CloudWatch Logs?](#)

Aurora PostgreSQL supports publishing logs to CloudWatch Logs for the following versions.

- 14.3 and higher 14 versions
- 13.3 and higher 13 versions
- 12.8 and higher 12 versions
- 11.12 and higher 11 versions

## Turning on the option to publish logs to Amazon CloudWatch

To publish your Aurora PostgreSQL DB cluster's PostgreSQL log to CloudWatch Logs, choose the **Log export** option for the cluster. You can choose the Log export setting when you create your Aurora PostgreSQL DB cluster. Or, you can modify the cluster later on. When you modify an existing cluster, its PostgreSQL logs from each instance are published to CloudWatch cluster from that point on. For Aurora PostgreSQL, the PostgreSQL log (`postgresql.log`) is the only log that gets published to Amazon CloudWatch.

You can use the AWS Management Console, the AWS CLI, or the RDS API to turn on the Log export feature for your Aurora PostgreSQL DB cluster.

### Console

You choose the Log exports option to start publishing the PostgreSQL logs from your Aurora PostgreSQL DB cluster to CloudWatch Logs.

#### To turn on the Log export feature 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 whose log data you want to publish to CloudWatch Logs.
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 turn on the log export option to start publishing Aurora PostgreSQL logs to Amazon CloudWatch Logs with the AWS CLI. To do so, 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 turn on the log export option to start publishing Aurora PostgreSQL logs with the RDS API. To do so, 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

With Aurora PostgreSQL log events published and available as Amazon CloudWatch Logs, you can view and monitor events using Amazon CloudWatch. For more information about monitoring, see [View log data sent to CloudWatch Logs](#).

When you turn on Log exports, a new log group is automatically created using the prefix `/aws/rds/cluster/` with the name of your Aurora PostgreSQL and the log type, as in the following pattern.

```
/aws/rds/cluster/your-cluster-name/postgresql
```

As an example, suppose that an Aurora PostgreSQL DB cluster named `docs-lab-apg-small` exports its log to Amazon CloudWatch Logs. Its log group name in Amazon CloudWatch is shown following.

```
/aws/rds/cluster/docs-lab-apg-small/postgresql
```

If a log group with the specified name exists, Aurora uses that log group to export log data for the Aurora DB cluster. Each DB instance in the Aurora PostgreSQL DB cluster uploads its PostgreSQL log to the log group as a distinct log stream. You can examine the log group and its log streams using the various graphical and analytical tools available in Amazon CloudWatch.

For example, you can search for information within the log events from your Aurora PostgreSQL DB cluster, and filter events by using the CloudWatch Logs console, the AWS CLI, or the CloudWatch Logs API. For more information, [Searching and filtering log data](#) in the *Amazon CloudWatch Logs User Guide*.

By default, new log groups are created using **Never expire** for their retention period. You can use the CloudWatch Logs console, the AWS CLI, or the CloudWatch Logs API to change the log retention period. To learn more, see [Change log data retention in CloudWatch Logs](#) in the *Amazon CloudWatch Logs User Guide*.

### Tip

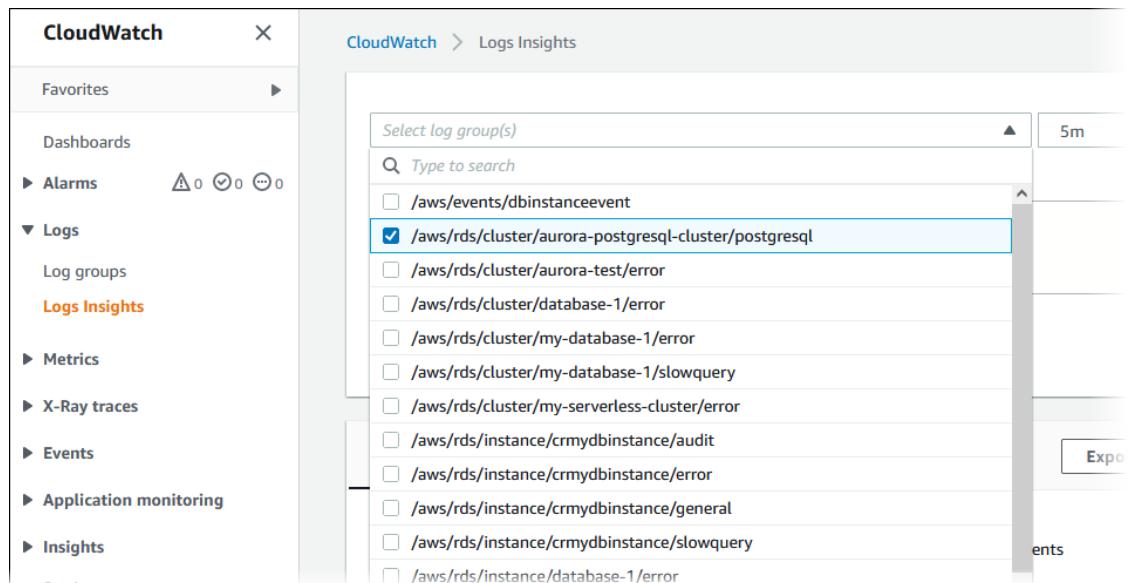
You can use automated configuration, such as AWS CloudFormation, to create log groups with predefined log retention periods, metric filters, and access permissions.

## Analyzing PostgreSQL logs using CloudWatch Logs Insights

With the PostgreSQL logs from your Aurora PostgreSQL DB cluster published as CloudWatch Logs, you can use CloudWatch Logs Insights to interactively search and analyze your log data in Amazon CloudWatch Logs. CloudWatch Logs Insights includes a query language, sample queries, and other tools for analyzing your log data so that you can identify potential issues and verify fixes. To learn more, see [Analyzing log data with CloudWatch Logs Insights](#) in the *Amazon CloudWatch Logs User Guide*. Amazon CloudWatch Logs

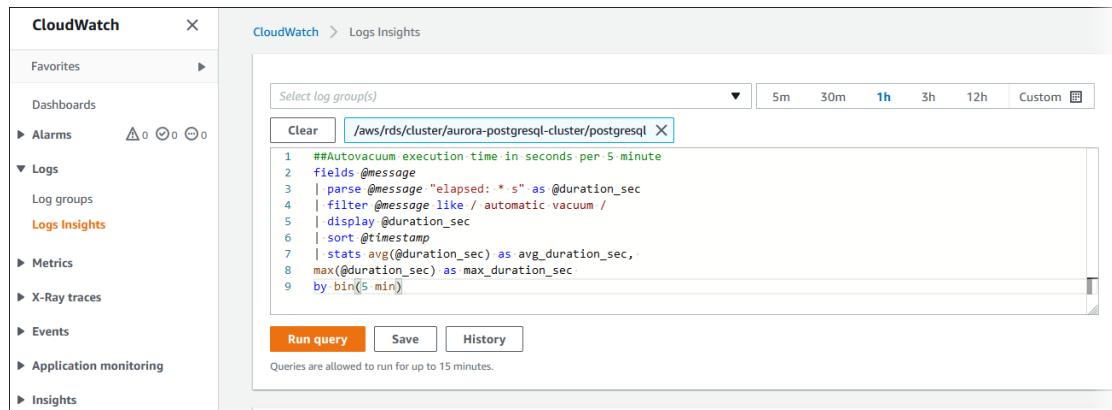
### To analyze PostgreSQL logs with CloudWatch Logs Insights

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 Aurora PostgreSQL DB cluster.

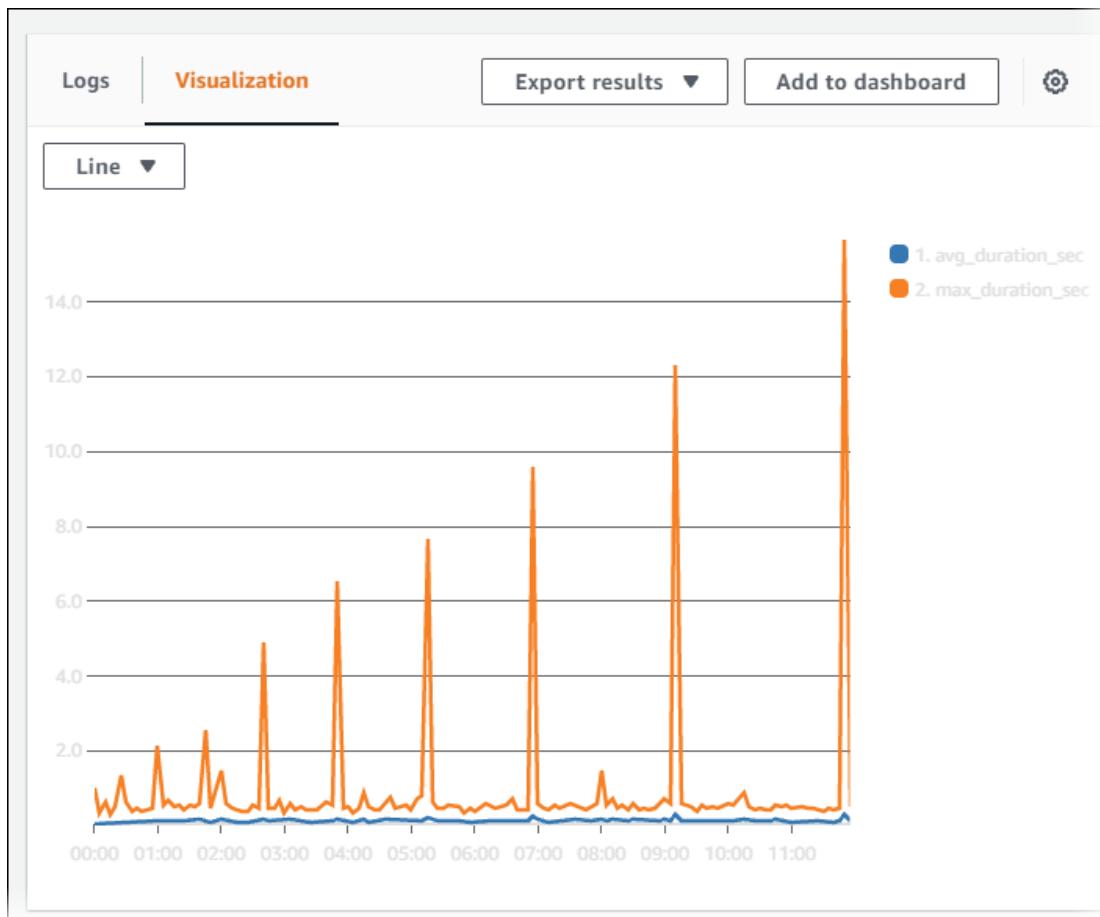


4. In the query editor, delete the query that is currently shown, enter the following, and then choose **Run query**.

```
##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** X

Select a dashboard  
Select an existing dashboard or create a new one.  
 X

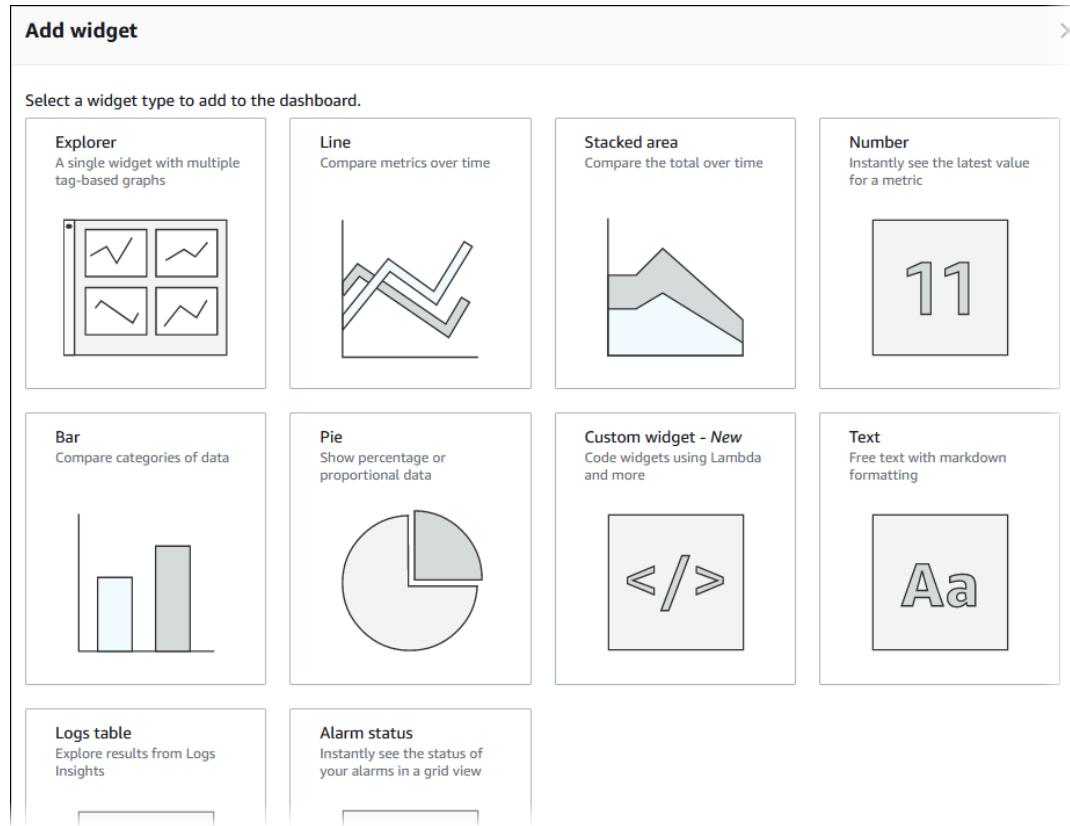
Widget type  
Select a widget type to add to the dashboard.

Customize widget title  
Widgets get an automatic title. You can optionally customize the title here.

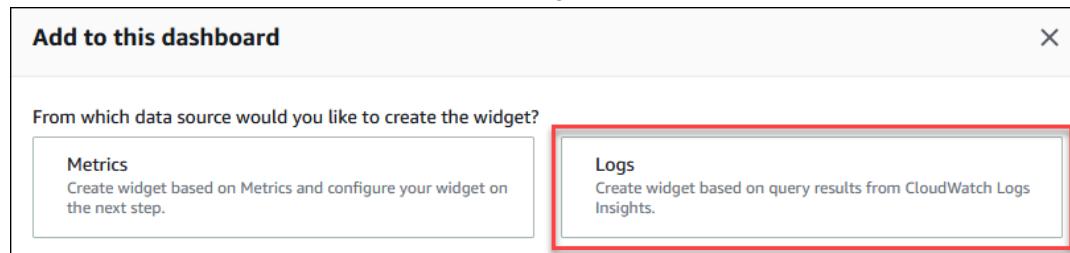
**Preview**  
This is how your chart will appear in your dashboard.  
**Autovacuum Duration - Avg and Max**

9. (Optional) Add more widgets based on your log query results.

- a. Choose **Add widget**.
- b. Choose a widget type, such as **Line**.

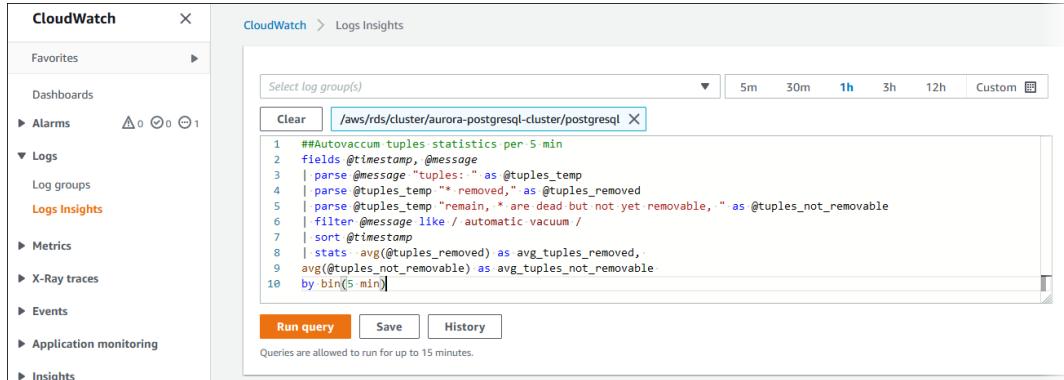


- c. In the **Add to this dashboard** window, choose **Logs**.



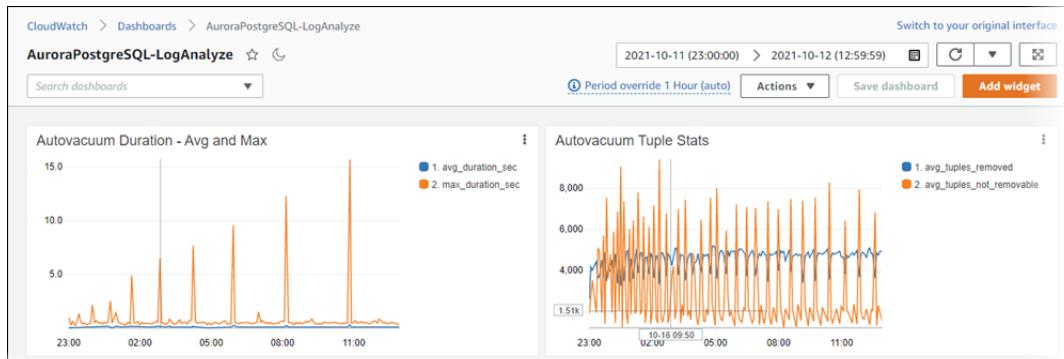
- d. In **Select log group(s)**, select the log group for your DB cluster.
- e. In the query editor, delete the query that is currently shown, enter the following, and then choose **Run query**.

```
##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 \(p. 1275\)](#).

For general information about SageMaker, see [SageMaker](#). For details about using Aurora and SageMaker together, see [Using SageMaker to run your own ML models \(p. 1277\)](#).

**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 \(p. 1273\)](#)
- [Enabling Aurora machine learning \(p. 1273\)](#)
- [Using Amazon Comprehend for natural language processing \(p. 1275\)](#)
- [Exporting data to Amazon S3 for SageMaker model training \(p. 1276\)](#)
- [Using SageMaker to run your own ML models \(p. 1277\)](#)
- [Best practices with Aurora machine learning \(p. 1280\)](#)
- [Monitoring Aurora machine learning \(p. 1283\)](#)
- [PostgreSQL function reference for Aurora machine learning \(p. 1285\)](#)
- [Manually setting up IAM roles for SageMaker and Amazon Comprehend using the AWS CLI \(p. 1286\)](#)

## Prerequisites for 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 \(p. 1415\)](#).

For more information about Regions and Aurora version availability, see [Aurora machine learning \(p. 24\)](#).

## Enabling Aurora machine learning

Enabling the ML capabilities involves the following steps.

**Topics**

- [Setting up IAM access to AWS machine learning services \(p. 1273\)](#)
- [Installing the aws\\_ml extension for model inference \(p. 1275\)](#)

## 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 \(p. 1286\)](#).

### 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 \(p. 1285\)](#).

## 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 \(p. 1275\)](#)), it provides the [aws\\_comprehend.detect\\_sentiment \(p. 1285\)](#) 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 \(p. 1280\)](#).

For information about parameters and return types for the sentiment detection function, see [aws\\_comprehend.detect\\_sentiment \(p. 1285\)](#).

## 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 \(p. 1243\)](#).

## 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 \(p. 1275\)](#)), it provides the `aws_sagemaker.invoke_endpoint` (p. 1285) 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 \(p. 1277\)](#)
- [Passing an array as input to a SageMaker model \(p. 1278\)](#)
- [Specifying batch size when invoking a SageMaker model \(p. 1279\)](#)
- [Invoking a SageMaker model that has multiple outputs \(p. 1279\)](#)

## 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 $$$
    SELECT aws_sagemaker.invoke_endpoint (
        'sagemaker_model_endpoint_name', NULL,
        arg1, arg2
        -- model inputs are separate arguments
        )::INT
        -- cast the output to INT
$$ 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 \(p. 1285\)](#) 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 \(p. 1282\)](#).
- 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 \(p. 1285\)](#) 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 $$$
    SELECT aws_sagemaker.invoke_endpoint (
        'sagemaker_model_endpoint_name', NULL,
        params
        -- model input parameters as an array
        )::REAL
        -- cast output to REAL
$$ 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 \(p. 1280\)](#).
- 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 \(p. 1280\)](#)
- [Exploiting parallel query processing \(p. 1282\)](#)
- [Using materialized views and materialized columns \(p. 1283\)](#)

## 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 \(p. 1280\)](#)
- [Using the `max\_rows\_per\_batch` parameter \(p. 1281\)](#)
- [Verifying batch-mode execution \(p. 1281\)](#)

### 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 \(p. 1290\)](#).

### Using the `max_rows_per_batch` parameter

The `max_rows_per_batch` parameter of the [aws\\_sagemaker.invoke\\_endpoint \(p. 1285\)](#) and [aws\\_comprehend.detect\\_sentiment \(p. 1285\)](#) 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 \(p. 1290\)](#).

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 \(p. 1285\)](#)
- [aws\\_sagemaker.invoke\\_endpoint \(p. 1285\)](#)

### [aws\\_comprehend.detect\\_sentiment](#)

Performs sentiment analysis using Amazon Comprehend. For more about usage, see [Using Amazon Comprehend for natural language processing \(p. 1275\)](#).

#### 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 \(p. 1280\)](#).

#### 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 \(p. 1277\)](#).

#### 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 \(p. 1280\)](#).

### **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 6 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 \(p. 1273\)](#).

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 \(p. 1287\)](#)
- [Creating an IAM policy to access Amazon Comprehend using the AWS CLI \(p. 1287\)](#)
- [Creating an IAM role to access SageMaker and Amazon Comprehend \(p. 1288\)](#)
- [Associating an IAM role with an Aurora PostgreSQL DB cluster using the AWS CLI \(p. 1288\)](#)

## 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 \(p. 1273\)](#).

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 \(p. 1288\)](#).

## 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 \(p. 1273\)](#).

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 \(p. 1288\)](#).

### 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 \(p. 1273\)](#).

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 \(p. 1689\)](#).

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 \(p. 1288\)](#).

### 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 \(p. 1273\)](#).

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.

## 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 being used, and assess the impact of creating or dropping an index.
- Automatically detect a new minimum-cost plan generated by the optimizer.
- Try new optimizer features with less risk, because you can choose to approve only the plan changes that improve performance.

### Topics

- [Turning on query plan management for Aurora PostgreSQL \(p. 1291\)](#)
- [Upgrading Aurora PostgreSQL query plan management \(p. 1292\)](#)
- [Basics of Aurora PostgreSQL query plan management \(p. 1292\)](#)
- [Best practices for Aurora PostgreSQL query plan management \(p. 1295\)](#)
- [Examining Aurora PostgreSQL query plans in the dba\\_plans view \(p. 1297\)](#)

- [Capturing Aurora PostgreSQL execution plans \(p. 1299\)](#)
- [Using Aurora PostgreSQL managed plans \(p. 1301\)](#)
- [Maintaining Aurora PostgreSQL execution plans \(p. 1304\)](#)
- [Parameter reference for Aurora PostgreSQL query plan management \(p. 1308\)](#)
- [Function reference for Aurora PostgreSQL query plan management \(p. 1311\)](#)

## Turning on query plan management for Aurora PostgreSQL

Query plan management is available with the following Aurora PostgreSQL versions:

- All Aurora PostgreSQL 14 and 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 \(p. 228\)](#). 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 \(p. 249\)](#).
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 \(p. 219\)](#). 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 \(p. 248\)](#).
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 \(p. 221\)](#).
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 Aurora PostgreSQL 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 \(p. 1415\)](#).

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 Aurora PostgreSQL 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

- [Capturing an Aurora PostgreSQL query plan manually \(p. 1292\)](#)
- [Viewing captured plans \(p. 1292\)](#)
- [Working with managed statements and the SQL hash \(p. 1293\)](#)
- [Working with automatic plan capture \(p. 1294\)](#)
- [Validating plans \(p. 1294\)](#)
- [Approving new plans that improve performance \(p. 1294\)](#)
- [Deleting plans \(p. 1295\)](#)

## Capturing an Aurora PostgreSQL query plan manually

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 \(p. 1300\)](#). 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 Aurora PostgreSQL query plans in the dba\\_plans view \(p. 1297\)](#).

## 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 \(p. 231\)](#).

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 Aurora PostgreSQL execution plans \(p. 1299\)](#).

## 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 \(p. 1305\)](#).

## 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 \(p. 1304\)](#).

## 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 \(p. 1307\)](#).

# Best practices for Aurora PostgreSQL query plan management

Query plan management lets you control how and when query execution plans change. As a DBA, your main goals when using QPM include preventing regressions when there are changes to your database, and controlling when the optimizer can use a new plan. In the following, you can find some recommended best practices for using query plan management. Proactive and reactive plan management approaches differ in how and when new plans get approved for use.

## Proactive plan management to help prevent performance regression

To prevent plan performance regressions from occurring, you gather evidence and then manually approve new plans after you have verified that they are faster.

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 \(p. 1300\)](#) and [Automatically capturing plans \(p. 1300\)](#).

2. Export the captured plans from the development environment and import them into the production environment. For more information, see [Exporting and importing plans \(p. 1307\)](#).
3. In production, run your application and enforce the use of approved managed plans. For more information, see [Using Aurora PostgreSQL managed plans \(p. 1301\)](#). While the application runs, also add new plans as the optimizer discovers them. For more information, see [Automatically capturing plans \(p. 1300\)](#).
4. Analyze the unapproved plans and approve those that perform well. For more information, see [Evaluating plan performance \(p. 1304\)](#).
5. While your application continues to run, the optimizer begins to use the new plans as appropriate.

## Ensuring plan stability after a major version upgrade

Each major version of PostgreSQL includes enhancements and changes to the query optimizer that are designed to improve performance. However, your workload may include queries that result in a worse performing plan in the new version. You can use the query plan manager to ensure plan stability after a major version upgrade.

The optimizer always uses the minimum cost plan, even if more than one approved plans exist. That means that you can have multiple approved plans for each statement in your workload prior to upgrading. After the upgrade, the optimizer will use only one of the approved plans, even if the changes in the new major version might lead to a different plan being created. After upgrading, you can use the `evolve_plan_baselines` function to compare plan performance before and after the upgrade using your query parameter bindings. The following steps assume that you have been using approved managed plans in your production environment, as detailed in [Using Aurora PostgreSQL managed plans \(p. 1301\)](#).

1. Before upgrading, run your application with the query plan manager running. While the application runs, add new plans as the optimizer discovers them. For more information, see [Automatically capturing plans \(p. 1300\)](#).
2. Evaluate each plan's performance. For more information, see [Evaluating plan performance \(p. 1304\)](#).
3. After upgrading, analyze your approved plans again using the `evolve_plan_baselines` function. Compare performance before and after using your query parameter bindings. If the new plan is fast, you can add it to your approved plans. If it's faster than another plan for the same parameter bindings, then you can mark the slower plan as Rejected.

For more information, see [Approving better plans \(p. 1304\)](#). For reference information about this function, see [apg\\_plan\\_mgmt.evolve\\_plan\\_baselines \(p. 1312\)](#).

For more information, see [Ensuring consistent performance after major version upgrades with Amazon Aurora PostgreSQL-Compatible Edition Query Plan Management](#).

## Reactive plan management to detect and repair performance regressions

By monitoring your application as it runs, you can detect plans that cause performance regressions. When you detect regressions, you manually reject or fix the bad plans by following these steps:

1. While your application runs, enforce the use of managed plans and automatically add newly discovered plans as unapproved. For more information, see [Using Aurora PostgreSQL managed plans \(p. 1301\)](#) and [Automatically capturing plans \(p. 1300\)](#).
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 \(p. 1305\)](#).

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 \(p. 1306\)](#).

## Examining Aurora PostgreSQL query plans in the dba\_plans view

Query plan management provides 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 about 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 <a href="#">apg_plan_mgmt.evolve_plan_baselines (p. 1312)</a> 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.

dba_plans column	Description
	from being used by the optimizer. To modify this value, use the <a href="#">apg_plan_mgmt.set_plan_enabled (p. 1315)</a> function.
environment_variables	The PostgreSQL Grand Unified Configuration (GUC) parameters and values that the optimizer has overridden at the time the plan was captured.
estimated_startup_cost	The estimated optimizer setup cost before the optimizer delivers rows of a table.
estimated_total_cost	The estimated optimizer cost to deliver the final table row.
execution_time_benefit	The execution time benefit in milliseconds of enabling the plan. This column is populated by the <a href="#">apg_plan_mgmt.evolve_plan_baselines (p. 1312)</a> function.
execution_time_ms	The estimated time in milliseconds that the plan would run. This column is populated by the <a href="#">apg_plan_mgmt.evolve_plan_baselines (p. 1312)</a> function.
has_side_effects	A value that indicates that the SQL statement is a data manipulation language (DML) statement or a SELECT statement that contains a VOLATILE function.
last_used	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 <a href="#">apg_plan_mgmt.plan_retention_period (p. 1310)</a> parameter.
last_validated	The most recent date and time when it was verified that the plan could be recreated by either the <a href="#">apg_plan_mgmt.validate_plans (p. 1317)</a> function or the <a href="#">apg_plan_mgmt.evolve_plan_baselines (p. 1312)</a> function.
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 <a href="#">apg_plan_mgmt.evolve_plan_baselines (p. 1312)</a> function.
origin	How the plan was captured with the <a href="#">apg_plan_mgmt.capture_plan_baselines (p. 1309)</a> 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 that 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 <a href="#">apg_plan_mgmt.evolve_plan_baselines (p. 1312)</a> function.

dba_plans column	Description
queryId	A statement hash, as calculated by the pg_stat_statements 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	<p>A plan's status, which determines how the optimizer uses a plan. Valid values include the following.</p> <ul style="list-style-type: none"> <li>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 <a href="#">apg_plan_mgmt.evolve_plan_baselines (p. 1312)</a> function.</li> <li>Unapproved – A captured plan that you have not verified for use. For more information, see <a href="#">Evaluating plan performance (p. 1304)</a>.</li> <li>Rejected – A plan that the optimizer won't use. For more information, see <a href="#">Rejecting or disabling slower plans (p. 1305)</a>.</li> <li>Preferred – A plan that you have determined is a preferred plan to use for a managed statement.</li> </ul> <p>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.</p> <p>To reset a plan to Preferred, use the <a href="#">apg_plan_mgmt.set_plan_status (p. 1315)</a> function.</p>
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	<p>The total time benefit in milliseconds of enabling this plan. This value considers both planning time and execution time.</p> <p>If this value is negative, there is a disadvantage to enabling this plan. This column is populated by the <a href="#">apg_plan_mgmt.evolve_plan_baselines (p. 1312)</a> function.</p>

## Capturing Aurora PostgreSQL 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 \(p. 1310\)](#) parameter.

### Topics

- [Manually capturing plans for specific SQL statements \(p. 1300\)](#)
- [Automatically capturing plans \(p. 1300\)](#)

## 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_baselines = manual;
psql> \i my-statements.sql
psql> SET apg_plan_mgmt.capture_plan_baselines = 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 \(p. 1306\)](#). To compare the performance of the unapproved and approved plans and approve, reject, or delete them, see [Evaluating plan performance \(p. 1304\)](#).

## 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 \(p. 231\)](#).
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 \(p. 1304\)](#).

## Using Aurora PostgreSQL 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.

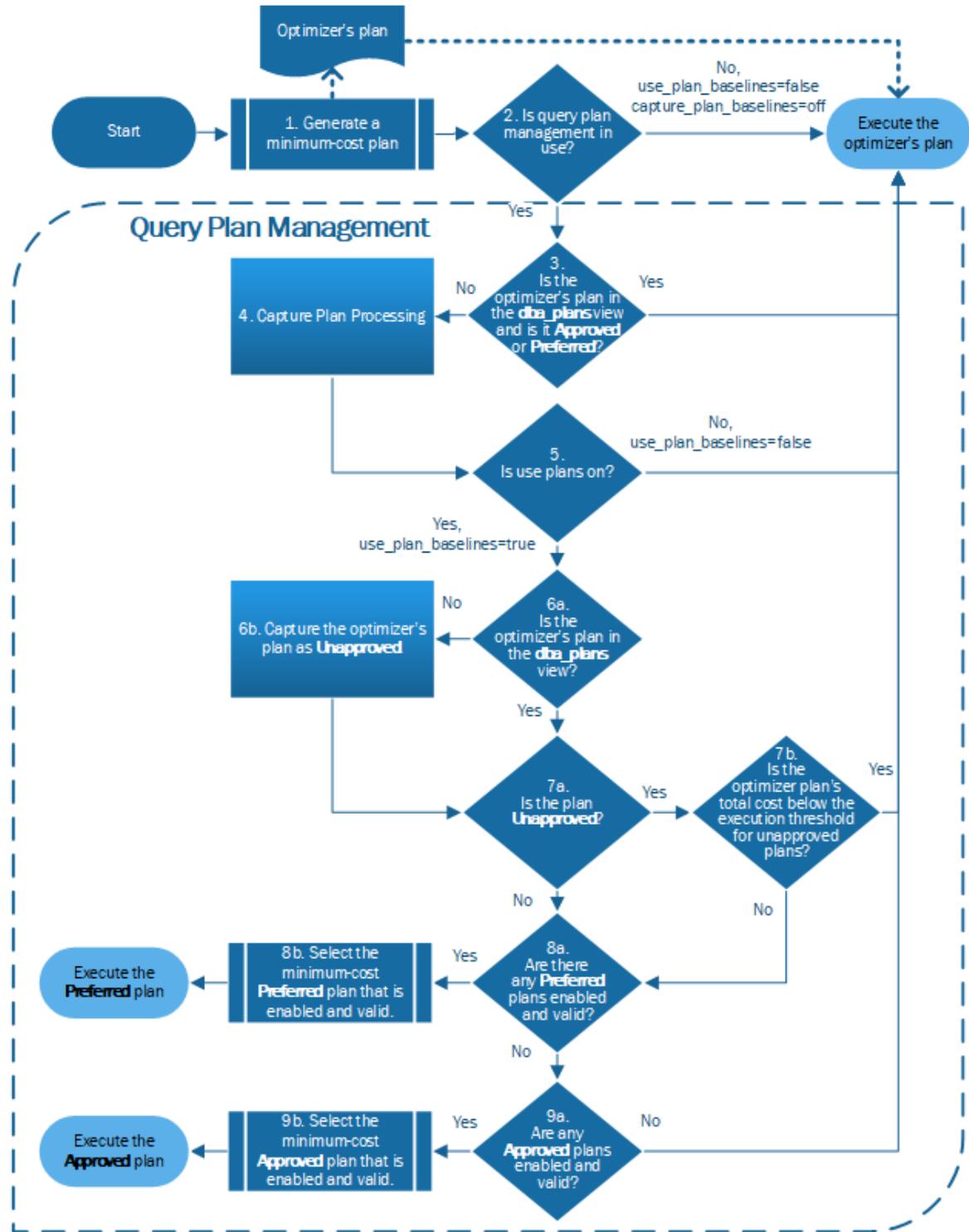
```
SET apg_plan_mgmt.use_plan_baselines = true;
```

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 Aurora PostgreSQL execution plans \(p. 1299\)](#).

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 \(p. 1310\)](#) 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 \(p. 1300\)](#).

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 \(p. 1304\)](#).

## Maintaining Aurora PostgreSQL execution plans

Query plan management provides techniques and functions to add, maintain, and improve execution plans.

### Topics

- [Evaluating plan performance \(p. 1304\)](#)
- [Validating plans \(p. 1305\)](#)
- [Fixing plans using pg\\_hint\\_plan \(p. 1306\)](#)
- [Deleting plans \(p. 1307\)](#)
- [Exporting and importing plans \(p. 1307\)](#)

## 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 \(p. 1304\)](#)
- [Rejecting or disabling slower plans \(p. 1305\)](#)

## 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 disable 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 \(p. 1305\)](#).

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's 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 \(p. 1310\)](#) 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 Aurora PostgreSQL query plan management

You can set your preferences for the `apg_plan_mgmt` extension by using the parameters listed in this section. These are available in the custom DB cluster parameter and the DB parameter group associated with your Aurora PostgreSQL DB cluster. These parameters control the behavior of the query plan management feature and how it affects the optimizer. For information about setting up query plan management, see [Turning on query plan management for Aurora PostgreSQL \(p. 1291\)](#). Changing the parameters following has no effect if the `apg_plan_mgmt` extension isn't set up as detailed in

that section. For information about modifying parameters, see [Modifying parameters in a DB cluster parameter group \(p. 221\)](#) and [Working with DB parameter groups \(p. 228\)](#).

#### Parameters

- [apg\\_plan\\_mgmt.capture\\_plan\\_baselines \(p. 1309\)](#)
- [apg\\_plan\\_mgmt.max\\_databases \(p. 1309\)](#)
- [apg\\_plan\\_mgmt.max\\_plans \(p. 1310\)](#)
- [apg\\_plan\\_mgmt.plan\\_retention\\_period \(p. 1310\)](#)
- [apg\\_plan\\_mgmt.unapproved\\_plan\\_execution\\_threshold \(p. 1310\)](#)
- [apg\\_plan\\_mgmt.use\\_plan\\_baselines \(p. 1311\)](#)

## apg\_plan\_mgmt.capture\_plan\_baselines

Captures query execution plans generated by the optimizer for each SQL statement and stores them in the `dba_plans` view. By default, the maximum number of plans that can be stored is 10,000 as specified by the `apg_plan_mgmt.max_plans` parameter. For reference information, see [apg\\_plan\\_mgmt.max\\_plans \(p. 1310\)](#).

You can set this parameter in the custom DB cluster parameter group or in the custom DB parameter group. Changing the value of this parameter doesn't require a reboot.

Default	Allowed values	Description
off	automatic	Turns on plan capture for all databases on the DB instance. Collects a plan for each SQL statement that runs two or more times. Use this setting for large or evolving workloads to provide plan stability.
	manual	Turns on plan capture for subsequent statements only, until you turn it off again. Using this setting lets you capture query execution plans for specific critical SQL statements only or for known problematic queries.
	off	Turns off plan capture.

For more information, see [Capturing Aurora PostgreSQL execution plans \(p. 1299\)](#).

## apg\_plan\_mgmt.max\_databases

Specifies the maximum number of databases on your Aurora PostgreSQL DB cluster's Writer instance that can use query plan management. By default, up to 10 databases can use query plan management. If you have more than 10 databases on the instance, you can change the value of this setting. To find out how many databases are on a given instance, connect to the instance using `psql`. Then, use the `psql` metacommand, `\l`, to list the databases.

Changing the value of this parameter requires that you reboot the instance for the setting to take effect.

Default	Allowed values	Description
10	10-2147483647	Maximum number of databases that can use query plan management on the instance.

You can set this parameter in the custom DB cluster parameter group or in the custom DB parameter group.

## 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.

You can set this parameter in the custom DB cluster parameter group or in the custom DB parameter group. Changing the value of this parameter requires that you reboot the instance for the setting to take effect.

Default	Allowed values	Description
10000	10-2147483647	Maximum number of plans that can be stored in the <code>apg_plan_mgmt.dba_plans</code> view.  Default for Aurora PostgreSQL version 10 and older versions is 1000.

For more information, see [Examining Aurora PostgreSQL query plans in the dba\\_plans view \(p. 1297\)](#).

## apg\_plan\_mgmt.plan\_retention\_period

Specifies the number of days to keep plans in the `apg_plan_mgmt.dba_plans` view, after which they're automatically deleted. By default, a plan is deleted when 32 days have elapsed since the plan was last used (the `last_used` column in the `apg_plan_mgmt.dba_plans` view). You can change this setting to any number, 32 and over.

Changing the value of this parameter requires that you reboot the instance for the setting to take effect.

Default	Allowed values	Description
32	32-2147483647	Maximum number of days since a plan was last used before it's deleted.

For more information, see [Examining Aurora PostgreSQL query plans in the dba\\_plans view \(p. 1297\)](#).

## apg\_plan\_mgmt.unapproved\_plan\_execution\_threshold

Specifies a threshold below which an Unapproved plan can be used by the optimizer. By default, the optimizer doesn't run Unapproved plans. If you have an Unapproved plan that you suspect might be better than an Approved plan, you can use this parameter to sidestep the optimizer's default behavior. The value of this parameter represents a cost estimate for running a given plan. If an Unapproved plan is below that estimated cost, the optimizer uses it for the SQL statement. You can see captured plans and their status (Approved, Unapproved) in the `dba_plans` view. To learn more, see [Examining Aurora PostgreSQL query plans in the dba\\_plans view \(p. 1297\)](#).

Changing the value of this parameter doesn't require a reboot.

Default	Allowed values	Description
0	0-2147483647	Estimated plan cost below which an Unapproved plan is used.

For more information, see [Using Aurora PostgreSQL managed plans \(p. 1301\)](#).

## apg\_plan\_mgmt.use\_plan\_baselines

Specifies that the optimizer should use one of the Approved plans captured and stored in the `apg_plan_mgmt.dba_plans` view. By default, this parameter is off (false), causing the optimizer to use the minimum-cost plan that it generates without any further assessment. Turning this parameter on (setting it to true) forces the optimizer to choose a query execution plan for the statement from its plan baseline. For more information, see [Using Aurora PostgreSQL managed plans \(p. 1301\)](#). To find an image detailing this process, see [How the optimizer chooses which plan to run \(p. 1301\)](#).

You can set this parameter in the custom DB cluster parameter group or in the custom DB parameter group. Changing the value of this parameter doesn't require a reboot.

Default	Allowed values	Description
false	true	Use an Approved, Preferred, or Unapproved plan from the <code>apg_plan_mgmt.dba_plans</code> . If none of those meet all evaluation criterion for the optimizer, it can then use its own generated minimum-cost plan. For more information, see <a href="#">How the optimizer chooses which plan to run (p. 1301)</a> .
	false	Use the minimum cost plan generated by the optimizer.

You can evaluate response times of different captured plans and change plan status, as needed. For more information, see [Maintaining Aurora PostgreSQL execution plans \(p. 1304\)](#).

## Function reference for Aurora PostgreSQL query plan management

The `apg_plan_mgmt` extension provides the following functions.

### Functions

- [apg\\_plan\\_mgmt.delete\\_plan \(p. 1311\)](#)
- [apg\\_plan\\_mgmt.evolve\\_plan\\_baselines \(p. 1312\)](#)
- [apg\\_plan\\_mgmt.get\\_explain\\_plan \(p. 1313\)](#)
- [apg\\_plan\\_mgmt.plan\\_last\\_used \(p. 1314\)](#)
- [apg\\_plan\\_mgmt.reload \(p. 1314\)](#)
- [apg\\_plan\\_mgmt.set\\_plan\\_enabled \(p. 1315\)](#)
- [apg\\_plan\\_mgmt.set\\_plan\\_status \(p. 1315\)](#)
- [apg\\_plan\\_mgmt.update\\_plans\\_last\\_used \(p. 1316\)](#)
- [apg\\_plan\\_mgmt.validate\\_plans \(p. 1317\)](#)

## 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\_baseline

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_baseline(
    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.
action	The action the function is to perform. Valid values include the following. Case does not matter. <ul style="list-style-type: none"><li>• 'disable' – Disable each matching plan that does not meet the minimum speedup factor.</li><li>• 'approve' – Enable each matching plan that meets the minimum speedup factor and set its status to <code>approved</code>.</li><li>• 'reject' – For each matching plan that does not meet the minimum speedup factor, set its status to <code>rejected</code>.</li></ul>

Parameter	Description
	<ul style="list-style-type: none"> <li>• NULL – The function simply returns the number of plans that have no performance benefit because they do not meet the minimum speedup factor.</li> </ul>

### 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.

Parameter	Description
plan_hash	The managed plan's plan_hash ID.
explainOptionList	A comma-separated list of explain options. Valid values include 'analyze', 'verbose', 'buffers', 'hashes', and 'format json'. If the explainOptionList is NULL or an empty string (""), this function generates an EXPLAIN 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 that you provide to the EXPLAIN statement.

## 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
sql_hash	The sql_hash ID of the plan's managed SQL statement.
plan_hash	The managed plan's plan_hash 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"><li>• A value of <code>true</code> enables the plan.</li><li>• A value of <code>false</code> disables the plan.</li></ul>

## 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 sql_hash ID of the plan's managed SQL statement.
plan_hash	The managed plan's plan_hash ID.
status	A string with one of the following values: <ul style="list-style-type: none"><li>• 'Approved'</li><li>• 'Unapproved'</li><li>• 'Rejected'</li><li>• 'Preferred'</li></ul> The case you use does not matter, however the status value is set to initial uppercase in the apg_plan_mgmt.dba_plans view. For more information about these values, see status in <a href="#">Reference for the apg_plan_mgmt.dba_plans view (p. 1297)</a> .

## 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"><li>• <code>'disable'</code> – Each invalid plan is disabled.</li><li>• <code>'delete'</code> – Each invalid plan is deleted.</li><li>• <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.</li><li>• <code>NULL</code> – The function simply returns the number of invalid plans. No other action is performed.</li><li>• <code>" "</code> – An empty string produces a message indicating the number of both valid and invalid plans.</li></ul> <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`.

## Working with extensions and foreign data wrappers

To add some types of functionality to your Aurora PostgreSQL-Compatible Edition DB cluster, you can install and use various PostgreSQL extensions. For example, if your use case calls for intensive data entry across very large tables, you can install the `pg_partman` extension to partition your data and thus spread the workload. An extension that provides access to external data is generally known as a *foreign data wrapper* (FDW). As one example, the `oracle_fdw` extension allows your Aurora PostgreSQL DB cluster to work with Oracle databases.

Following, you can find information about setting up and using some of the extensions and FDWs available for Aurora PostgreSQL. For listings of the extensions and FDWs that you can use with currently available Aurora PostgreSQL versions, see [Extension versions for Aurora PostgreSQL-Compatible Edition](#) in the *Release Notes for Aurora PostgreSQL*.

- Managing large objects with the `lo` module ([p. 1318](#))
- Managing spatial data with the PostGIS extension ([p. 1320](#))
- Scheduling maintenance with the PostgreSQL `pg_cron` extension ([p. 1327](#))
- Managing PostgreSQL partitions with the `pg_partman` extension ([p. 1333](#))
- Working with Oracle databases by using the `oracle_fdw` extension ([p. 1337](#))
- Working with SQL Server databases by using the `tds_fdw` extension ([p. 1340](#))

## Managing large objects with the `lo` module

The `lo` module (extension) is for database users and developers working with PostgreSQL databases through JDBC or ODBC drivers. Both JDBC and ODBC expect the database to handle deletion of large objects when references to them change. However, PostgreSQL doesn't work that way. PostgreSQL doesn't assume that an object should be deleted when its reference changes. The result is that objects remain on disk, unreferenced. The `lo` extension includes a function that you use to trigger on reference changes to delete objects if needed.

### Tip

To determine if your database can benefit from the `lo` extension, use the `vacuumlo` utility to check for orphaned large objects. To get counts of orphaned large objects without taking any action, run the utility with the `-n` option (no-op). To learn how, see [vacuumlo utility](#) following.

The `lo` module is available for Aurora PostgreSQL 13.7, 12.11, 11.16, 10.21 and higher minor versions.

To install the module (extension), you need `rds_superuser` privileges. Installing the `lo` extension adds the following to your database:

- `lo` – This is a large object (`lo`) data type that you can use for binary large objects (BLOBs) and other large objects. The `lo` data type is a domain of the `oid` data type. In other words, it's an object

identifier with optional constraints. For more information, see [Object identifiers](#) in the PostgreSQL documentation. In simple terms, you can use the `lo` data type to distinguish your database columns that hold large object references from other object identifiers (OIDs).

- `lo_manage` – This is a function that you can use in triggers on table columns that contain large object references. Whenever you delete or modify a value that references a large object, the trigger unlinks the object (`lo_unlink`) from its reference. Use the trigger on a column only if the column is the sole database reference to the large object.

For more information about the large objects module, see [lo](#) in the PostgreSQL documentation.

## Installing the lo extension

Before installing the `lo` extension, make sure that you have `rds_superuser` privileges.

### To install the `lo` extension

1. Use `psql` to connect to the primary DB instance of your Aurora PostgreSQL DB cluster.

```
psql --host=your-cluster-instance-1.666666666666.aws-region.rds.amazonaws.com --  
port=5432 --username=postgres --password
```

When prompted, enter your password. The `psql` client connects and displays the default administrative connection database, `postgres=>`, as the prompt.

2. Install the extension as follows.

```
postgres=> CREATE EXTENSION lo;  
CREATE EXTENSION
```

You can now use the `lo` data type to define columns in your tables. For example, you can create a table (`images`) that contains raster image data. You can use the `lo` data type for a column `raster`, as shown in the following example, which creates a table.

```
postgres=> CREATE TABLE images (image_name text, raster lo);
```

## Using the `lo_manage` trigger function to delete objects

You can use the `lo_manage` function in a trigger on a `lo` or other large object columns to clean up (and prevent orphaned objects) when the `lo` is updated or deleted.

### To set up triggers on columns that reference large objects

- Do one of the following:
  - Create a BEFORE UPDATE OR DELETE trigger on each column to contain unique references to large objects, using the column name for the argument.

```
postgres=> CREATE TRIGGER t_raster BEFORE UPDATE OR DELETE ON images  
FOR EACH ROW EXECUTE FUNCTION lo_manage(raster);
```

- Apply a trigger only when the column is being updated.

```
postgres=> CREATE TRIGGER t_raster BEFORE UPDATE OF images  
FOR EACH ROW EXECUTE FUNCTION lo_manage(raster);
```

The `lo_manage` trigger function works only in the context of inserting or deleting column data, depending on how you define the trigger. It has no effect when you perform a `DROP` or `TRUNCATE` operation on a database. That means that you should delete object columns from any tables before dropping, to prevent creating orphaned objects.

For example, suppose that you want to drop the database containing the `images` table. You delete the column as follows.

```
postgres=> DELETE FROM images COLUMN raster
```

Assuming that the `lo_manage` function is defined on that column to handle deletes, you can now safely drop the table.

## Using the `vacuumlo` utility

The `vacuumlo` utility identifies and can remove orphaned large objects from databases. This utility has been available since PostgreSQL 9.1.24. If your database users routinely work with large objects, we recommend that you run `vacuumlo` occasionally to clean up orphaned large objects.

Before installing the `lo` extension, you can use `vacuumlo` to assess whether your Aurora PostgreSQL DB cluster can benefit. To do so, run `vacuumlo` with the `-n` option (no-op) to show what would be removed, as shown in the following:

```
$ vacuumlo -v -n -h your-cluster-instance-1.666666666666.aws-region.rds.amazonaws.com -p
5433 -U postgres docs-lab-spatial-db
Password:*****
Connected to database "docs-lab-spatial-db"
Test run: no large objects will be removed!
Would remove 0 large objects from database "docs-lab-spatial-db".
```

As the output shows, orphaned large objects aren't a problem for this particular database.

For more information about this utility, see [vacuumlo](#) in the PostgreSQL documentation.

## Managing spatial data with the PostGIS extension

PostGIS is an extension to PostgreSQL for storing and managing spatial information. To learn more about PostGIS, see [PostGIS.net](#).

Starting with version 10.5, PostgreSQL supports the libprotobuf 1.3.0 library used by PostGIS for working with map box vector tile data.

Setting up the PostGIS extension requires `rds_superuser` privileges. We recommend that you create a user (role) to manage the PostGIS extension and your spatial data. The PostGIS extension and its related components add thousands of functions to PostgreSQL. Consider creating the PostGIS extension in its own schema if that makes sense for your use case. The following example shows how to install the extension in its own database, but this isn't required.

### Topics

- [Step 1: Create a user \(role\) to manage the PostGIS extension \(p. 1321\)](#)
- [Step 2: Load the PostGIS extensions \(p. 1321\)](#)
- [Step 3: Transfer ownership of the extensions \(p. 1322\)](#)
- [Step 4: Transfer ownership of the PostGIS objects \(p. 1322\)](#)
- [Step 5: Test the extensions \(p. 1323\)](#)
- [Step 6: Upgrade the PostGIS extension \(p. 1323\)](#)

- [PostGIS extension versions \(p. 1324\)](#)
- [Upgrading PostGIS 2 to PostGIS 3 \(p. 1324\)](#)

## Step 1: Create a user (role) to manage the PostGIS extension

First, connect to your RDS for PostgreSQL DB instance as a user that has `rds_superuser` privileges. If you kept the default name when you set up your instance, you connect as `postgres`.

```
psql --host=111122223333.aws-region.rds.amazonaws.com --port=5432 --username=postgres --password
```

Create a separate role (user) to administer the PostGIS extension.

```
postgres=> CREATE ROLE gis_admin LOGIN PASSWORD 'change_me';
CREATE ROLE
```

Grant this role `rds_superuser` privileges, to allow the role to install the extension.

```
postgres=> GRANT rds_superuser TO gis_admin;
GRANT
```

Create a database to use for your PostGIS artifacts. This step is optional. Or you can create a schema in your user database for the PostGIS extensions, but this also isn't required.

```
postgres=> CREATE DATABASE lab_gis;
CREATE DATABASE
```

Give the `gis_admin` all privileges on the `lab_gis` database.

```
postgres=> GRANT ALL PRIVILEGES ON DATABASE lab_gis TO gis_admin;
GRANT
```

Exit the session and reconnect to your RDS for PostgreSQL DB instance as `gis_admin`.

```
postgres=> --host=111122223333.aws-region.rds.amazonaws.com --port=5432 --
username=gis_admin --password --dbname=lab_gis
Password for user gis_admin:...
lab_gis=>
```

Continue setting up the extension as detailed in the next steps.

## Step 2: Load the PostGIS extensions

The PostGIS extension includes several related extensions that work together to provide geospatial functionality. Depending on your use case, you might not need all the extensions created in this step.

Use `CREATE EXTENSION` statements to load the PostGIS extensions.

```
CREATE EXTENSION postgis;
CREATE EXTENSION
CREATE EXTENSION postgis_raster;
CREATE EXTENSION
CREATE EXTENSION postgis_tiger_geocoder;
```

```
CREATE EXTENSION
CREATE EXTENSION postgis_topology;
CREATE EXTENSION
CREATE EXTENSION fuzzystrmatch;
CREATE EXTENSION
CREATE EXTENSION address_standardizer_data_us;
CREATE EXTENSION
```

You can verify the results by running the SQL query shown in the following example, which lists the extensions and their owners.

```
SELECT n.nspname AS "Name",
       pg_catalog.pg_get_userbyid(n.nspowner) AS "Owner"
  FROM pg_catalog.pg_namespace n
 WHERE n.nspname !~ '^pg_' AND n.nspname <> 'information_schema'
 ORDER BY 1;
List of schemas
   Name      |    Owner
-----+-----
 public    | postgres
 tiger     | rdsadmin
 tiger_data | rdsadmin
 topology   | rdsadmin
(4 rows)
```

## Step 3: Transfer ownership of the extensions

Use the ALTER SCHEMA statements to transfer ownership of the schemas to the gis\_admin role.

```
ALTER SCHEMA tiger OWNER TO gis_admin;
ALTER SCHEMA
ALTER SCHEMA tiger_data OWNER TO gis_admin;
ALTER SCHEMA
ALTER SCHEMA topology OWNER TO gis_admin;
ALTER SCHEMA
```

You can confirm the ownership change by running the following SQL query. Or you can use the \dn metacommand from the psql command line.

```
SELECT n.nspname AS "Name",
       pg_catalog.pg_get_userbyid(n.nspowner) AS "Owner"
  FROM pg_catalog.pg_namespace n
 WHERE n.nspname !~ '^pg_' AND n.nspname <> 'information_schema'
 ORDER BY 1;
List of schemas
   Name      |    Owner
-----+-----
 public    | postgres
 tiger     | gis_admin
 tiger_data | gis_admin
 topology   | gis_admin
(4 rows)
```

## Step 4: Transfer ownership of the PostGIS objects

Use the following function to transfer ownership of the PostGIS objects to the gis\_admin role. Run the following statement from the psql prompt to create the function.

```
CREATE FUNCTION exec(text) returns text language plpgsql volatile AS $f$ BEGIN EXECUTE $1;
  RETURN $1; END; $f$;
CREATE FUNCTION
```

Next, run the following query to run the `exec` function that in turn runs the statements and alters the permissions.

```
SELECT exec('ALTER TABLE ' || quote_ident(s.nspname) || '.' || quote_ident(s.relname) || '
OWNER TO gis_admin;');
FROM (
  SELECT nspname, relname
  FROM pg_class c JOIN pg_namespace n ON (c.relnamespace = n.oid)
  WHERE nspname in ('tiger','topology') AND
    relkind IN ('r','S','v') ORDER BY relkind = 'S')
s;
```

## Step 5: Test the extensions

To avoid needing to specify the schema name, add the `tiger` schema to your search path using the following command.

```
SET search_path=public,tiger;
SET
```

Test the `tiger` schema by using the following `SELECT` statement.

```
SELECT address, streetname, streettypeabbrev, zip
  FROM normalize_address('1 Devonshire Place, Boston, MA 02109') AS na;
address | streetname | streettypeabbrev | zip
-----+-----+-----+-----
      1 | Devonshire | Pl             | 02109
(1 row)
```

To learn more about this extension, see [Tiger Geocoder](#) in the PostGIS documentation.

Test access to the `topology` schema by using the following `SELECT` statement. This calls the `createtopology` function to register a new topology object (`my_new_topo`) with the specified spatial reference identifier (26986) and default tolerance (0.5). To learn more, see [CreateTopology](#) in the PostGIS documentation.

```
SELECT topology.createtopology('my_new_topo',26986,0.5);
createtopology
-----
      1
(1 row)
```

## Step 6: Upgrade the PostGIS extension

Each new release of PostgreSQL supports one or more versions of the PostGIS extension compatible with that release. Upgrading the PostgreSQL engine to a new version doesn't automatically upgrade the PostGIS extension. Before upgrading the PostgreSQL engine, you typically upgrade PostGIS to the newest available version for the current PostgreSQL version. For details, see [PostGIS extension versions \(p. 1324\)](#).

After the PostgreSQL engine upgrade, you then upgrade the PostGIS extension again, to the version supported for the newly upgraded PostgreSQL engine version. For more information about

upgrading the PostgreSQL engine, see [Before upgrading your production DB cluster to a new major version \(p. 1419\)](#).

You can check for available PostGIS extension version updates on your Aurora PostgreSQL DB cluster at any time. To do so, run the following command. This function is available with PostGIS 2.5.0 and higher versions.

```
SELECT postGIS_extensions_upgrade();
```

If your application doesn't support the latest PostGIS version, you can install an older version of PostGIS that's available in your major version as follows.

```
CREATE EXTENSION postgis VERSION "2.5.5";
```

If you want to upgrade to a specific PostGIS version from an older version, you can also use the following command.

```
ALTER EXTENSION postgis UPDATE TO "2.5.5";
```

Depending on the version that you're upgrading from, you might need to use this function again. The result of the first run of the function determines if an additional upgrade function is needed. For example, this is the case for upgrading from PostGIS 2 to PostGIS 3. For more information, see [Upgrading PostGIS 2 to PostGIS 3 \(p. 1324\)](#).

If you upgraded this extension to prepare for a major version upgrade of the PostgreSQL engine, you can continue with other preliminary tasks. For more information, see [Before upgrading your production DB cluster to a new major version \(p. 1419\)](#).

## PostGIS extension versions

We recommend that you install the versions of all extensions such as PostGIS as listed in [Extension versions for Aurora PostgreSQL-Compatible Edition](#) in the *Release Notes for Aurora PostgreSQL*. To get a list of versions that are available in your release, use the following command.

```
SELECT * FROM pg_available_extension_versions WHERE name='postgis';
```

You can find version information in the following sections in the *Release Notes for Aurora PostgreSQL*:

- [Extension versions for Aurora PostgreSQL 14](#)
- [Extension versions for Aurora PostgreSQL-Compatible Edition 13](#)
- [Extension versions for Aurora PostgreSQL-Compatible Edition 12](#)
- [Extension versions for Aurora PostgreSQL-Compatible Edition 11](#)
- [Extension versions for Aurora PostgreSQL-Compatible Edition 10](#)
- [Extension versions for Aurora PostgreSQL-Compatible Edition 9.6](#)

## Upgrading PostGIS 2 to PostGIS 3

Starting with version 3.0, the PostGIS raster functionality is now a separate extension, `postgis_raster`. This extension has its own installation and upgrade path. This removes dozens of functions, data types, and other artifacts required for raster image processing from the core `postgis`

extension. That means that if your use case doesn't require raster processing, you don't need to install the `postgis_raster` extension.

In the following upgrade example, the first upgrade command extracts raster functionality into the `postgis_raster` extension. A second upgrade command is then required to upgrade `postres_raster` to the new version.

### To upgrade from PostGIS 2 to PostGIS 3

1. Identify the default version of PostGIS that's available to the PostgreSQL version on your Aurora PostgreSQL DB cluster. To do so, run the following query.

```
SELECT * FROM pg_available_extensions
  WHERE default_version > installed_version;
    name | default_version | installed_version | comment
-----+-----+-----+
+-----+
 postgis | 3.1.4           | 2.3.7          | PostGIS geometry and geography spatial
 types and functions
(1 row)
```

2. Identify the versions of PostGIS installed in each database on the writer instance of your Aurora PostgreSQL DB cluster. In other words, query each user database as follows.

```
SELECT
  e.extname AS "Name",
  e.extversion AS "Version",
  n.nspname AS "Schema",
  c.description AS "Description"
FROM
  pg_catalog.pg_extension e
  LEFT JOIN pg_catalog.pg_namespace n ON n.oid = e.extnamespace
  LEFT JOIN pg_catalog.pg_description c ON c.objoid = e.oid
    AND c.classoid = 'pg_catalog.pg_extension'::pg_catalog.regclass
WHERE
  e.extname LIKE '%postgis%'
ORDER BY
  1;
  Name | Version | Schema | Description
-----+-----+-----+
+-----+
 postgis | 2.3.7   | public  | PostGIS geometry, geography, and raster spatial types and
 functions
(1 row)
```

This mismatch between the default version (PostGIS 3.1.4) and the installed version (PostGIS 2.3.7) means that you need to upgrade the PostGIS extension.

```
ALTER EXTENSION postgis UPDATE;
ALTER EXTENSION
WARNING: unpackaging raster
WARNING: PostGIS Raster functionality has been unpackaged
```

3. Run the following query to verify that the raster functionality is now in its own package.

```
SELECT
  probin,
  count(*)
FROM
  pg_proc
WHERE
```

```

    probin LIKE '%postgis%'
GROUP BY
    probin;
    probin      | count
-----
$libdir/rtpostgis-2.3   | 107
$libdir/postgis-3       | 487
(2 rows)

```

The output shows that there's still a difference between versions. The PostGIS functions are version 3 (postgis-3), while the raster functions (rtpostgis) are version 2 (rtpostgis-2.3). To complete the upgrade, you run the upgrade command again, as follows.

```
postgres=> SELECT postgis_extensions_upgrade();
```

You can safely ignore the warning messages. Run the following query again to verify that the upgrade is complete. The upgrade is complete when PostGIS and all related extensions aren't marked as needing upgrade.

```
SELECT postgis_full_version();
```

4. Use the following query to see the completed upgrade process and the separately packaged extensions, and verify that their versions match.

```

SELECT
    e.extname AS "Name",
    e.extversion AS "Version",
    n.nspname AS "Schema",
    c.description AS "Description"
FROM
    pg_catalog.pg_extension e
    LEFT JOIN pg_catalog.pg_namespace n ON n.oid = e.extnamespace
    LEFT JOIN pg_catalog.pg_description c ON c.objoid = e.oid
        AND c.classoid = 'pg_catalog.pg_extension'::pg_catalog.regclass
WHERE
    e.extname LIKE '%postgis%'
ORDER BY
    1;
    Name      | Version | Schema | Description
-----
+-----+
postgis      | 3.1.5   | public  | PostGIS geometry, geography, and raster spatial
types and functions
postgis_raster | 3.1.5   | public  | PostGIS raster types and functions
(2 rows)

```

The output shows that the PostGIS 2 extension was upgraded to PostGIS 3, and both `postgis` and the now separate `postgis_raster` extension are version 3.1.5.

After this upgrade completes, if you don't plan to use the raster functionality, you can drop the extension as follows.

```
DROP EXTENSION postgis_raster;
```

# 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 more information about the extension, see [What is pg\\_cron?](#) in the pg\_cron documentation.

The pg\_cron extension is supported on Aurora PostgreSQL engine versions 12.6 and higher version

To learn more about using pg\_cron, see [Schedule jobs with pg\\_cron on your RDS for PostgreSQL or your Aurora PostgreSQL-Compatible Edition databases](#)

## Topics

- [Setting up the pg\\_cron extension \(p. 1327\)](#)
- [Granting permissions to use pg\\_cron \(p. 1327\)](#)
- [Scheduling pg\\_cron jobs \(p. 1328\)](#)
- [pg\\_cron reference \(p. 1330\)](#)

## Setting up the pg\_cron extension

Set up the pg\_cron extension as follows:

1. Modify the custom parameter group associated with your PostgreSQL DB instance by adding pg\_cron to the shared\_preload\_libraries parameter value.

Restart the PostgreSQL DB instance to have changes to the parameter group take effect. To learn more about working with parameter groups, see [Amazon Aurora PostgreSQL parameters \(p. 1369\)](#).

2. After the PostgreSQL DB instance has restarted, run the following command using an account that has rds\_superuser permissions. For example, if you used the default settings when you created your Aurora PostgreSQL DB cluster, 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 \(p. 1329\)](#).

## Granting permissions to use pg\_cron

Installing the pg\_cron extension requires the rds\_superuser privileges. However, permissions to use the pg\_cron can be granted (by a member of the rds\_superuser group/role) to other database users, so that they can schedule their own jobs. We recommend that you grant permissions to the cron schema only as needed if it improves operations in your production environment.

To grant a database user permission in the cron schema, run the following command:

```
postgres=> GRANT USAGE ON SCHEMA cron TO db-user;
```

This gives `db-user` permission to access the `cron` schema to schedule cron jobs for the objects that they have permissions to access. If the database user doesn't have permissions, the job fails after posting the error message to the `postgresql.log` file, as shown in the following:

```
2020-12-08 16:41:00 UTC::@[30647]:ERROR: permission denied for table table-name
2020-12-08 16:41:00 UTC::@[27071]:LOG: background worker "pg_cron" (PID 30647) exited with
exit code 1
```

IN other words, make sure that database users that are granted permissions on the `cron` schema also have permissions on the objects (tables, schemas, and so on) that they plan to schedule.

The details of the cron job and its success or failure are also captured in the `cron.job_run_details` table. For more information, see [The pg\\_cron tables \(p. 1332\)](#).

## Scheduling pg\_cron jobs

The following sections show how you can schedule various management tasks using `pg_cron` jobs.

### Note

When you create `pg_cron` jobs, check that the `max_worker_processes` setting is larger than the number of `cron.max_running_jobs`. A `pg_cron` job fails 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 \(p. 1330\)](#).

### Topics

- [Vacuuming a table \(p. 1328\)](#)
- [Purging the pg\\_cron history table \(p. 1329\)](#)
- [Disabling logging of pg\\_cron history \(p. 1329\)](#)
- [Scheduling a cron job for a database other than postgres \(p. 1329\)](#)

## 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
      |       |         |          |          |          | 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 querying the cron.job\_run\_details table to see failed jobs.

```
postgres=> SELECT * FROM cron.job_run_details WHERE status = 'failed';
jobid | runid | job_pid | database | username | command
      |       |          |          |          |           | status
      |       |          |          |          |           | 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 \(p. 1332\)](#).

## 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 \(p. 1331\)](#) 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 \(p. 1332\)](#).

## 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 \(p. 231\)](#).

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 \(p. 1330\)](#).

## 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 \(p. 1331\)](#).

```
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 \(p. 1330\)](#)
- [The cron.schedule\(\) function \(p. 1331\)](#)
- [The cron.unschedule\(\) function \(p. 1331\)](#)
- [The pg\\_cron tables \(p. 1332\)](#)

## The pg\_cron parameters

Following is a list of parameters that control the pg\_cron extension behavior.

Parameter	Description
<code>cron.database_name</code>	The database in which pg_cron metadata is kept.
<code>cron.host</code>	The hostname to connect to PostgreSQL. You can't modify this value.
<code>cron.log_run</code>	Log every job that runs in the <code>job_run_details</code> table. Values are <code>on</code> or <code>off</code> . For more information, see <a href="#">The pg_cron tables (p. 1332)</a> .
<code>cron.log_statement</code>	Log all cron statements before running them. Values are <code>on</code> or <code>off</code> .
<code>cron.max_running_jobs</code>	The maximum number of jobs that can run concurrently.
<code>cron.use_background_workers</code>	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 \(p. 1329\)](#).

The function has two syntax formats.

### Syntax

```
cron.schedule (job_name,  
              schedule,  
              command  
);  
  
cron.schedule (schedule,  
              command  
);
```

### Parameters

Parameter	Description
<code>job_name</code>	The name of the cron job.
<code>schedule</code>	Text indicating the schedule for the cron job. The format is the standard cron format.
<code>command</code>	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 specify either 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 <code>cron.schedule</code> function when the cron job was scheduled.
job_name	The name of a cron job that was scheduled with the <code>cron.schedule</code> 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><b>Important</b> We recommend that you don't give 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><b>Note</b> To prevent this table from growing indefinitely, purge it on a regular basis.</p>

Table	Description
	For an example, see <a href="#">Purging the pg_cron history table (p. 1329)</a> .

## 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 \(p. 1334\)](#)
- [Enabling the pg\\_partman extension \(p. 1334\)](#)
- [Configuring partitions using the create\\_parent function \(p. 1335\)](#)

- Configuring partition maintenance using the `run_maintenance_proc` function (p. 1336)

## 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 \(p. 1334\)](#). 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 \(p. 1334\)](#) 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 \(p. 231\)](#).
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 \(p. 1327\)](#).
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.

# Working with the supported foreign data wrappers for Amazon Aurora PostgreSQL

A foreign data wrapper (FDW) is a specific type of extension that provides access to external data. For example, the `oracle_fdw` extension allows your Aurora PostgreSQL DB instance to work with Oracle databases.

Following, you can find information about several supported PostgreSQL foreign data wrappers.

## Topics

- [Working with Oracle databases by using the oracle\\_fdw extension \(p. 1337\)](#)
- [Working with SQL Server databases by using the tds\\_fdw extension \(p. 1340\)](#)

## Working with Oracle databases by using the oracle\_fdw extension

To access an Oracle database from your Aurora PostgreSQL DB cluster you can install and use the `oracle_fdw` extension. This extension is a foreign data wrapper for Oracle databases. To learn more about this extension, see the [oracle\\_fdw](#) documentation.

The `oracle_fdw` extension is supported on Aurora PostgreSQL 12.7 (Amazon Aurora release 4.2) and higher versions.

## Topics

- [Turning on the oracle\\_fdw extension \(p. 1337\)](#)
- [Example: Using a foreign server linked to an Amazon RDS for Oracle database \(p. 1337\)](#)
- [Working with encryption in transit \(p. 1338\)](#)
- [Understanding the pg\\_user\\_mappings view and permissions \(p. 1338\)](#)

### 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 Amazon RDS for Oracle database

The following example shows the use of a foreign server linked to an Amazon RDS for Oracle database.

#### To create a foreign server linked to an RDS for Oracle database

1. Note the following on 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` privileges, 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 Amazon RDS is configured with `ACCEPTED`, meaning that the encryption depends on the Oracle database server configuration.

If your database is on RDS for Oracle, see [Oracle native network encryption](#) to configure the encryption.

## Understanding the `pg_user_mappings` view and permissions

The PostgreSQL catalog `pg_user_mapping` stores the mapping from an Aurora PostgreSQL user to the user on a foreign data (remote) server. Access to the catalog is restricted, but you use the `pg_user_mappings` view to see the mappings. In the following, you can find an example that shows how permissions apply with an example Oracle database, but this information applies more generally to any foreign data wrapper.

In the following output, you can find roles and permissions mapped to three different example users. Users `rdssu1` and `rdssu2` are members of the `rds_superuser` role, and `user1` isn't. The example uses the `psql` metacommand `\du` to list existing roles.

```
test=> \du
```

List of roles

Role name	Attributes
Member of	
rdssu1	
{rds_superuser}	
rdssu2	
{rds_superuser}	
user1	{}

All users, including users that have `rds_superuser` privileges, are allowed to view their own user mappings (`umoptions`) in the `pg_user_mappings` table. As shown in the following example, when `rdssu1` tries to obtain all user mappings, an error is raised even though `rdssu1`'s `rds_superuser` privileges:

```
test=> SELECT * FROM pg_user_mapping;
ERROR: permission denied for table pg_user_mapping
```

Following are some examples.

```
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 implementation differences between `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 an `rds_superuser` 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`.

## Working with SQL Server databases by using the tds\_fdw extension

You can use the PostgreSQL tds\_fdw extension to access databases that support the tabular data stream (TDS) protocol, such as Sybase and Microsoft SQL Server databases. This foreign data wrapper lets you connect from your Aurora PostgreSQL DB cluster to databases that use the TDS protocol, including Amazon RDS for Microsoft SQL Server. For more information, see [tds-fdw/tds\\_fdw](#) documentation on GitHub.

The tds\_fdw extension is supported on Amazon Aurora PostgreSQL version 13.6 and higher releases.

### Setting up your RDS for PostgreSQL DB to use the tds\_fdw extension

In the following procedures, you can find an example of setting up and using the tds\_fdw with an Aurora PostgreSQL DB cluster. Before you can connect to a SQL Server database using tds\_fdw, you need to get the following details for the instance:

- Hostname or endpoint. For an RDS for SQL Server DB instance, you can find the endpoint by using the Console. Choose the Connectivity & security tab and look in the "Endpoint and port" section.
- Port number. The default port number for Microsoft SQL Server is 1433.
- Name of the database. The DB identifier.

You also need to provide access on the security group or the access control list (ACL) for the SQL Server port, 1433. Both the Aurora PostgreSQL DB cluster and the RDS for SQL Server DB instance need access to port 1433. If access isn't configured correctly, when you try to query the Microsoft SQL Server you see the following error message:

```
ERROR: DB-Library error: DB #: 20009, DB Msg: Unable to connect:  
Adaptive Server is unavailable or does not exist (mssql2019.aws-region.rds.amazonaws.com),  
OS #: 0, OS Msg: Success, Level: 9
```

### To use tds\_fdw to connect to a SQL Server database

1. Connect to your Aurora PostgreSQL DB cluster's primary instance using an account that has the rds\_superuser role:

```
psql --host=your-cluster-name-instance-1.aws-region.rds.amazonaws.com --port=5432 --  
username=test --password
```

2. Install the tds\_fdw extension:

```
test=> CREATE EXTENSION tds_fdw;  
CREATE EXTENSION
```

After the extension is installed on your Aurora PostgreSQL DB cluster , you set up the foreign server.

### To create the foreign server

Perform these tasks on the Aurora PostgreSQL DB cluster using an account that has rds\_superuser privileges.

1. Create a foreign server in the Aurora PostgreSQL DB cluster:

```
test=> CREATE SERVER sqlserverdb FOREIGN DATA WRAPPER tds_fdw OPTIONS (servername  
'mssql2019.aws-region.rds.amazonaws.com', port '1433', database 'tds\_fdw\_testing');
```

```
CREATE SERVER
```

2. Grant permissions to a user who doesn't have `rds_superuser` role privileges, for example, `user1`:

```
test=> GRANT USAGE ON FOREIGN SERVER sqlserverdb TO user1;
```

3. Connect as `user1` and create a mapping to a SQL Server user:

```
test=> CREATE USER MAPPING FOR user1 SERVER sqlserverdb OPTIONS (username 'sqlserveruser', password 'password');  
CREATE USER MAPPING
```

4. Create a foreign table linked to a SQL Server table:

```
test=> CREATE FOREIGN TABLE mytab (a int) SERVER sqlserverdb OPTIONS (table 'MYTABLE');  
CREATE FOREIGN TABLE
```

5. Query the foreign table:

```
test=> SELECT * FROM mytab;  
a  
---  
1  
(1 row)
```

### Using encryption in transit for the connection

The connection from Aurora PostgreSQL to SQL Server uses encryption in transit (TLS/SSL) depending on the SQL Server database configuration. If the SQL Server isn't configured for encryption, the RDS for PostgreSQL client making the request to the SQL Server database falls back to unencrypted.

You can enforce encryption for the connection to RDS for SQL Server DB instances by setting the `rds.force_ssl` parameter. To learn how, see [Forcing connections to your DB instance to use SSL](#). For more information about SSL/TLS configuration for RDS for SQL Server, see [Using SSL with a Microsoft SQL Server DB instance](#).

## Amazon Aurora PostgreSQL reference

### Topics

- [Aurora PostgreSQL collations for EBCDIC and other mainframe migrations \(p. 1341\)](#)
- [Aurora PostgreSQL functions reference \(p. 1343\)](#)
- [Amazon Aurora PostgreSQL parameters \(p. 1369\)](#)
- [Amazon Aurora PostgreSQL wait events \(p. 1395\)](#)

## Aurora PostgreSQL collations for EBCDIC and other mainframe migrations

Migrating mainframe applications to new platforms such as AWS ideally preserves application behavior. To preserve application behavior on a new platform exactly as it was on the mainframe requires that migrated data be collated using the same collation and sorting rules. For example, many Db2 migration solutions shift null values to u0180 (Unicode position 0180), so these collations sort u0180 first. This is

one example of how collations can vary from their mainframe source and why it's necessary to choose a collation that better maps to the original EBCDIC collation.

Aurora PostgreSQL 14.3 and higher versions provide many ICU and EBCDIC collations to support such migration to AWS using the AWS Mainframe Modernization service. To learn more about this service, see [What is AWS Mainframe Modernization?](#)

In the following table, you can find Aurora PostgreSQL–provided collations. These collations follow EBCDIC rules and ensure that mainframe applications function the same on AWS as they did in the mainframe environment. The collation name includes the relevant code page, (cpnnnn), so that you can choose the appropriate collation for your mainframe source. For example, use en-US-cp037-x-icu for to achieve the collation behavior for EBCDIC data that originated from a mainframe application that used code page 037.

EBCDIC collations	AWS Blu Age collations	AWS Micro Focus collations
da-DK-cp1142-x-icu	da-DK-cp1142b-x-icu	da-DK-cp1142m-x-icu
da-DK-cp277-x-icu	da-DK-cp277b-x-icu	–
de-DE-cp1141-x-icu	de-DE-cp1141b-x-icu	de-DE-cp1141m-x-icu
de-DE-cp273-x-icu	de-DE-cp273b-x-icu	–
en-GB-cp1146-x-icu	en-GB-cp1146b-x-icu	en-GB-cp1146m-x-icu
en-GB-cp285-x-icu	en-GB-cp285b-x-icu	–
en-US-cp037-x-icu	en-US-cp037b-x-icu	–
en-US-cp1140-x-icu	en-US-cp1140b-x-icu	en-US-cp1140m-x-icu
es-ES-cp1145-x-icu	es-ES-cp1145b-x-icu	es-ES-cp1145m-x-icu
es-ES-cp284-x-icu	es-ES-cp284b-x-icu	–
fi-FI-cp1143-x-icu	fi-FI-cp1143b-x-icu	fi-FI-cp1143m-x-icu
fi-FI-cp278-x-icu	fi-FI-cp278b-x-icu	–
fr-FR-cp1147-x-icu	fr-FR-cp1147b-x-icu	fr-FR-cp1147m-x-icu
fr-FR-cp297-x-icu	fr-FR-cp297b-x-icu	–
it-IT-cp1144-x-icu	it-IT-cp1144b-x-icu	it-IT-cp1144m-x-icu
it-IT-cp280-x-icu	it-IT-cp280b-x-icu	–
nl-BE-cp1148-x-icu	nl-BE-cp1148b-x-icu	nl-BE-cp1148m-x-icu
nl-BE-cp500-x-icu	nl-BE-cp500b-x-icu	–

To learn more about AWS Blu Age, see [Tutorial: Managed Runtime for AWS Blu Age in the AWS Mainframe Modernization User Guide](#).

For more information about working with AWS Micro Focus, see [Tutorial: Managed Runtime for Micro Focus in the AWS Mainframe Modernization User Guide](#).

For more information about managing collations in PostgreSQL, see [Collation Support](#) 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\\_db\\_instance\\_identifier \(p. 1343\)](#)
- [aurora\\_ccm\\_status \(p. 1344\)](#)
- [aurora\\_global\\_db\\_instance\\_status \(p. 1346\)](#)
- [aurora\\_global\\_db\\_status \(p. 1348\)](#)
- [aurora\\_list\\_builtin \(p. 1350\)](#)
- [aurora\\_replica\\_status \(p. 1351\)](#)
- [aurora\\_stat\\_backend\\_waits \(p. 1354\)](#)
- [aurora\\_stat\\_dml\\_activity \(p. 1357\)](#)
- [aurora\\_stat\\_get\\_db\\_commit\\_latency \(p. 1359\)](#)
- [aurora\\_stat\\_system\\_waits \(p. 1361\)](#)
- [aurora\\_stat\\_wait\\_event \(p. 1362\)](#)
- [aurora\\_stat\\_wait\\_type \(p. 1364\)](#)
- [aurora\\_version \(p. 1365\)](#)
- [aurora\\_wait\\_report \(p. 1367\)](#)

## aurora\_db\_instance\_identifier

Reports the name of the DB instance name to which you're connected.

### Syntax

```
aurora_db_instance_identifier()
```

### Arguments

None

### Return type

VARCHAR string

### Usage notes

This function displays the name of Aurora PostgreSQL-Compatible Edition cluster's DB instance for your database client or application connection.

This function is available starting with the release of Aurora PostgreSQL versions 13.7, 12.11, 11.16, 10.21 and for all other later versions.

## Examples

The following example shows results of calling the `aurora_db_instance_identifier` function.

```
=> SELECT aurora_db_instance_identifier();aurora_db_instance_identifier
-----
 test-my-instance-name
```

You can join the results of this function with the `aurora_replica_status` function to obtain details about the DB instance for your connection. The [aurora\\_replica\\_status \(p. 1351\)](#) alone doesn't provide show you which DB instance you're using. The following example shows you how.

```
=> SELECT *
   FROM aurora_replica_status() rt,
        aurora_db_instance_identifier() di
  WHERE r.server_id = di;
-[ RECORD 1 ]-----+
server_id           | test-my-instance-name
session_id          | MASTER_SESSION_ID
durable_lsn         | 88492069
highest_lsn_rcvd    |
current_read_lsn    |
cur_replay_latency_in_usec |
active_txns         |
is_current          | t
last_transport_error | 0
last_error_timestamp |
last_update_timestamp | 2022-06-03 11:18:25+00
feedback_xmin       |
feedback_epoch       |
replica_lag_in_msec |
log_stream_speed_in_kib_per_second | 0
log_buffer_sequence_number | 0
oldest_read_view_trx_id |
oldest_read_view_lsn  |
pending_read_ios     | 819
```

## aurora\_ccm\_status

Displays the status of cluster cache manager.

### Syntax

```
aurora_ccm_status()
```

### Arguments

None.

### Return type

SETOF record with following columns:

- `buffers_sent_last_minute` – The number of buffers sent to the designated reader in the past minute.
- `buffers_found_last_minute` – The number of frequently access buffers during the past minute.

- `buffers_sent_last_scan` – The number of buffers sent to the designated reader during the last complete scan of the buffer cache.
- `buffers_found_last_scan` – The number of frequently accessed buffers sent during the last complete scan of the buffer cache. Buffers that are already cached on the designated reader aren't sent.
- `buffers_sent_current_scan` – The number of buffers sent during the current scan.
- `buffers_found_current_scan` – The number of frequently accessed buffers that were identified in the current scan.
- `current_scan_progress` – The number of buffers visited so far during the current scan.

## Usage notes

You can use this function to check and monitor the cluster cache management (CCM) feature. This function works only if CCM is active on your Aurora PostgreSQL DB cluster. To use this function you connect to the Write DB instance on your Aurora PostgreSQL DB cluster.

You turn on CCM for an Aurora PostgreSQL DB cluster by setting the `apg_ccm_enabled` to 1 in the cluster's custom DB cluster parameter group. To learn how, see [Configuring cluster cache management \(p. 1209\)](#).

Cluster cache management is active on an Aurora PostgreSQL DB cluster when the cluster has an Aurora Reader instance configured as follows:

- The Aurora Reader instance uses same DB instance class type and size as the cluster's Writer instance.
- The Aurora Reader instance is configured as Tier-0 for the cluster. If the cluster has more than one Reader, this is its only Tier-0 Reader.

Setting more than one Reader to Tier-0 disables CCM. When CCM is disabled, calling this function returns the following error message:

```
ERROR: Cluster Cache Manager is disabled
```

You can also the PostgreSQL `pg_buffercache` extension to analyze the buffer cache. For more information, see [pg\\_buffercache](#) in the PostgreSQL documentation.

For more information, see [Introduction to Aurora PostgreSQL cluster cache management](#).

## Examples

The following example shows the results of calling the `aurora_ccm_status` function. This first example shows CCM statistics.

```
=> SELECT * FROM aurora_ccm_status();
buffers_sent_last_minute | buffers_found_last_minute | buffers_sent_last_scan |
buffers_found_last_scan | buffers_sent_current_scan | buffers_found_current_scan |
current_scan_progress
-----+-----+-----+
-----+-----+-----+
-----+-----+-----+
          2242000 |           2242003 |         17920442 |
17923410 |           14098000 |         14100964 |
15877443 |
```

For more complete detail, you can use expanded display, as shown following:

```
\x
```

```
Expanded display is on.
SELECT * FROM aurora_ccm_status();
[ RECORD 1 ]-----+-----
buffers_sent_last_minute | 2242000
buffers_found_last_minute | 2242003
buffers_sent_last_scan | 17920442
buffers_found_last_scan | 17923410
buffers_sent_current_scan | 14098000
buffers_found_current_scan | 14100964
current_scan_progress | 15877443
```

This example shows how to check 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 ();
      warm_rate_kbps |      warm_percent
-----+-----
    16523 |        100.0
```

## [aurora\\_global\\_db\\_instance\\_status](#)

Displays the status of all Aurora instances, including replicas in an Aurora global DB cluster.

### Syntax

```
aurora_global_db_instance_status()
```

### Arguments

None

### Return type

SETOF record with following columns:

- **server\_id** – The identifier of the DB instance.
- **session\_id** – A unique identifier for the current session. A value of **MASTER\_SESSION\_ID** identifies the Writer (primary) DB instance.
- **aws\_region** – The AWS Region in which this global DB instance runs. For a list of Regions, see [Region availability \(p. 12\)](#).
- **durable\_lsn** – The log sequence number (LSN) made durable in storage. 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.
- **highest\_lsn\_rcvd** – The highest LSN received by the DB instance from the writer DB instance.
- **feedback\_epoch** – The epoch that the DB instance uses when it generates hot standby information. A *hot standby* is a DB instance that supports connections and queries while the primary DB is in recovery or standby mode. The hot standby information includes the epoch (point in time) and other details about the DB instance that's being used as a hot standby. For more information, see [Hot Standby](#) in the PostgreSQL documentation.
- **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 in milliseconds.

## Usage notes

This function shows replication statistics for an Aurora DB cluster. For each Aurora PostgreSQL DB instance in the cluster, the function shows a row of data that includes any cross-region replicas in a global database configuration.

You can run this function from any instance in an Aurora PostgreSQL DB cluster or an Aurora PostgreSQL global database. The function returns details about lag for all replica instances.

To learn more about monitoring lag using this function (`aurora_global_db_instance_status`) or by using `aurora_global_db_status`, see [Monitoring Aurora PostgreSQL-based Aurora global databases \(p. 203\)](#).

For more information about Aurora global databases, see [Overview of Amazon Aurora global databases \(p. 151\)](#).

To get started with Aurora global databases, see [Getting started with Amazon Aurora global databases \(p. 154\)](#) or see [Amazon Aurora FAQs](#).

## Examples

This example shows cross-region instance stats.

```
=> SELECT *
   FROM aurora_global_db_instance_status();
    server_id          |           session_id
aws_region | durable_lsn | highest_lsn_rcvd | feedback_epoch | feedback_xmin |
oldest_read_view_lsn | visibility_lag_in_msec
-----+-----+-----+-----+
-----+-----+
-----+-----+
db-119-001-instance-01          | MASTER_SESSION_ID          | eu-
west-1  | 2534560273 | [NULL] | [NULL] | [NULL] |
[NULL] | [NULL]          | [NULL]          | [NULL]          | [NULL] |
db-119-001-instance-02          | 4ecff34d-d57c-409c-ba28-278b31d6fc40 | eu-
west-1  | 2534560266 | 2534560273 | 0 | 19669196 |
2534560266 | 6          | 3e8a20fc-be86-43d5-95e5-bdf19d27ad6b | eu-
west-1  | 2534560266 | 2534560273 | 0 | 19669196 |
2534560266 | 6          | fc1b0023-e8b4-4361-bede-2a7e926cead6 | eu-
west-1  | 2534560266 | 2534560273 | 0 | 19669196 |
2534560254 | 23         | 30319b74-3f08-4e13-9728-e02aa1aa8649 | eu-
west-1  | 2534560266 | 2534560273 | 0 | 19669196 |
2534560254 | 23         | a331ffbb-d982-49ba-8973-527c96329c60 | eu-
central-1 | 2534560254 | 2534560266 | 0 | 19669196 |
2534560247 | 996        | db-119-001-global-instance-1          | e0955367-7082-43c4-b4db-70674064a9da | eu-
west-2  | 2534560254 | 2534560266 | 0 | 19669196 |
2534560247 | 14         | db-119-001-global-instance-1-eu-west-2a | 1248dc12-d3a4-46f5-a9e2-85850491a897 | eu-
west-2  | 2534560254 | 2534560266 | 0 | 19669196 |
2534560247 | 0          |
```

This example shows how to check global replica lag in milliseconds.

```
=> SELECT CASE
      WHEN 'MASTER_SESSION_ID' = session_id THEN 'Primary'
      ELSE 'Secondary'
```

```

        END AS global_role,
        aws_region,
        server_id,
        visibility_lag_in_msec
    FROM aurora_global_db_instance_status()
    ORDER BY 1, 2, 3;
        global_role | aws_region | server_id | visibility_lag_in_msec
-----+-----+-----+
Primary | eu-west-1 | db-119-001-instance-01 | [NULL]
[NULL]
Secondary | eu-central-1 | db-119-001-global-instance-1 | 13
10
Secondary | eu-west-1 | db-119-001-instance-02 | 9
Secondary | eu-west-1 | db-119-001-instance-03 | 2
Secondary | eu-west-1 | db-119-001-instance-04 | 18
Secondary | eu-west-1 | db-119-001-instance-05 | 14
Secondary | eu-west-2 | db-119-001-global-instance-1 | 13
Secondary | eu-west-2 | db-119-001-global-instance-1-eu-west-2a |

```

This example shows how to check min, max and average lag per AWS Region from the global database configuration.

```

=> SELECT 'Secondary' global_role,
        aws_region,
        min(visibility_lag_in_msec) min_lag_in_msec,
        max(visibility_lag_in_msec) max_lag_in_msec,
        round(avg(visibility_lag_in_msec),0) avg_lag_in_msec
    FROM aurora_global_db_instance_status()
    WHERE aws_region NOT IN (SELECT aws_region
                                FROM aurora_global_db_instance_status()
                                WHERE session_id='MASTER_SESSION_ID')
                                GROUP BY aws_region
UNION ALL
SELECT 'Primary' global_role,
        aws_region,
        NULL,
        NULL,
        NULL
    FROM aurora_global_db_instance_status()
    WHERE session_id='MASTER_SESSION_ID'
ORDER BY 1, 5;
        global_role | aws_region | min_lag_in_msec | max_lag_in_msec | avg_lag_in_msec
-----+-----+-----+-----+
Primary | eu-west-1 | [NULL] | [NULL] | [NULL]
Secondary | eu-central-1 | 133 | 133 | 133
Secondary | eu-west-2 | 0 | 495 | 248

```

## [aurora\\_global\\_db\\_status](#)

Displays information about various aspects of Aurora global database lag, specifically, lag of the underlying Aurora storage (so called durability lag) and lag between the recovery point objective (RPO).

This function is available for the following Aurora PostgreSQL versions:

- PostgreSQL 11.7 (Aurora PostgreSQL release 3.2) and higher versions

- PostgreSQL 10.11 (Aurora PostgreSQL release 2.4) and higher versions

When this function is used with Aurora PostgreSQL DB versions earlier than PostgreSQL 11.7 (Aurora PostgreSQL release 3.2), it returns less information than when used with PostgreSQL 11.7 and later versions. PostgreSQL 11.7 (Aurora PostgreSQL release 3.2) and higher versions returned enhanced statistics to this function.

## Syntax

```
aurora_global_db_status()
```

## Arguments

None.

## Return type

SETOF record with following columns:

- `aws_region` – The AWS Region that this DB cluster is in. For a complete listing of AWS Regions by engine, see [Regions and Availability Zones \(p. 11\)](#).
- `highest_lsn_written` – The highest log sequence number (LSN) that currently exists 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 difference in the timestamp values between the `highest_lsn_written` on a secondary DB cluster and the `highest_lsn_written` on the primary DB cluster. A value of -1 identifies the primary global database of an Aurora global database.
- `rpo_lag_in_msec` – The recovery point objective (RPO) lag. The RPO lag is the time it takes for the most recent user transaction COMMIT to be stored on a secondary DB cluster after it's been stored on the primary DB cluster of an Aurora global database. A value of -1 denotes the primary global database (and thus, lag isn't relevant).

In simple terms, this metric calculates the recovery point objective for each Aurora PostgreSQL DB cluster in an Aurora global database, that is, how much data might be lost if there were an outage. As with lag, RPO is measured in time.

- `last_lag_calculation_time` – The timestamp that specifies when values were last calculated for `replication_lag_in_msec` and `rpo_lag_in_msec`. A time value such as 1970-01-01 00:00:00+00 means this is the primary global database.
- `feedback_epoch` – The epoch that the secondary DB cluster uses when it generates hot standby information. A *hot standby* is a DB instance that supports connections and queries while the primary DB is in recovery or standby mode. The hot standby information includes the epoch (point in time) and other details about the DB instance that's being used as a 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.

## Usage notes

This function shows replication statistics for an Aurora global database. It shows one row for each DB cluster in an Aurora PostgreSQL global database. You can run this function from any instance in your Aurora PostgreSQL global database.

To evaluate Aurora global database replication lag, which is the visible data lag, see [aurora\\_global\\_db\\_instance\\_status \(p. 1346\)](#).

To learn more about using `aurora_global_db_status` and `aurora_global_db_instance_status` to monitor Aurora global database lag, see [Monitoring Aurora PostgreSQL-based Aurora global databases \(p. 203\)](#). For more information about Aurora global databases, see [Overview of Amazon Aurora global databases \(p. 151\)](#).

## Examples

This example shows how to display cross-region storage statistics.

```
=> SELECT CASE
      WHEN '-1' = durability_lag_in_msec THEN 'Primary'
      ELSE 'Secondary'
    END AS global_role,
    *
  FROM aurora_global_db_status();
global_role | aws_region | highest_lsn_written | durability_lag_in_msec | rpo_lag_in_msec
| last_lag_calculation_time | feedback_epoch | feedback_xmin
-----+-----+-----+-----+-----+
-----+-----+-----+-----+
Primary | eu-west-1 | 131031557 | -1 | -1
| 1970-01-01 00:00:00+00 | 0 | 0
Secondary | eu-west-2 | 131031554 | 410 | 0
| 2021-06-01 18:59:36.124+00 | 0 | 12640
Secondary | eu-west-3 | 131031554 | 410 | 0
| 2021-06-01 18:59:36.124+00 | 0 | 12640
```

## aurora\_list\_builtin

Lists all available Aurora PostgreSQL built-in functions, along with brief descriptions and function details.

### Syntax

```
aurora_list_builtin()
```

### Arguments

None

### Return type

SETOF record

## Examples

The following example shows results from calling the `aurora_list_builtin` function.

```
=> SELECT *
FROM aurora_list_builtin();

          Name          | Result data type | Argument data
types      | Type | Volatility | Parallel | Security |
Description
-----+-----+-----+-----+-----+
```

```

aurora_version | text          |           | invoker | Amazon Aurora
                | func | stable    | safe      |
PostgreSQL-Compatible Edition version string
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
                                |               | .me text
                                |               |
aurora_list_builtins | SETOF record | OUT "Name" text, OUT "Result data
type" text, OUT "Argum.| func | stable    | safe      | invoker | Lists all Aurora
built-in functions
                                |               | .ent data types" text, OUT "Type"
text, OUT "Volatility" .|       |               |           |
                                |               | .text, OUT "Parallel" text, OUT
"Security" text, OUT "Des.|       |               |           |
                                |               | .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
                                |               | .bytes bigint
                                |               |
aurora_stat_get_db_commit_latency | bigint     | oid
                                | func | stable    | restricted | invoker | Per DB commit latency
in microsecs

```

## [aurora\\_replica\\_status](#)

Displays the status of all Aurora PostgreSQL reader nodes.

### Syntax

```
aurora_replica_status()
```

### Arguments

None

### Return type

SETOF record with the following columns:

- **server\_id** – The DB instance ID (identifier).
- **session\_id** – A unique identifier for the current session, returned for primary instance and reader instances as follows:
  - For the primary instance, **session\_id** is always `MASTER\_SESSION\_ID`.
  - For reader instances, **session\_id** is always the UUID (universally unique identifier) of the reader instance.
- **durable\_lsn** – The log sequence number (LSN) that's been saved in storage.
  - For the primary volume, the primary volume durable LSN (VDL) that's currently in effect.
  - For any secondary volumes, the VDL of the primary up to which the secondary has successfully been applied.

### Note

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 transaction that's occurred later in the sequence.

- `highest_lsn_rcvd` – The highest (most recent) LSN received by the DB instance from the writer DB instance.
- `current_read_lsn` – The LSN of the most recent snapshot that's been applied to all readers.
- `cur_replay_latency_in_usecs` – The number of microseconds that it's expected to take to replay the log on the secondary.
- `active_txns` – The number of currently active transactions.
- `is_current` – Not used.
- `last_transport_error` – Last replication error code.
- `last_error_timestamp` – Timestamp of last replication error.
- `last_update_timestamp` – Timestamp of last update to replica status.
- `feedback_xmin` – The hot standby feedback\_xmin of the replica. The minimum (oldest) active transaction ID used by the DB instance.
- `feedback_epoch` – The epoch the DB instance uses when it generates hot standby information.
- `replica_lag_in_msec` – Time that reader instance lags behind writer writer instance, in milliseconds.
- `log_stream_speed_in_kib_per_second` – The log stream throughput in kilobytes per second.
- `log_buffer_sequence_number` – The log buffer sequence number.
- `oldest_read_view_trx_id` – Not used.
- `oldest_read_view_lsn` – The oldest LSN used by the DB instance to read from storage.
- `pending_read_ios` – The outstanding page reads that are still pending on replica.
- `read_ios` – The total number of page reads on replica.
- `iops` – Not used.
- `cpu` – CPU usage by the replica process. Note that this isn't CPU usage by the instance but rather the process. For information about CPU usage by the instance, see [Instance-level metrics for Amazon Aurora \(p. 531\)](#).

## Usage notes

The `aurora_replica_status` function returns values from an Aurora PostgreSQL DB cluster's replica status manager. You can use this function to obtain information about the status of replication on your Aurora PostgreSQL DB cluster, including metrics for all DB instances in your Aurora DB cluster. For example, you can do the following:

- **Get information about the type of instance (writer, reader) in the Aurora PostgreSQL DB cluster** – You can obtain this information by checking the values of the following columns:
  - `server_id` – Contains the name of the instance that you specified when you created the instance. In some cases, such as for the primary (writer) instance, the name is typically created for you by appending `-instance-1` to the name that you create for your Aurora PostgreSQL DB cluster.
  - `session_id` – The `session_id` field indicates whether the instance is a reader or a writer. For a writer instance, `session_id` is always set to "MASTER\_SESSION\_ID". For a reader instance, `session_id` is set to the UUID of the specific reader.
- **Diagnose common replication issues, such as replica lag** – Replica lag is the time in milliseconds that the page cache of a reader instance is behind that of the writer instance. This lag occurs because Aurora clusters use asynchronous replication, as described in [Replication with Amazon Aurora \(p. 73\)](#). It's shown in the `replica_lag_in_msec` column in the results returned by this function. Lag can also occur when a query is cancelled due to conflicts with recovery on a standby server. You can check

`pg_stat_database_conflicts()` to verify that such a conflict is causing the replica lag (or not). For more information, see [The Statistics Collector](#) in the *PostgreSQL documentation*. To learn more about high availability and replication, see [Amazon Aurora FAQs](#).

Amazon CloudWatch stores `replica_lag_in_msec` results over time, as the `AuroraReplicaLag` metric. For information about using CloudWatch metrics for Aurora, see [Monitoring Amazon Aurora metrics with Amazon CloudWatch \(p. 452\)](#)

To learn more about troubleshooting Aurora read replicas and restarts, see [Why did my Amazon Aurora read replica fall behind and restart?](#) in the [AWS Support Center](#).

## Examples

The following example shows how to get the replication status of all instances in an Aurora PostgreSQL DB cluster:

```
=> SELECT *
FROM aurora_replica_status();
```

The following example shows the writer instance in the `docs-lab-apg-main` Aurora PostgreSQL DB cluster:

```
=> SELECT server_id,
CASE
    WHEN 'MASTER_SESSION_ID' = session_id THEN 'writer'
    ELSE 'reader'
END AS instance_role
FROM aurora_replica_status()
WHERE session_id = 'MASTER_SESSION_ID';
server_id      | instance_role
-----+-----
db-119-001-instance-01 | writer
```

The following example lists all reader instances in a cluster:

```
=> SELECT server_id,
CASE
    WHEN 'MASTER_SESSION_ID' = session_id THEN 'writer'
    ELSE 'reader'
END AS instance_role
FROM aurora_replica_status()
WHERE session_id <> 'MASTER_SESSION_ID';
server_id      | instance_role
-----+-----
db-119-001-instance-02 | reader
db-119-001-instance-03 | reader
db-119-001-instance-04 | reader
db-119-001-instance-05 | reader
(4 rows)
```

The following example lists all instances, how far each instance is lagging behind the writer, and how long since the last update:

```
=> SELECT server_id,
CASE
    WHEN 'MASTER_SESSION_ID' = session_id THEN 'writer'
    ELSE 'reader'
END AS instance_role,
replica_lag_in_msec AS replica_lag_ms,
```

```

    round(extract (epoch FROM (SELECT age(clock_timestamp(), last_update_timestamp))) *
1000) AS last_update_age_ms
FROM aurora_replica_status()
ORDER BY replica_lag_in_msec NULLS FIRST;
server_id | instance_role | replica_lag_ms | last_update_age_ms
-----+-----+-----+
db-124-001-instance-03 | writer | [NULL] | 1756
db-124-001-instance-01 | reader | 13 | 1756
db-124-001-instance-02 | reader | 13 | 1756
(3 rows)

```

## aurora\_stat\_backend\_waits

Displays statistics for wait activity for a specific backend process.

### Syntax

```
aurora_stat_backend_waits(pid)
```

### Arguments

**pid** – The ID for the backend process. You can obtain process IDs by using the `pg_stat_activity` view.

### Return type

SETOF record with following columns:

- **type\_id** – A number that denotes the type of wait event, such as 1 for a lightweight lock (`LWLock`), 3 for a lock, or 6 for a client session, to name some examples. These values become meaningful when you join the results of this function with columns from `aurora_stat_wait_type` function, as shown in the [Examples \(p. 1354\)](#).
- **event\_id** – An identifying number for the wait event. Join this value with the columns from `aurora_stat_wait_event` to obtain meaningful event names.
- **waits** – A count of the number of waits accumulated for the specified process ID.
- **wait\_time** – Wait time in milliseconds.

### Usage notes

You can use this function to analyze specific backend (session) wait events that occurred since a connection opened. To get more meaningful information about wait event names and types, you can combine this function `aurora_stat_wait_type` and `aurora_stat_wait_event`, by using JOIN as shown in the examples.

### Examples

This example shows all waits, types, and event names for a backend process ID 3027.

```

=> SELECT type_name, event_name, waits, wait_time
      FROM aurora_stat_backend_waits(3027)
  NATURAL JOIN aurora_stat_wait_type()
  NATURAL JOIN aurora_stat_wait_event();
type_name | event_name | waits | wait_time
-----+-----+-----+
LWLock   | ProcArrayLock | 3 | 27
LWLock   | wal_insert | 423 | 16336
LWLock   | buffer_content | 11840 | 1033634

```

LWLock	lock_manager	23821   5664506
Lock	tuple	10258   152280165
Lock	transactionid	78340   1239808783
Client	ClientRead	34072   17616684
IO	ControlFileSyncUpdate	2   0
IO	ControlFileWriteUpdate	4   32
IO	RelationMapRead	2   795
IO	WALWrite	36666   98623
IO	XactSync	4867   7331963

This example shows current and cumulative wait types and wait events for all active sessions (`pg_stat_activity state <> 'idle'`) (but without the current session that's invoking the function (`pid <> pg_backend_pid()`)).

```
=> SELECT a.pid,
        a.username,
        a.app_name,
        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,
                (aurora_stat_backend_waits(pid)).*
          FROM pg_stat_activity
         WHERE pid <> pg_backend_pid()
           AND state <> 'idle') a
  NATURAL JOIN aurora_stat_wait_type() wt
  NATURAL JOIN aurora_stat_wait_event() we;
    pid | username | app_name | current_wait_type | current_wait_event | current_state | wait_type | wait_event | waits | wait_time
-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
30099 | postgres | pgbench | Lock | transactionid | active | LWLock | wal_insert | 1937 | 29975
30099 | postgres | pgbench | Lock | transactionid | active | LWLock | buffer_content | 22903 | 760498
30099 | postgres | pgbench | Lock | transactionid | active | LWLock | lock_manager | 10012 | 223207
30099 | postgres | pgbench | Lock | transactionid | active | Lock | tuple | 20315 | 63081529
.
.
.
30099 | postgres | pgbench | Lock | transactionid | active | IO | WALWrite | 93293 | 237440
30099 | postgres | pgbench | Lock | transactionid | active | IO | XactSync | 13010 | 19525143
30100 | postgres | pgbench | Lock | transactionid | active | LWLock | ProcArrayLock | 6 | 53
30100 | postgres | pgbench | Lock | transactionid | active | LWLock | wal_insert | 1913 | 25450
30100 | postgres | pgbench | Lock | transactionid | active | LWLock | buffer_content | 22874 | 778005
.
```

30109   postgres   pgbench   IO			XactSync		active	
LWLock   ProcArrayLock		3	71			
30109   postgres   pgbench   IO			XactSync		active	
LWLock   wal_insert		1940	27741			
30109   postgres   pgbench   IO			XactSync		active	
LWLock   buffer_content		22962	776352			
30109   postgres   pgbench   IO			XactSync		active	
LWLock   lock_manager		9879	218826			
30109   postgres   pgbench   IO			XactSync		active	
Lock   tuple		20401	63581306			
30109   postgres   pgbench   IO			XactSync		active	
Lock   transactionid		50769	211645008			
30109   postgres   pgbench   IO			XactSync		active	
Client   ClientRead		89901	44192439			

This example shows current and the top three (3) cumulative wait type and wait events for all active sessions (`pg_stat_activity state <> 'idle'`) excluding current session (`pid <> pg_backend_pid()`).

=> SELECT top3.*						
FROM (SELECT a.pid,						
a.username,						
a.app_name,						
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,						
RANK() OVER (PARTITION BY pid ORDER BY a.wait_time DESC)						
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,						
(aurora_stat_backend_waits(pid)).*						
FROM pg_stat_activity						
WHERE pid <> pg_backend_pid()						
AND state <> 'idle')						
a						
NATURAL JOIN aurora_stat_wait_type() wt						
NATURAL JOIN aurora_stat_wait_event() we) top3						
WHERE RANK <=3;						
pid   username   app_name   current_wait_type   current_wait_event   current_state						
wait_type   wait_event   waits   wait_time   rank						
-----+-----+-----+-----+-----+-----+-----						
20567   postgres   psql   CPU			CPU		active	
LWLock   wal_insert		25000	67512003	1		
20567   postgres   psql   CPU			CPU		active	IO
WALWrite		3071758	1016961	2		
20567   postgres   psql   CPU			CPU		active	IO
BuffFileWrite		20750	184559	3		
27743   postgres   pgbench   Lock			transactionid		active	
Lock   transactionid		237350	1265580011	1		
27743   postgres   pgbench   Lock			transactionid		active	
Lock   tuple		93641	341472318	2		
27743   postgres   pgbench   Lock			transactionid		active	
Client   ClientRead		417556	204796837	3		
.						
.						
.						

27745   postgres   pgbench	IO	XactSync	active	
Lock   transactionid	238068   1265816822	1		
27745   postgres   pgbench	IO	XactSync	active	
Lock   tuple	93210   338312247	2		
27745   postgres   pgbench	IO	XactSync	active	
Client   ClientRead	419157   207836533	3		
27746   postgres   pgbench	Lock	transactionid	active	
Lock   transactionid	237621   1264528811	1		
27746   postgres   pgbench	Lock	transactionid	active	
Lock   tuple	93563   339799310	2		
27746   postgres   pgbench	Lock	transactionid	active	
Client   ClientRead	417304   208372727	3		

## 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)
```

### Arguments

*database\_oid*

The object ID (OID) of the database in the Aurora PostgreSQL cluster.

### Return type

SETOF record

### 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	0		
template1								
fault_test	66	452099			0	0	0	
0	0	0			0	0		
db_access_test	1	5982			0	0	0	
0	0	0			0	0		
postgres	42035	95179203			5752		2678832898	
21157	441883182488	2			1520			
mydb	71	453514			0		0	
1	190				1		152	

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)
---

## Arguments

### *database\_oid*

The object ID (OID) of the Aurora PostgreSQL database.

## Return type

SETOF record

## 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 \(p. 449\)](#) 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

<code>aurora_stat_system_waits()</code>
---

### Arguments

None

### Return type

SETOF record

### 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 \(p. 1395\)](#).

## Syntax

```
aurora_stat_wait_event()
```

## Arguments

None

## Return type

SETOF record with following columns:

- `type_id` – The ID of the type of wait event.
- `event_id` – The ID of the wait event.
- `type_name` – Wait type name
- `event_name` – Wait event name

## 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()
```

## Arguments

None

## Return type

SETOE record with following columns:

- `type_id` – The ID of the type of wait event.
  - `type_name` – Wait type name.

## 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
```

## aurora\_version

Returns the string value of the Amazon Aurora PostgreSQL-Compatible Edition version number.

### Syntax

```
aurora_version()
```

### Arguments

None

### Return type

CHAR or VARCHAR string

### Usage notes

This function displays the version of the Amazon Aurora PostgreSQL-Compatible Edition database engine. The version number is returned as a string formatted as `major.minor.patch`. For more information about Aurora PostgreSQL version numbers, see [Aurora version number \(p. 1413\)](#).

You can choose when to apply minor version upgrades by setting the maintenance window for your Aurora PostgreSQL DB cluster. To learn how, see [Maintaining an Amazon Aurora DB cluster \(p. 321\)](#).

Starting with the release of Aurora PostgreSQL versions 13.3, 12.8, 11.13, 10.18, and for all other later versions, Aurora version numbers follow PostgreSQL version numbers. For more information about all Aurora PostgreSQL releases, see [Amazon Aurora PostgreSQL updates](#) in the *Release Notes for Aurora PostgreSQL*.

## Examples

The following example shows the results of calling the `aurora_version` function on an Aurora PostgreSQL DB cluster running [PostgreSQL 12.7, Aurora PostgreSQL release 4.2](#) and then running the same function on a cluster running [Aurora PostgreSQL version 13.3](#).

```
=> SELECT * FROM aurora_version();  
aurora_version  
-----
```

```

4.2.2
SELECT * FROM aurora_version();
aurora_version
-----
13.3.0

```

This example shows how to use the function with various options to get more detail about the Aurora PostgreSQL version. This example has an Aurora version number that's distinct from the PostgreSQL version number.

```

=> SHOW SERVER_VERSION;
server_version
-----
12.7
(1 row)

=> SELECT * FROM aurora_version();
aurora_version
-----
4.2.2
(1 row)

=> SELECT current_setting('server_version') AS "PostgreSQL Compatibility";
PostgreSQL Compatibility
-----
12.7
(1 row)

=> SELECT version() AS "PostgreSQL Compatibility Full String";
PostgreSQL Compatibility Full String
-----
PostgreSQL 12.7 on aarch64-unknown-linux-gnu, compiled by aarch64-unknown-linux-gnu-gcc
(GCC) 7.4.0, 64-bit
(1 row)

=> SELECT 'Aurora: '
  || aurora_version()
  || ' Compatible with PostgreSQL: '
  || current_setting('server_version') AS "Instance Version";
Instance Version
-----
Aurora: 4.2.2 Compatible with PostgreSQL: 12.7
(1 row)

```

This next example uses the function with the same options in the previous example. This example doesn't have an Aurora version number that's distinct from the PostgreSQL version number.

```

=> SHOW SERVER_VERSION;
server_version
-----
13.3

=> SELECT * FROM aurora_version();
aurora_version
-----
13.3.0
=> SELECT current_setting('server_version') AS "PostgreSQL Compatibility";
PostgreSQL Compatibility
-----
13.3

```

```
=> SELECT version() AS "PostgreSQL Compatibility Full String";
PostgreSQL Compatibility Full String
-----
PostgreSQL 13.3 on x86_64-pc-linux-gnu, compiled by x86_64-pc-linux-gnu-gcc (GCC) 7.4.0,
64-bit
=> SELECT 'Aurora: '
    || aurora_version()
    || ' Compatible with PostgreSQL: '
    || current_setting('server_version') AS "Instance Version";
Instance Version
-----
Aurora: 13.3.0 Compatible with PostgreSQL: 13.3
```

## aurora\_wait\_report

This function shows wait event activity over a period of time.

### Syntax

```
aurora_wait_report([time])
```

### Arguments

*time (optional)*

The time in seconds. Default is 10 seconds.

### Return type

SETOF record with following columns:

- `type_name` – Wait type name
- `event_name` – Wait event name
- `wait` – Number of waits
- `wait_time` – Wait time in milliseconds
- `ms_per_wait` – Average milliseconds by the number of an wait
- `waits_per_xact` – Average waits by the number of one transaction
- `ms_per_xact` – Average milliseconds by the number of transactions

### Usage notes

This function is available as of Aurora PostgreSQL release 1.1 compatible with PostgreSQL 9.6.6 and higher versions.

To use this function, you need to first create the Aurora PostgreSQL `aurora_stat_utils` extension, as follows:

```
=> CREATE extension aurora_stat_utils;
CREATE EXTENSION
```

For more information about available Aurora PostgreSQL extension versions, see [Extension versions for Amazon Aurora PostgreSQL](#) in *Release Notes for Aurora PostgreSQL*.

This function calculates the instance-level wait events by comparing two snapshots of statistics data from `aurora_stat_system_waits()` function and `pg_stat_database` PostgreSQL Statistics Views.

For more information about `aurora_stat_system_waits()` and `pg_stat_database`, see [The Statistics Collector](#) in the *PostgreSQL documentation*.

When run, this function takes an initial snapshot, waits the number of seconds specified, and then takes a second snapshot. The function compares the two snapshots and returns the difference. This difference represents the instance's activity for that time interval.

On the writer instance, the function also displays the number of committed transactions and TPS (transactions per second). This function returns information at the instance level and includes all databases on the instance.

## Examples

This example shows how to create `aurora_stat_utils` extension to be able to use `aurora_log_report` function.

```
=> CREATE extension aurora_stat_utils;
CREATE EXTENSION
```

This example shows how to check wait report for 10 seconds.

```
=> SELECT *
   FROM aurora_wait_report();
NOTICE: committed 34 transactions in 10 seconds (tps 3)
 type_name | event_name      | waits | wait_time | ms_per_wait | waits_per_xact |
 ms_per_xact
-----+-----+-----+-----+-----+-----+
 Client    | ClientRead      | 26   | 30003.00 | 1153.961  | 0.76  |
 882.441
 Activity   | WalWriterMain   | 50   | 10051.32 | 201.026  | 1.47  |
 295.627
 Timeout    | PgSleep          | 1    | 10049.52 | 10049.516 | 0.03  |
 295.574
 Activity   | BgWriterHibernate | 1    | 10048.15 | 10048.153 | 0.03  |
 295.534
 Activity   | AutoVacuumMain   | 18   | 9941.66  | 552.314  | 0.53  |
 292.402
 Activity   | BgWriterMain      | 1    | 201.09   | 201.085  | 0.03  |
 5.914
 IO         | XactSync          | 15   | 25.34    | 1.690    | 0.44  |
 0.745
 IO         | RelationMapRead   | 12   | 0.54     | 0.045    | 0.35  |
 0.016
 IO         | WALWrite           | 84   | 0.21     | 0.002    | 2.47  |
 0.006
 IO         | DataFileExtend     | 1    | 0.02     | 0.018    | 0.03  |
 0.001
```

This example shows how to check wait report for 60 seconds.

```
=> SELECT *
   FROM aurora_wait_report(60);
NOTICE: committed 1544 transactions in 60 seconds (tps 25)
 type_name | event_name      | waits | wait_time | ms_per_wait | waits_per_xact |
 ms_per_xact
-----+-----+-----+-----+-----+-----+
 Lock      | transactionid    | 6422 | 477000.53 | 74.276   | 4.16  |
 308.938
 Client    | ClientRead      | 8265 | 270752.99 | 32.759   | 5.35  |
 175.358
```

Activity   CheckpointerMain 38.925	1   60100.25   60100.246   0.00
Timeout   PgSleep 38.924	1   60098.49   60098.493   0.00
Activity   WalWriterMain 38.867	296   60010.99   202.740   0.19
Activity   AutoVacuumMain 38.749	107   59827.84   559.139   0.07
Activity   BgWriterMain 38.097	290   58821.83   202.834   0.19
IO   XactSync 35.764	1295   55220.13   42.641   0.84
IO   WALWrite 30.966	6602259   47810.94   0.007   4276.07
Lock   tuple 19.353	473   29880.67   63.173   0.31
LWLock   buffer_mapping 2.293	142   3540.13   24.930   0.09
Activity   BgWriterHibernate 0.728	290   1124.15   3.876   0.19
IO   BuffFileRead 0.401	7615   618.45   0.081   4.93
LWLock   buffer_content 0.224	73   345.93   4.739   0.05
LWLock   lock_manager 0.124	62   191.44   3.088   0.04
IO   RelationMapRead 0.003	72   5.16   0.072   0.05
LWLock   ProcArrayLock 0.001	1   2.01   2.008   0.00
IO   ControlFileWriteUpdate 0.000	2   0.03   0.013   0.00
IO   DataFileExtend 0.000	1   0.02   0.018   0.00
IO   ControlFileSyncUpdate 0.000	1   0.00   0.000   0.00

## 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 parameter groups \(p. 215\)](#).

## 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 using 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 this same default DB cluster parameter group, see [Aurora PostgreSQL instance-level parameters \(p. 1384\)](#).

Parameter name	Description
<code>ansi_constraint_trigger_ordering</code>	Change the firing order of constraint triggers to be compatible with the ANSI SQL standard.
<code>ansi_force_foreign_key_checks</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_target</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_execution</code>	Enables batch-mode functions to process sets of rows at a time.
<code>apg_enable_correlated_any_transform</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_push_down</code>	Enables the use of semijoin filters for hash joins.
<code>apg_plan_mgmt.capture_plan_baselines</code>	Capture 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.

Parameter name	Description
<code>apg_plan_mgmt.plan_retention_period</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_execution_limit</code>	<del>Estimated total plan cost below which an Unapproved plan will be executed.</del>
<code>apg_plan_mgmt.use_plan_baselines</code>	Use only approved or fixed plans for managed statements.
<code>array_nulls</code>	Enable input of NULL elements in arrays.
<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.
<code>auto_explain.log_nested_statements</code>	Log nested statements.
<code>auto_explain.log_timing</code>	Collect timing data not just row counts.
<code>auto_explain.log_triggers</code>	Include trigger statistics in plans.
<code>auto_explain.log_verbose</code>	Use EXPLAIN VERBOSE for plan logging.
<code>auto_explain.sample_rate</code>	Fraction of queries to process.
<code>autovacuum</code>	Starts the autovacuum subprocess.
<code>autovacuum_analyze_scale_factor</code>	Number of tuple inserts updates or deletes prior to analyze as a fraction of reltuples.
<code>autovacuum_analyze_threshold</code>	Minimum number of tuple inserts updates or deletes prior to analyze.
<code>autovacuum_freeze_max_age</code>	Static. Age at which to autovacuum a table to prevent transaction ID wraparound.
<code>autovacuum_max_workers</code>	Static. Sets the maximum number of simultaneously running autovacuum worker processes.
<code>autovacuum_multixact_freeze_max_age</code>	Static. Multixact age at which to autovacuum a table to prevent multixact wraparound.
<code>autovacuum_naptime</code>	(s) Time to sleep between autovacuum runs.
<code>autovacuum_vacuum_cost_delay</code>	(ms) Vacuum cost delay in milliseconds for autovacuum.
<code>autovacuum_vacuum_cost_limit</code>	Vacuum cost amount available before napping for autovacuum.

Parameter name	Description
<code>autovacuum_vacuum_insert_scale_factor</code>	Number of tuple inserts prior to vacuum as a fraction of <code>reltuples</code> .
<code>autovacuum_vacuum_insert_threshold</code>	Minimum number of tuple inserts prior to vacuum or -1 to disable insert vacuums.
<code>autovacuum_vacuum_scale_factor</code>	Number of tuple updates or deletes prior to vacuum as a fraction of <code>reltuples</code> .
<code>autovacuum_vacuum_threshold</code>	Minimum number of tuple updates or deletes prior to vacuum.
<code>autovacuum_work_mem</code>	(kB) Sets the maximum memory to be used by each autovacuum worker process.
<code>babelfishpg_tsql.default_locale</code>	Static. Default locale to be used for collations created by <code>CREATE COLLATION</code> .
<code>babelfishpg_tds.port</code>	Static. Sets the TDS TCP port the server listens on.
<code>babelfishpg_tds.tds_debug_log_level</code>	Sets logging level in TDS 0 disables logging
<code>babelfishpg_tds.tds_default_numeric_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_numeric_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_protocol_version</code>	Sets the maximum SSL/TLS protocol version to use for tds session.
<code>babelfishpg_tds.tds_ssl_min_protocol_version</code>	Sets the minimum SSL/TLS protocol version to use for tds session.
<code>babelfishpg_tsql.migration_mode</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 <code>CREATE FUNCTION</code> .
<code>client_min_messages</code>	Sets the message levels that are sent to the client.

Parameter name	Description
<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 <code>job_run_details</code> 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.
<code>deadlock_timeout</code>	(ms) Sets the time to wait on a lock before checking for deadlock.
<code>debug_pretty_print</code>	Indents parse and plan tree displays.
<code>debug_print_parse</code>	Logs each querys parse tree.
<code>debug_print_plan</code>	Logs each querys execution plan.
<code>debug_print_rewritten</code>	Logs each querys rewritten parse tree.
<code>default_statistics_target</code>	Sets the default statistics target.
<code>default_transaction_deferrable</code>	Sets the default deferrable status of new transactions.
<code>default_transaction_isolation</code>	Sets the transaction isolation level of each new transaction.
<code>default_transaction_read_only</code>	Sets the default read-only status of new transactions.
<code>effective_cache_size</code>	(8kB) Sets the planners assumption about the size of the disk cache.
<code>effective_io_concurrency</code>	Number of simultaneous requests that can be handled efficiently by the disk subsystem.
<code>enable_bitmapscan</code>	Enables the planners use of bitmap-scan plans.
<code>enable_gathermerge</code>	Enables the planners use of gather merge plans.
<code>enable_hashagg</code>	Enables the planners use of hashed aggregation plans.

Parameter name	Description
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_aggregate	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.
extra_float_digits	Sets the number of digits displayed for floating-point values.
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.
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.

Parameter name	Description
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.
hot_standby_feedback	Allows feedback from a hot standby to the primary that will avoid query conflicts.
huge_pages	Static. Use of huge pages on Linux.
idle_in_transaction_session_timeout	(ms) Sets the maximum allowed duration of any idling transaction.
intervalstyle	Sets the display format for interval values.
joinCollapse_limit	Sets the FROM-list size beyond which JOIN constructs are not flattened.
lo_compat_privileges	Enables backward compatibility mode for privilege checks on large objects.
log_autovacuum_min_duration	(ms) Sets the minimum execution time above which autovacuum actions will be logged.
log_connections	Logs each successful connection.
log_destination	Sets the destination for server log output.
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_filename	Sets the file name pattern for log files.
log_hostname	Logs the host name in the connection logs.
log_lock_waits	Logs long lock waits.
log_min_duration_sample	(ms) Sets the minimum execution time above which a sample of statements will be logged. Sampling is determined by log_statement_sample_rate.
log_min_duration_statement	(ms) Sets the minimum execution time above which statements will be logged.
log_min_error_statement	Causes all statements generating error at or above this level to be logged.
log_min_messages	Sets the message levels that are logged.

Parameter name	Description
<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_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.
<code>max_locks_per_transaction</code>	Static. Sets the maximum number of locks per transaction.

Parameter name	Description
<code>max_logical_replication_workers</code>	Static. Maximum number of logical replication worker processes.
<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_prepared_transactions</code>	Static. Sets the maximum number of simultaneously prepared transactions.
<code>max_replication_slots</code>	Static. Sets the maximum number of replication slots that the server can support.
<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_sync_workers_per_subscription</code>	Maximum number of synchronization workers per subscription
<code>max_wal_senders</code>	Static. Sets the maximum number of simultaneously running WAL sender processes.
<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.

Parameter name	Description
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.
password_encryption	Encrypt passwords.
pg_bigm.enable_recheck	It specifies whether to perform Recheck which is an internal process of full text search.
pg_bigm.gin_key_limit	It specifies the maximum number of 2-grams of the search keyword to be used for full text search.
pg_hint_plan.debug_print	Logs results of hint parsing.
pg_hint_plan.enable_hint	Force planner to use plans specified in the hint comment preceding to the query.
pg_hint_plan.enable_hint_table	Force planner to not get hint by using table lookups.
pg_hint_plan.message_level	Message level of debug messages.
pg_hint_plan.parse_messages	Message level of parse errors.
pg_prewarm.autoprewarm	Starts the autoprewarm worker.
pg_prewarm.autoprewarm_interval	Sets the interval between dumps of shared buffers
pg_stat_statements.max	Static. Sets the maximum number of statements tracked by pg_stat_statements.
pg_stat_statements.save	Save pg_stat_statements statistics across server shutdowns.
pg_stat_statements.track	Selects which statements are tracked by pg_stat_statements.
pg_stat_statements.track_planning	Selects whether planning duration is tracked by pg_stat_statements.
pg_stat_statements.track_utility	Selects whether utility commands are tracked by pg_stat_statements.
pgaudit.log	Specifies which classes of statements will be logged by session audit logging.
pgaudit.log_catalog	Specifies that session logging should be enabled in the case where all relations in a statement are in pg_catalog.
pgaudit.log_level	Specifies the log level that will be used for log entries.

Parameter name	Description
<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.
<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>	See log messages for RDS admin user actions in customer databases.

Parameter name	Description
rds.force_autovacuum_logging_level	See log messages related to autovacuum operations.
rds.force_ssl	Force SSL connections.
rds.global_db_rpo	(s) Recovery point objective threshold in seconds that blocks user commits when it is violated.
rds.log_retention_period	Amazon RDS will delete PostgreSQL log that are older than N minutes.
rds.logical_replication	Static. Enables logical decoding.
rds.pg_stat_ramdisk_size	Static. Size of the stats ramdisk in MB. A nonzero value will setup the ramdisk.
rds.rds_superuser_reserved_connections	Static. Sets the number of connection slots reserved for rds_superusers.
rds.restrict_password_commands	Static. Restricts password-related commands to members of rds_password
restart_after_crash	Reinitialize server after backend crash.
row_security	Enable row security.
seq_page_cost	Sets the planners estimate of the cost of a sequentially fetched disk page.
session_replication_role	Sets the sessions behavior for triggers and rewrite rules.
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_factor	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.

Parameter name	Description
<code>vacuum_freeze_table_age</code>	Age at which VACUUM should scan whole table to freeze tuples.
<code>vacuum_multixact_freeze_min_age</code>	Minimum age at which VACUUM should freeze a MultiXactId in a table row.
<code>vacuum_multixact_freeze_table_age</code>	Multixact age at which VACUUM should scan whole table to freeze tuples.
<code>wal_buffers</code>	Static. (8kB) Sets the number of disk-page buffers in shared memory for WAL.
<code>wal_receiver_create_temp_slot</code>	Sets whether a WAL receiver should create a temporary replication slot if no permanent slot is configured.
<code>wal_receiver_status_interval</code>	(s) Sets the maximum interval between WAL receiver status reports to the primary.
<code>wal_receiver_timeout</code>	(ms) Sets the maximum wait time to receive data from the primary.
<code>wal_sender_timeout</code>	(ms) Sets the maximum time to wait for WAL replication.
<code>work_mem</code>	(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 this same default DB parameter group, see [Aurora PostgreSQL cluster-level parameters \(p. 1372\)](#).

Parameter name	Description
<code>apg_enable_batch_mode_function_execution</code>	Enables batch-mode functions to process sets of rows at a time.
<code>apg_enable_correlated_any_transform</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_push_down</code>	Enables the use of semijoin filters for hash joins.
<code>apg_plan_mgmt.capture_plan_baselines</code>	Capture 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_period</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_execution_limit</code>	Estimated total plan cost below which an Unapproved plan will be executed.
<code>apg_plan_mgmt.use_plan_baselines</code>	Use 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.

Parameter name	Description
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	Log 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	Static. Sets the host name or IP address(es) to listen TDS to.
babelfishpg_tds.tds_debug_log_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

Parameter name	Description
<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.
<code>deadlock_timeout</code>	(ms) Sets the time to wait on a lock before checking for deadlock.
<code>debug_pretty_print</code>	Indents parse and plan tree displays.
<code>debug_print_parse</code>	Logs each querys parse tree.
<code>debug_print_plan</code>	Logs each querys execution plan.
<code>debug_print_rewritten</code>	Logs each querys rewritten parse tree.
<code>default_statistics_target</code>	Sets the default statistics target.
<code>default_transaction_deferrable</code>	Sets the default deferrable status of new transactions.
<code>default_transaction_isolation</code>	Sets the transaction isolation level of each new transaction.
<code>default_transaction_read_only</code>	Sets the default read-only status of new transactions.
<code>effective_cache_size</code>	(8kB) Sets the planners assumption about the size of the disk cache.
<code>effective_ioConcurrency</code>	Number of simultaneous requests that can be handled efficiently by the disk subsystem.
<code>enable_bitmapscan</code>	Enables the planners use of bitmap-scan plans.
<code>enable_gathermerge</code>	Enables the planners use of gather merge plans.
<code>enable_hashagg</code>	Enables the planners use of hashed aggregation plans.
<code>enable_hashjoin</code>	Enables the planners use of hash join plans.
<code>enable_incremental_sort</code>	Enables the planners use of incremental sort steps.
<code>enable_indexonlyscan</code>	Enables the planners use of index-only-scan plans.
<code>enable_indexscan</code>	Enables the planners use of index-scan plans.
<code>enable_material</code>	Enables the planners use of materialization.
<code>enable_mergejoin</code>	Enables the planners use of merge join plans.
<code>enable_nestloop</code>	Enables the planners use of nested-loop join plans.
<code>enable_parallel_append</code>	Enables the planners use of parallel append plans.
<code>enable_parallel_hash</code>	Enables the planners user of parallel hash plans.

Parameter name	Description
<code>enable_partition_pruning</code>	Enable plan-time and run-time partition pruning.
<code>enable_partitionwise_aggregate</code>	Enables partitionwise aggregation and grouping.
<code>enable_partitionwise_join</code>	Enables partitionwise join.
<code>enable_seqscan</code>	Enables the planners use of sequential-scan plans.
<code>enable_sort</code>	Enables the planners use of explicit sort steps.
<code>enable_tidscan</code>	Enables the planners use of TID scan plans.
<code>escape_string_warning</code>	Warn about backslash escapes in ordinary string literals.
<code>exit_on_error</code>	Terminate session on any error.
<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 <code>work_mem</code> to use for hash tables.
<code>hba_file</code>	Static. Sets the servers hba configuration file.
<code>hot_standby_feedback</code>	Allows feedback from a hot standby to the primary that will avoid query conflicts.
<code>ident_file</code>	Static. Sets the servers ident configuration file.
<code>idle_in_transaction_session_timeout</code>	(ms) Sets the maximum allowed duration of any idling transaction.
<code>joinCollapse_limit</code>	Sets the FROM-list size beyond which JOIN constructs are not flattened.

Parameter name	Description
<code>lc_messages</code>	Sets the language in which messages are displayed.
<code>listen_addresses</code>	Static. Sets the host name or IP address(es) to listen to.
<code>lo_compat_privileges</code>	Enables backward compatibility mode for privilege checks on large objects.
<code>log_connections</code>	Logs each successful connection.
<code>log_destination</code>	Sets the destination for server log output.
<code>log_directory</code>	Sets the destination directory for log files.
<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_file_mode</code>	Sets the file permissions for log files.
<code>log_filename</code>	Sets the file name pattern for log files.
<code>logging_collector</code>	Static. Start a subprocess to capture stderr output and/or csvlogs into log files.
<code>log_hostname</code>	Logs the host name in the connection logs.
<code>logical_decoding_work_mem</code>	(kB) This much memory can be used by each internal reorder buffer before spilling to disk.
<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.

Parameter name	Description
<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_ioConcurrency</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.
<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.

Parameter name	Description
<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 <code>pg_catalog</code> .
<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>	Force 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>	connection 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.

Parameter name	Description
pg_similarity.cosine_tokenizer	Sets the tokenizer for Cosine similarity function.
pg_similarity.dice_is_normalized	Sets if the result value is normalized or not.
pg_similarity.dice_threshold	Sets the threshold used by the Dice similarity measure.
pg_similarity.dice_tokenizer	Sets the tokenizer for Dice similarity measure.
pg_similarity.euclidean_is_normalized	Sets if the result value is normalized or not.
pg_similarity.euclidean_threshold	Sets the threshold used by the Euclidean similarity measure.
pg_similarity.euclidean_tokenizer	Sets the tokenizer for Euclidean similarity measure.
pg_similarity.hamming_is_normalized	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_normalized	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_normalized	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_normalized	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_normalized	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_normalized	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_normalized	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.

Parameter name	Description
<code>pg_similarity.nw_gap_penalty</code>	Sets the gap penalty used by the Needleman-Wunsch similarity measure.
<code>pg_similarity.nw_is_normalized</code>	Sets if the result value is normalized or not.
<code>pg_similarity.nw_threshold</code>	Sets the threshold used by the Needleman-Wunsch similarity measure.
<code>pg_similarity.overlap_is_normalized</code>	Sets if the result value is normalized or not.
<code>pg_similarity.overlap_threshold</code>	Sets the threshold used by the Overlap Coefficient similarity measure.
<code>pg_similarity.overlap_tokenizer</code>	Sets the tokenizer for Overlap Coefficientsimilarity measure.
<code>pg_similarity.qgram_is_normalized</code>	Sets if the result value is normalized or not.
<code>pg_similarity.qgram_threshold</code>	Sets the threshold used by the Q-Gram similarity measure.
<code>pg_similarity.qgram_tokenizer</code>	Sets the tokenizer for Q-Gram measure.
<code>pg_similarity.swg_is_normalized</code>	Sets if the result value is normalized or not.
<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_planning</code>	Selects whether planning duration is tracked by <code>pg_stat_statements</code> .
<code>pg_stat_statements.track_utility</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.

Parameter name	Description
rds.log_retention_period	Amazon RDS will delete PostgreSQL log that are older than N minutes.
rds.pg_stat_ramdisk_size	Size of the stats ramdisk in MB. A nonzero value will setup the ramdisk.
rds.rds_superuser_reserved_connections	Sets the number of connection slots reserved for rds_superusers.
rds.superuser_variables	List of superuser-only variables for which we elevate rds_superuser modification statements.
restart_after_crash	Reinitialize server after backend crash.
row_security	Enable row security.
search_path	Sets the schema search order for names that are not schema-qualified.
seq_page_cost	Sets the planners estimate of the cost of a sequentially fetched disk page.
session_replication_role	Sets the sessions behavior for triggers and rewrite rules.
shared_buffers	(8kB) Sets the number of shared memory buffers used by the server.
shared_preload_libraries	Lists shared libraries to preload into server.
ssl_ca_file	Location of the SSL server authority file.
ssl_cert_file	Location of the SSL server certificate file.
ssl_key_file	Location of the SSL server private key file
standard_conforming_strings	Causes ... strings to treat backslashes literally.
statement_timeout	(ms) Sets the maximum allowed duration of any statement.
stats_temp_directory	Writes temporary statistics files to the specified directory.
superuser_reserved_connections	Static. Sets the number of connection slots reserved for superusers.
synchronize_seqscans	Enable synchronized sequential scans.
tcp_keepalives_count	Maximum number of TCP keepalive retransmits.
tcp_keepalives_idle	(s) Time between issuing TCP keepalives.
tcp_keepalives_interval	(s) Time between TCP keepalive retransmits.
temp_buffers	(8kB) Sets the maximum number of temporary buffers used by each session.

Parameter name	Description
<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 <code>pg_stat_activity.current_query</code> , 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 <code>expr=NULL</code> as <code>expr IS NULL</code> .
<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. To learn more about wait events and tuning your Aurora PostgreSQL DB cluster, see [Tuning with wait events for Aurora PostgreSQL \(p. 1152\)](#).

### **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 \(p. 1158\)](#).

**Client:ClientWrite**

A backend process is waiting to send more data to a PostgreSQL client. For more information, see [Client:ClientWrite \(p. 1160\)](#).

**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 \(p. 1161\)](#).

**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:BufFileRead**

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:BufFileRead and IO:BufFileWrite \(p. 1166\)](#).

**IO:BufFileWrite**

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:BufFileRead and IO:BufFileWrite \(p. 1166\)](#).

**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 \(p. 1171\)](#).

**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 \(p. 1177\)](#).

**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 \(p. 1179\)](#).

**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 finishing 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 \(p. 1180\)](#).

**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 \(p. 1182\)](#).

**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 \(p. 1184\)](#).

**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 \(p. 1187\)](#).

**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 \(p. 1189\)](#).

**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 \(BufferContent\) \(p. 1192\)](#).

### **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 \(p. 1193\)](#).

### **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 \(p. 1195\)](#).

### **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 \(p. 1196\)](#).

**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:multipxact\_member**

A process is waiting for I/O on a multipxact\_member buffer.

**Lwlock:multipxact\_offset**

A process is waiting for I/O on a multipxact 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 \(p. 1199\)](#).

#### **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 [The Statistics Collector > Wait Event tables](#) in the PostgreSQL documentation.

## 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 \(p. 1413\)](#)
- [Amazon Aurora PostgreSQL releases and engine versions \(p. 1414\)](#)
- [Extension versions for Amazon Aurora PostgreSQL \(p. 1415\)](#)
- [Upgrading the PostgreSQL DB engine for Aurora PostgreSQL \(p. 1415\)](#)
- [Aurora PostgreSQL long-term support \(LTS\) releases \(p. 1429\)](#)

## 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 \(p. 1414\)](#) 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 9.6.21 or 9.6.22 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 \(p. 1415\)](#).

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

Amazon Aurora PostgreSQL-Compatible Edition releases update regularly. Updates are applied to Aurora PostgreSQL DB clusters during system maintenance windows. The timing when updates are applied depends on the AWS Region and maintenance window setting for the DB cluster, and also the type of

update. 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 \(p. 1413\)](#).

For information about extensions and modules, see [Extension versions for Amazon Aurora PostgreSQL \(p. 1415\)](#).

For more information about Amazon Aurora available 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 \(p. 14\)](#).

For details about the PostgreSQL versions that are available on Aurora PostgreSQL, see the [Release Notes for Aurora PostgreSQL](#).

## Extension versions for Amazon Aurora PostgreSQL

You can install and configure various PostgreSQL extensions for use with Aurora PostgreSQL DB clusters. For example, you can use the PostgreSQL `pg_partman` extension to automate the creation and maintenance of table partitions. To learn more about this and other extensions available for Aurora PostgreSQL, see [Working with extensions and foreign data wrappers \(p. 1318\)](#).

For details about the PostgreSQL extensions that are supported on Aurora PostgreSQL, see [Extension versions for Amazon Aurora PostgreSQL](#) in [Release Notes for Aurora PostgreSQL](#).

## Upgrading the PostgreSQL DB engine for Aurora PostgreSQL

Amazon Aurora makes new versions of the PostgreSQL database engine available in AWS Regions only after extensive testing. You can upgrade your Aurora PostgreSQL DB clusters to the new version when it's available in your Region.

Depending on the version of Aurora PostgreSQL that your DB cluster is currently running, an upgrade to the new release is either a minor upgrade or a major upgrade. For example, upgrading an Aurora PostgreSQL 11.15 DB cluster to Aurora PostgreSQL 13.6 is a *major version upgrade*. Upgrading an Aurora PostgreSQL 13.3 DB cluster to Aurora PostgreSQL 13.7 is a *minor version upgrade*. In the following topics, you can find information about how to perform both types of upgrades.

### Contents

- [Overview of the Aurora PostgreSQL upgrade processes \(p. 1416\)](#)
- [How to perform a major version upgrade \(p. 1417\)](#)
  - [Getting a list of available versions in your AWS Region \(p. 1418\)](#)
  - [Before upgrading your production DB cluster to a new major version \(p. 1419\)](#)
  - [Upgrading the Aurora PostgreSQL engine to a new major version \(p. 1422\)](#)
    - [Major upgrades for global databases \(p. 1424\)](#)
- [How to perform minor version upgrades and apply patches \(p. 1425\)](#)

- Minor release upgrades and zero-downtime patching ([p. 1426](#))
- Upgrading the Aurora PostgreSQL engine to a new minor version ([p. 1427](#))
- Upgrading PostgreSQL extensions ([p. 1428](#))

## Overview of the Aurora PostgreSQL upgrade processes

The differences between major and minor version upgrades are as follows:

### Minor version upgrades and patches

Minor version upgrades and patches include only those changes that are backward-compatible with existing applications. Minor version upgrades and patches become available to you only after Aurora PostgreSQL tests and approves them.

Minor version upgrades can be applied for you automatically by Aurora. When you create a new Aurora PostgreSQL DB cluster, the **Enable minor version upgrade** option is preselected. Unless you turn off this option, minor version upgrades are applied automatically during your scheduled maintenance window. For more information about the automatic minor version upgrade (AMVU) option and how to modify your Aurora DB cluster to use it, see [Automatic minor version upgrades for Aurora DB clusters \(p. 327\)](#).

If the automatic minor version upgrade option isn't set for your Aurora PostgreSQL DB cluster, your Aurora PostgreSQL isn't automatically upgraded to the new minor version. Instead, when a new minor version is released in your AWS Region and your Aurora PostgreSQL DB cluster is running an older minor version, Aurora prompts you to upgrade. It does so by adding a recommendation to the maintenance tasks for your cluster.

Patches aren't considered an upgrade, and they aren't applied automatically. Aurora PostgreSQL prompts you to apply any patches by adding a recommendation to maintenance tasks for your Aurora PostgreSQL DB cluster. For more information, see [How to perform minor version upgrades and apply patches \(p. 1425\)](#).

#### Note

Patches that resolve security or other critical issues are also added as maintenance tasks.

However, these patches are required. Make sure to apply security patches to your Aurora PostgreSQL DB cluster when they become available in your pending maintenance tasks.

The upgrade process involves the possibility of brief outages as each instance in the cluster is upgraded to the new version. However, after Aurora PostgreSQL versions 14.3.1, 13.7.1, 12.11.1, 11.16.1, and 10.21.1 and other higher releases of these minor versions, the upgrade process uses the zero-downtime patching (ZDP) feature. This feature minimizes outages, and in most cases completely eliminates them. For more information, see [Minor release upgrades and zero-downtime patching \(p. 1426\)](#).

#### Note

ZDP isn't supported for Aurora PostgreSQL DB clusters that are configured as Aurora Serverless v2, Aurora Serverless v1, Aurora global databases, or Babelfish.

### Major version upgrades

Unlike for minor version upgrades and patches, Aurora PostgreSQL doesn't have an automatic major version upgrade option. New major PostgreSQL versions might contain database changes that aren't backward-compatible with existing applications. The new functionality can cause your existing applications to stop working correctly.

To prevent any issues, we strongly recommend that you follow the process outlined in [Before upgrading your production DB cluster to a new major version \(p. 1419\)](#) before upgrading the DB instances in your Aurora PostgreSQL DB clusters. First ensure that your applications can run on the

new version by following that procedure. Then you can manually upgrade your Aurora PostgreSQL DB cluster to the new version.

The upgrade process involves the possibility of brief outages as each instance in the cluster is upgraded to the new version. The preliminary planning process also takes time. We recommend that you always perform upgrade tasks during your cluster's maintenance window or when operations are minimal. For more information, see [How to perform a major version upgrade \(p. 1417\)](#).

**Note**

Both minor version upgrades and major version upgrades might involve brief outages. For that reason, we recommend strongly that you perform or schedule upgrades during your maintenance window or during other periods of low utilization.

## How to perform a major version upgrade

Major version upgrades might contain database changes that are not backward-compatible with previous versions of the database. New functionality in a new version can cause your existing applications to stop working correctly. To avoid issues, Amazon Aurora doesn't apply major version upgrades automatically. Rather, we recommend that you carefully plan for a major version upgrade by following these steps:

1. Choose the major version that you want from the list of available targets from those listed for your version in the table. You can get a precise list of versions available in your AWS Region for your current version by using the AWS CLI. For details, see [Getting a list of available versions in your AWS Region \(p. 1418\)](#).
2. Verify that your applications work as expected on a trial deployment of the new version. For information about the complete process, see [Before upgrading your production DB cluster to a new major version \(p. 1419\)](#).
3. After verifying that your applications work as expected on the trial deployment, you can upgrade your cluster. For details, see [Upgrading the Aurora PostgreSQL engine to a new major version \(p. 1422\)](#).

**Note**

Currently, you can't upgrade an Aurora PostgreSQL DB cluster running Babelfish to a new major version.

In the table, you can find the major version upgrades that are available for various Aurora PostgreSQL DB versions.

Current source version	Major upgrade targets											
13.7	14.3											
13.6	14.3	13.7										
13.5	14.3	13.7	13.6									
13.4	14.3	13.7	13.6	13.5								
13.3	14.3	13.7	13.6	13.5	13.4							
12.11	14.3	13.7										
12.10	13.7	13.6	12.11									
12.9	13.7	13.6	13.5	12.11	12.10							

<b>Current source version</b>	<b>Major upgrade targets</b>													
12.8	13.7	13.6	13.5	13.4	12.11	12.10	12.9							
12.7	13.7	13.6	13.5	13.4	13.3	12.11	12.10	12.9	12.8					
12.6	13.7	13.6	13.5	13.4	13.3	12.11	12.10	12.9	12.8	12.7				
12.4	13.7	13.6	13.5	13.4	13.3	12.11	12.10	12.9	12.8	12.7	12.6			
11.16	14.3	13.7	12.11											
11.15	13.6	12.11	12.10	11.16										
11.14	13.5	12.11	12.10	12.9	11.16	11.15								
11.13	13.4	12.11	12.10	12.9	12.8	11.16	11.15	11.14						
11.12	12.11	12.10	12.9	12.8	12.7	11.16	11.15	11.14	11.13					
11.11	12.11	12.10	12.9	12.8	12.7	12.6	11.16	11.15	11.14	11.13	11.12			
11.9	12.11	12.10	12.9	12.8	12.7	12.6	12.4	11.16	11.15	11.14	11.13	11.12	11.11	
10.21	14.3	13.7	12.11	11.16										
10.20	13.6	12.10	11.16	11.15	10.21									
10.19	13.5	12.9	11.16	11.15	11.14	10.21	10.20							
10.18	13.4	12.8	11.16	11.15	11.14	11.13	10.21	10.20	10.19					
10.17	11.16	11.15	11.14	11.13	11.12	10.21	10.20	10.19	10.18					
10.16	11.16	11.15	11.14	11.13	11.12	11.11	10.21	10.20	10.19	10.18	10.17			
10.14	11.16	11.15	11.14	11.13	11.12	11.11	11.9	10.20	10.19	10.18	10.17	10.16		
9.6.22	14.3	13.4	12.8	11.13										

For any version that you're considering, always check the availability of your cluster's DB instance class. For more information about DB instance classes, including which ones are Graviton2-based and which ones are Intel-based, see [Aurora DB instance classes \(p. 56\)](#).

## Getting a list of available versions in your AWS Region

You can get a list of engine versions available as upgrade targets for your Aurora PostgreSQL DB by querying your AWS Region using the following [describe-db-engine-versions](#) AWS CLI command, as follows.

For Linux, macOS, or Unix:

```
aws rds describe-db-engine-versions \
--engine aurora-postgresql \
--engine-version 12.10 \
--query 'DBEngineVersions[].[ValidUpgradeTarget[?IsMajorVersionUpgrade == `true`].{EngineVersion:EngineVersion}}' \
--output text
```

For Windows:

```
aws rds describe-db-engine-versions ^
--engine aurora-postgresql ^
--engine-version 12.10 ^
--query "DBEngineVersions[?ValidUpgradeTarget[?IsMajorVersionUpgrade == `true`].
{EngineVersion:EngineVersion}" ^
--output text
```

In some cases, the version that you want to upgrade to isn't a target for your current version. In such cases, use the information in the [versions table](#) to perform minor version upgrades until your cluster is at a version that has your chosen target in its row of targets.

## Before upgrading your production DB cluster to a new major version

Each new major version includes enhancements to the query optimizer that are designed to improve performance. However, your workload might include queries that result in a worse performing plan in the new version. That's why we recommend that you test and review performance before upgrading in production. You can manage query plan stability across versions by using the Query Plan Management (QPM) extension, as detailed in [Ensuring plan stability after a major version upgrade \(p. 1296\)](#).

Before upgrading your production Aurora PostgreSQL DB clusters to a new major version, we strongly recommend that you test the upgrade to verify that all your applications work correctly:

1. Have a version-compatible parameter group ready.

If you are using a custom DB instance or DB cluster parameter group, you can choose from two options:

- a. Specify the default DB instance, DB cluster parameter group, or both for the new DB engine version.
- b. 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 (used by Amazon Aurora to do the upgrade) can't persist this data type. To learn more about this utility, see [pg\\_upgrade](#) in 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');
```

- If you are upgrading an Aurora PostgreSQL version 10.18 or higher DB cluster that has the pgRouting extension installed, drop the extension before upgrading to version 12.4 or higher.

### 3. Drop logical replication slots.

The upgrade process can't proceed if the Aurora PostgreSQL DB cluster is using any logical replication slots. Logical replication slots are typically used for short-term data migration tasks, such as migrating data using AWS DMS or for replicating tables from the database to data lakes, BI tools, or other targets. Before upgrading, make sure that you know the purpose of any logical replication slots that exist, and confirm that it's okay to delete them. You can check for logical replication slots using the following query:

```
SELECT * FROM pg_replication_slots;
```

If logical replication slots are still being used, you shouldn't delete them, and you can't proceed with the upgrade. However, if the logical replication slots aren't needed, you can delete them using the following SQL:

```
SELECT pg_drop_replication_slot(slot_name);
```

### 4. 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 \(p. 373\)](#) for more information.

### 5. 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's currently installed.

```
ALTER EXTENSION PostgreSQL-extension UPDATE TO 'new-version'
```

For more information, see [Upgrading PostgreSQL extensions \(p. 1428\)](#). To learn more about upgrading PostGIS, see [Step 6: Upgrade the PostGIS extension \(p. 1323\)](#).

### 6. 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

### 7. 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.reftype 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');
```

#### 8. 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 \(p. 376\)](#) or [Cloning a volume for an Amazon Aurora DB cluster \(p. 280\)](#).

For more information, see [Upgrading the Aurora PostgreSQL engine to a new major version \(p. 1422\)](#).

#### 9. 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 [Upgrading the Aurora PostgreSQL engine to a new major version \(p. 1422\)](#).

##### 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 \(p. 597\)](#).

10 Upgrade PostgreSQL extensions. The PostgreSQL upgrade process doesn't upgrade any PostgreSQL extensions. For more information, see [Upgrading PostgreSQL extensions \(p. 1428\)](#).

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;
```

- We recommend that you test 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.

## Upgrading the Aurora PostgreSQL engine to a new major version

When you initiate the upgrade process to a new major version, Aurora PostgreSQL takes a snapshot of the Aurora DB cluster before it makes any changes to your cluster. This snapshot is created for major version upgrades only, not minor version upgrades. When the upgrade process completes, you can find this snapshot among the manual snapshots listed under **Snapshots** in the RDS console. The snapshot name includes `preupgrade` as its prefix, the name of your Aurora PostgreSQL DB cluster, the source version, the target version, and the date and timestamp, as shown in the following example.

```
preupgrade-docs-lab-apg-global-db-12-8-to-13-6-2022-05-19-00-19
```

After the upgrade completes, you can use the snapshot that Aurora created and stored in your manual snapshot list to restore the DB cluster to its previous version, if necessary.

### Tip

In general, snapshots provide many ways to restore your Aurora DB cluster to various points in time. To learn more, see [Restoring from a DB cluster snapshot \(p. 375\)](#) and [Restoring a DB cluster to a specified time \(p. 415\)](#).

During the major version upgrade process, Aurora allocates a volume and clones the source Aurora PostgreSQL DB cluster. If the upgrade fails for any reason, Aurora PostgreSQL uses the clone to roll back the upgrade. After more than 15 clones of a source volume are allocated, subsequent clones become full copies and take longer. This can cause the upgrade process also to take longer. If Aurora PostgreSQL rolls back the upgrade, be aware of the following:

- You might see billing entries and metrics for both the original volume and the cloned volume allocated during the upgrade. Aurora PostgreSQL cleans 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.

To safely upgrade the DB instances that make up your cluster, Aurora PostgreSQL uses the `pg_upgrade` utility. After the writer upgrade completes, each reader instance experiences a brief outage while it's upgraded to the new major version. To learn more about this PostgreSQL utility, see [pg\\_upgrade](#) in the PostgreSQL documentation.

You can upgrade your Aurora PostgreSQL DB cluster to a new version by using the AWS Management Console, the AWS CLI, or the RDS API.

### Console

#### To upgrade the engine version of 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 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 \(p. 248\)](#).
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 [modify-db-cluster](#) AWS CLI 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`.

## Major upgrades for global databases

For an Aurora global database cluster, the upgrade process upgrades all DB clusters that make up your Aurora global database at the same time. It does so to ensure that each runs the same Aurora PostgreSQL version. It also ensures that any changes to system tables, data file formats, and so on, are automatically replicated to all secondary clusters.

To upgrade a global database cluster to a new major version of Aurora PostgreSQL, we recommend that you test your applications on the upgraded version, as detailed in [Before upgrading your production DB cluster to a new major version \(p. 1419\)](#). Be sure to prepare your DB cluster parameter group and DB parameter group settings for each AWS Region in your Aurora global database before the upgrade as detailed in [step 1. of Before upgrading your production DB cluster to a new major version \(p. 1419\)](#).

If your Aurora PostgreSQL global database cluster has a recovery point objective (RPO) set for its `rds.global_db_rpo` parameter, make sure to reset the parameter before upgrading. The major version upgrade process doesn't work if the RPO is turned on. By default, this parameter is turned off. For more information about Aurora PostgreSQL global databases and RPO, see [Managing RPOs for Aurora PostgreSQL-based global databases \(p. 198\)](#).

If you verify that your applications can run as expected on the trial deployment of the new version, you can start the upgrade process. To do so, see [Upgrading the Aurora PostgreSQL engine to a new major version \(p. 1422\)](#). Be sure to choose the top-level item from the **Databases** list in the RDS console, **Global database**, as shown in the following image.

DB identifier	Role	Engine	Region & AZ	Size
docs-lab-apg-aiml	Regional cluster	Aurora PostgreSQL	us-west-1	2 instances
docs-lab-apg-global-db	Global database	Aurora PostgreSQL	2 regions	2 clusters
docs-lab-apg-global-12-7	Primary cluster	Aurora PostgreSQL	us-west-1	2 instances
docs-lab-apg-global-12-7-instance-1	Writer instance	Aurora PostgreSQL	us-west-1c	db.r6g.large
docs-lab-apg-global-12-7-instance-1-us-west-1a	Reader instance	Aurora PostgreSQL	us-west-1a	db.r6g.large
docs-lab-apg-global-db-cluster-northwest	Secondary cluster	Aurora PostgreSQL	us-west-2	2 instances
docs-lab-apg-global-db-instance-north	Reader instance	Aurora PostgreSQL	us-west-2c	db.r6g.large
docs-lab-apg-global-db-instance-north-us-west-2b	Reader instance	Aurora PostgreSQL	us-west-2b	db.r6g.large
docs-lab-apg-main	Regional cluster	Aurora PostgreSQL	us-west-1	2 instances
docs-lab-apg-sless-test-aws-s3	Serverless	Aurora PostgreSQL	us-west-1	0 capacity units

As with any modification, you can confirm that you want the process to proceed when prompted.

RDS > Databases > Modify global database

## Modify global database: docs-lab-apg-global-db

**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 global database.

Attribute	Current value	New value
DB engine version	12.8	13.6
DB cluster parameter group	default.aurora-postgresql12	default.aurora-postgresql13
DB parameter group	default.aurora-postgresql12	default.aurora-postgresql13

**⚠ Potential unexpected downtime**  
This upgrade is applied immediately in an asynchronous fashion. If any pending modifications require rebooting your cluster, this upgrade can cause unexpected downtime.

**Note:**  
To schedule modifications in the next maintenance window, modify the DB cluster or DB instance individually.

Cancel Back **Modify global database**

Rather than using the console, you can start the upgrade process by using the AWS CLI or the RDS API. As with the console, you operate on the Aurora global database cluster rather than any of its constituents, as follows:

- Use the [modify-global-cluster](#) AWS CLI command to start the upgrade for your Aurora global database by using the AWS CLI.
- Use the [ModifyGlobalCluster](#) API to start the upgrade.

## How to perform minor version upgrades and apply patches

Minor version upgrades and patches become available in AWS Regions only after rigorous testing. Before releasing upgrades and patches, Aurora PostgreSQL tests to ensure that known security issues, bugs, and other issues that emerge after the release of the minor community version don't disrupt overall Aurora PostgreSQL fleet stability.

As Aurora PostgreSQL makes new minor versions available, the instances that make up your Aurora PostgreSQL DB cluster can be automatically upgraded during your specified maintenance window. For this to happen, your Aurora PostgreSQL DB cluster must have the **Enable auto minor version upgrade** option turned on. All DB instances that make up your Aurora PostgreSQL DB cluster must have the automatic minor version upgrade (AMVU) option turned on so that the minor upgrade to be applied throughout the cluster.

### Tip

Make sure that the **Enable auto minor version upgrade** option is turned on for all PostgreSQL DB instances that make up your Aurora PostgreSQL DB cluster. This option must be turned on for every instance in the DB cluster to work.

You can check the value of the **Enable auto minor version upgrade** option for all your Aurora PostgreSQL DB clusters by using the [describe-db-instances](#) AWS CLI command with the following query.

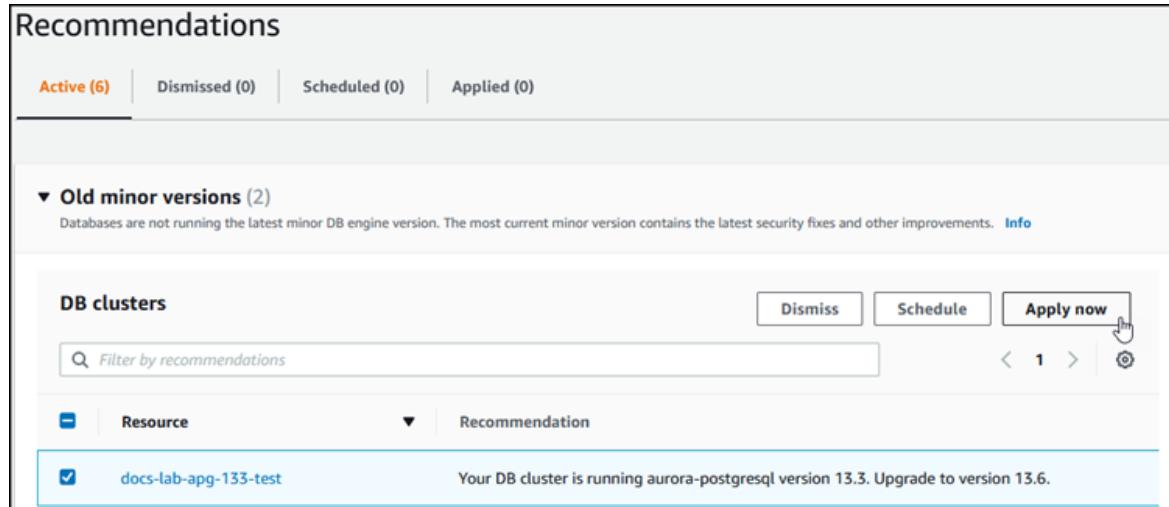
```
aws rds describe-db-instances \
--query '*[].
{DBClusterIdentifier:DBClusterIdentifier,DBInstanceIdentifier:DBInstanceIdentifier,AutoMinorVersionUpgrade:AutoMinorVersionUpgrade}'
```

This query returns a list of all Aurora DB clusters and their instances with a `true` or `false` value for the status of the `AutoMinorVersionUpgrade` setting. The command as shown assumes that you have your AWS CLI configured to target your default AWS Region.

For more information about the AMVU option and how to modify your Aurora DB cluster to use it, see [Automatic minor version upgrades for Aurora DB clusters \(p. 327\)](#).

You can upgrade your Aurora PostgreSQL DB clusters to new minor versions either by responding to maintenance tasks, or by modifying the cluster to use the new version.

You can identify any available upgrades or patches for your Aurora PostgreSQL DB clusters by using the RDS console and opening the **Recommendations** menu. There, you can find a list of various maintenance issues such as **Old minor versions**. Depending on your production environment, you can choose to **Schedule** the upgrade or take immediate action, by choosing **Apply now**, as shown following.



To learn more about how to maintain an Aurora DB cluster, including how to manually apply patches and minor version upgrades, see [Maintaining an Amazon Aurora DB cluster \(p. 321\)](#).

### Minor release upgrades and zero-downtime patching

Upgrading an Aurora PostgreSQL DB cluster involves the possibility of an outage. During the upgrade process, the database is shut down as it's being upgraded. If you start the upgrade while the database is busy, you lose all 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 improves the upgrading process. With ZDP, both minor version upgrades and patches can be applied with minimal impact to your Aurora PostgreSQL DB cluster. ZDP will be used when applying patches or newer minor version upgrades to Aurora PostgreSQL versions

14.3.1 or higher, 13.7.1 or higher, 12.11.1 or higher, 11.16.1 or higher, and 10.21.1 or higher. That is, upgrading to new minor versions from any of these releases onward uses ZDP.

**Note**

ZDP isn't supported for Aurora PostgreSQL DB clusters that are configured as Aurora Serverless v2, Aurora Serverless v1, Aurora global databases, or Babelfish.

ZDP works by preserving current client connections to your Aurora PostgreSQL DB cluster throughout the Aurora PostgreSQL upgrade process. When ZDP completes successfully, application sessions are preserved and the database engine restarts while the upgrade is still under way. Although the database engine restart can cause a drop in throughput, that typically lasts only for a few seconds or at most, approximately one minute.

In some cases, zero-downtime patching (ZDP) might not succeed. For example, if long-running queries or transactions are in progress, ZDP might need to cancel these to complete. If data definition language (DDL) statements are running or if temporary tables or table locks are in use for any other reason, ZDP might need to cancel the open transaction. Parameter changes that are in a pending state on your Aurora PostgreSQL DB cluster or its instances also interfere with ZDP.

You can find metrics and events for ZDP operations in **Events** page in the console. The events include the start of the ZDP upgrade and completion of the upgrade. In this event you can find how long the process took, and the numbers of preserved and dropped connections that occurred during the restart. You can find details in your database error log.

## Upgrading the Aurora PostgreSQL engine to a new minor version

You can upgrade your Aurora PostgreSQL DB cluster to a new minor version by using the console, the AWS CLI, or the RDS API. Before performing the upgrade, we recommend that you follow the same best practice that we recommend for major version upgrades. As with new major versions, new minor versions can also have optimizer improvements, such as fixes, that can cause query plan regressions. To ensure plan stability, we recommend that you use the Query Plan Management (QPM) extension as detailed in [Ensuring plan stability after a major version upgrade \(p. 1296\)](#).

### Console

#### To upgrade the engine version of your Aurora PostgreSQL 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 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 \(p. 248\)](#).
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 `modify-db-cluster` AWS CLI command with the following parameters:

- `--db-cluster-identifier` – The name of your Aurora PostgreSQL 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.
- `--no-apply-immediately` – Apply changes during the next maintenance window. To apply changes immediately, use `--apply-immediately` instead.

For Linux, macOS, or Unix:

```
aws rds modify-db-cluster \
  --db-cluster-identifier mydbcluster \
  --engine-version new_version \
  --no-apply-immediately
```

For Windows:

```
aws rds modify-db-cluster ^
  --db-cluster-identifier mydbcluster ^
  --engine-version new_version ^
  --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.
- `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`.

## Upgrading PostgreSQL extensions

Upgrading your Aurora PostgreSQL DB cluster to a new major version doesn't upgrade the PostgreSQL extensions at the same time. For most extensions, you upgrade the extension after the major version upgrade completes. However, in some cases, you upgrade the extension before you upgrade the Aurora PostgreSQL DB engine. For more information, see the [list of extensions to update](#) in [Before upgrading your production DB cluster to a new major version \(p. 1419\)](#).

Installing PostgreSQL extensions requires `rds_superuser` privileges. Typically, an `rds_superuser` delegates permissions over specific extensions to relevant users (roles), to facilitate the management of a given extension. That means that the task of upgrading all the extensions in your Aurora PostgreSQL DB cluster might involve many different users (roles). Keep this in mind especially 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 \(p. 1018\)](#).

### Note

For information about upgrading the PostGIS extension version, see [Managing spatial data with the PostGIS extension \(p. 1320\)](#) ([Step 6: Upgrade the PostGIS extension \(p. 1323\)](#)).

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 [PostgreSQL 12.9 in the Release Notes for Aurora PostgreSQL](#).
- 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 in the Release Notes for Aurora PostgreSQL](#).

For information about how to identify Aurora and database engine versions, see [Identifying versions of Amazon Aurora PostgreSQL \(p. 1413\)](#).

# 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.

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.

RDS Proxy is fully compatible with the engine versions that it supports. You can enable RDS Proxy for most applications with no code changes. For a list of supported engine versions, see [Amazon RDS Proxy \(p. 28\)](#).

## Topics

- [Region and version availability \(p. 1430\)](#)
- [Quotas and limitations for RDS Proxy \(p. 1431\)](#)
- [Planning where to use RDS Proxy \(p. 1432\)](#)
- [RDS Proxy concepts and terminology \(p. 1433\)](#)
- [Getting started with RDS Proxy \(p. 1437\)](#)
- [Managing an RDS Proxy \(p. 1448\)](#)
- [Working with Amazon RDS Proxy endpoints \(p. 1458\)](#)
- [Monitoring RDS Proxy metrics with Amazon CloudWatch \(p. 1467\)](#)
- [Working with RDS Proxy events \(p. 1472\)](#)
- [RDS Proxy command-line examples \(p. 1473\)](#)
- [Troubleshooting for RDS Proxy \(p. 1475\)](#)
- [Using RDS Proxy with AWS CloudFormation \(p. 1481\)](#)

## Region and version availability

For information about database engine version support and availability of RDS Proxy in a given AWS Region, see [Amazon RDS Proxy \(p. 28\)](#).

# 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 \(p. 1458\)](#).

For RDS DB instances in replication configurations, you can associate a proxy only with the writer DB instance, not a read replica.

- You can use RDS Proxy with Aurora Serverless v2 clusters but not with Aurora Serverless v1 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 \(p. 1462\)](#).

- 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 \(p. 1440\)](#) for details about IAM support in RDS Proxy.
- You can't use RDS Proxy with custom DNS.
- 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.
- Any statement with a text size greater than 16 KB causes the proxy to pin the session to the current connection.

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.
- RDS Proxy doesn't support MySQL compressed mode. For example, it doesn'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 \(p. 1455\)](#).

**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 following RDS Proxy limitations apply to PostgreSQL:

- RDS Proxy doesn't support session pinning filters for PostgreSQL.
- RDS Proxy doesn't support PostgreSQL SCRAM-SHA-256 authentication.
- 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 \(p. 1467\)](#).
- 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 typically open and close large numbers of database connections and don't have built-in connection pooling mechanisms are good candidates for using a proxy.
- 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 \(p. 1433\)](#)
- [Connection pooling \(p. 1434\)](#)
- [RDS Proxy security \(p. 1434\)](#)
- [Failover \(p. 1436\)](#)
- [Transactions \(p. 1436\)](#)

## 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 \(p. 1458\)](#).

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 \(p. 1634\)](#). 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.

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 the `.pem` file to use, download all root CA PEMs from [Amazon Trust Services](#) and place them into a single `.pem` file.

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 \(p. 1455\)](#).

## 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 \(p. 1437\)](#)
- [Setting up database credentials in AWS Secrets Manager \(p. 1438\)](#)
- [Setting up AWS Identity and Access Management \(IAM\) policies \(p. 1440\)](#)
- [Creating an RDS Proxy \(p. 1442\)](#)
- [Viewing an RDS Proxy \(p. 1445\)](#)
- [Connecting to a database through RDS Proxy \(p. 1446\)](#)

## 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 \(p. 87\)](#). You can also follow the tutorial in [Getting started with Amazon Aurora \(p. 93\)](#).

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][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 for Amazon Aurora \(p. 1653\)](#).

## 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 DB cluster, you can create a proxy. You can associate a proxy with an Aurora MySQL or Aurora PostgreSQL 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 family.** This setting determines which database network protocol the proxy recognizes when it interprets network traffic to and from the database. For Aurora MySQL, choose **MySQL**. For Aurora PostgreSQL, choose **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 \(p. 127\)](#). 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 \(p. 1454\)](#).
- **Session pinning filters.** (Optional) This is an advanced setting, for troubleshooting performance issues with particular applications. Currently, the setting isn't supported for PostgreSQL and 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 \(p. 1455\)](#).

- **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 | 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]
```

### Tip

If you don't already know the subnet IDs to use for the `--vpc-subnet-ids` parameter, see [Setting up network prerequisites \(p. 1437\)](#) 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 family, 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 \(p. 1458\)](#).

### Topics

- [Connecting to a proxy using native authentication \(p. 1446\)](#)
- [Connecting to a proxy using IAM authentication \(p. 1447\)](#)
- [Considerations for connecting to a proxy with PostgreSQL \(p. 1447\)](#)

## 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 \(p. 1634\)](#).

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 \(p. 1690\)](#).
- 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 \(p. 1455\)](#). 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 \(p. 1449\)](#)
- [Adding a new database user \(p. 1453\)](#)
- [Changing the password for a database user \(p. 1453\)](#)
- [Configuring connection settings \(p. 1453\)](#)

- [Avoiding pinning \(p. 1455\)](#)
- [Deleting an RDS Proxy \(p. 1457\)](#)

## 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. Currently, the setting isn't supported for PostgreSQL and the only choice is `EXCLUDE_VARIABLE_SETS`.

- **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": []
        },
        "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 \(p. 1438\)](#).
- 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 \(p. 1453\)](#)
- [MaxConnectionsPercent \(p. 1454\)](#)
- [MaxIdleConnectionsPercent \(p. 1454\)](#)
- [ConnectionBorrowTimeout \(p. 1454\)](#)

### 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 \(p. 1449\)](#). 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 \(p. 1449\)](#). 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 \(p. 1449\)](#). 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.

For MySQL engine family databases, this rule doesn't apply to all parameters that you can set. RDS Proxy tracks certain statements and variables. Thus RDS Proxy doesn't pin the session when you modify them. In this case, RDS Proxy only reuses the connection for other sessions that have the same values for those settings.

Following are the MySQL statements that RDS Proxy tracks:

- `DROP DATABASE`
- `DROP SCHEMA`
- `USE`

Following are the MySQL variables that RDS Proxy tracks:

- `AUTOCOMMIT`
- `AUTO_INCREMENT_INCREMENT`
- `CHARACTER SET` (or `CHAR SET`)
- `CHARACTER_SET_CLIENT`
- `CHARACTER_SET_DATABASE`
- `CHARACTER_SET_FILESYSTEM`
- `CHARACTER_SET_CONNECTION`
- `CHARACTER_SET_RESULTS`
- `CHARACTER_SET_SERVER`
- `COLLATION_CONNECTION`
- `COLLATION_DATABASE`

- `COLLATION_SERVER`
- `INTERACTIVE_TIMEOUT`
- `NAMES`
- `NET_WRITE_TIMEOUT`
- `QUERY_CACHE_TYPE`
- `SESSION_TRACK_SCHEMA`
- `SQL_MODE`
- `TIME_ZONE`
- `TRANSACTION_ISOLATION` (or `TX_ISOLATION`)
- `TRANSACTION_READ_ONLY` (or `TX_READ_ONLY`)
- `WAIT_TIMEOUT`

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.

- For a MySQL engine family database, 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 \(p. 1467\)](#).
- 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.

For metrics about how often pinning occurs for a proxy, see [Monitoring RDS Proxy metrics with Amazon CloudWatch \(p. 1467\)](#).

## Conditions that cause pinning for all engine families

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.

## Conditions that cause pinning for MySQL

For MySQL, the following interactions also cause pinning:

- Explicit table lock statements `LOCK TABLE`, `LOCK TABLES`, or `FLUSH TABLES WITH READ LOCK` cause the proxy to pin the session.
- Creating named locks by using `GET_LOCK` causes 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 session pinning filters property, see [Creating an RDS Proxy \(p. 1442\)](#) and [Modifying an RDS Proxy \(p. 1449\)](#).
- 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 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 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.

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.

## Conditions that cause pinning for PostgreSQL

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

## 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 \(p. 1458\)](#)
- [Using reader endpoints with Aurora clusters \(p. 1459\)](#)
- [Accessing Aurora and RDS databases across VPCs \(p. 1462\)](#)
- [Creating a proxy endpoint \(p. 1463\)](#)
- [Viewing proxy endpoints \(p. 1464\)](#)
- [Modifying a proxy endpoint \(p. 1465\)](#)
- [Deleting a proxy endpoint \(p. 1466\)](#)
- [Limitations for proxy endpoints \(p. 1467\)](#)

## 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 \(p. 35\)](#).

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 \(p. 1459\)](#).

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 [Limitations for proxy endpoints \(p. 1467\)](#).

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 \(p. 1467\)](#).

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 \(p. 1433\)](#).
- 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 \(p. 1458\)](#) 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 \(p. 93\)](#) 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 \(p. 1442\)](#).
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 \(p. 1463\)](#).
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 \(p. 1462\)](#).
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 \(p. 1459\)](#).
- `--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.

## Limitations 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.

## 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 \(p. 1458\)](#).

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
<code>AvailabilityPercentage</code>	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	<a href="#">Dimension set 4 (p. 1468)</a>
<code>ClientConnections</code>	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	<a href="#">Dimension set 1 (p. 1468), Dimension set 2 (p. 1468)</a>
<code>ClientConnectionsClosed</code>	The number of client connections closed. The most useful statistic for this metric is <code>Sum</code> .	1 minute and above	<a href="#">Dimension set 1 (p. 1468), Dimension set 2 (p. 1468)</a>
<code>ClientConnectionsNoTLS</code>	The current number of client connections without Transport Layer Security (TLS). This metric is reported	1 minute and above	<a href="#">Dimension set 1 (p. 1468), Dimension set 2 (p. 1468)</a>

Metric	Description	Valid period	CloudWatch dimension set
	every minute. The most useful statistic for this metric is Sum.		
ClientConnectionsReceived	The number of client connection requests received. The most useful statistic for this metric is Sum.	1 minute and above	Dimension set 1 (p. 1468), Dimension set 2 (p. 1468)
ClientConnectionsSetFailed	The number of client connection attempts that failed due to misconfigured authentication or TLS. The most useful statistic for this metric is Sum.	1 minute and above	Dimension set 1 (p. 1468), Dimension set 2 (p. 1468)
ClientConnectionsSetEstablished	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. 1468), Dimension set 2 (p. 1468)
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. 1468), Dimension set 2 (p. 1468)
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. 1468), Dimension set 3 (p. 1468), Dimension set 4 (p. 1468)
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. 1468), Dimension set 3 (p. 1468), Dimension set 4 (p. 1468)
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. 1468), Dimension set 3 (p. 1468), Dimension set 4 (p. 1468)

Metric	Description	Valid period	CloudWatch dimension set
DatabaseConnectionsTimeToFirstConnection	The time 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	<a href="#">Dimension set 1 (p. 1468)</a> , <a href="#">Dimension set 2 (p. 1468)</a>
DatabaseConnectionsCurrentBorrowed	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	<a href="#">Dimension set 1 (p. 1468)</a> , <a href="#">Dimension set 3 (p. 1468)</a> , <a href="#">Dimension set 4 (p. 1468)</a>
DatabaseConnectionsCurrentInTransaction	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	<a href="#">Dimension set 1 (p. 1468)</a> , <a href="#">Dimension set 3 (p. 1468)</a> , <a href="#">Dimension set 4 (p. 1468)</a>
DatabaseConnectionsCurrentPinned	The current number 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	<a href="#">Dimension set 1 (p. 1468)</a> , <a href="#">Dimension set 3 (p. 1468)</a> , <a href="#">Dimension set 4 (p. 1468)</a>
DatabaseConnectionsFailed	The number of database connection requests that failed. The most useful statistic for this metric is Sum.	1 minute and above	<a href="#">Dimension set 1 (p. 1468)</a> , <a href="#">Dimension set 3 (p. 1468)</a> , <a href="#">Dimension set 4 (p. 1468)</a>
DatabaseConnectionsSuccessful	The number of database connections successfully established with or without TLS. The most useful statistic for this metric is Sum.	1 minute and above	<a href="#">Dimension set 1 (p. 1468)</a> , <a href="#">Dimension set 3 (p. 1468)</a> , <a href="#">Dimension set 4 (p. 1468)</a>

Metric	Description	Valid period	CloudWatch dimension set
DatabaseConnectionsTLS	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. 1468), Dimension set 3 (p. 1468), Dimension set 4 (p. 1468)
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. 1468), Dimension set 3 (p. 1468), Dimension set 4 (p. 1468)
QueryDatabaseResponseTime	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. 1468), Dimension set 2 (p. 1468), Dimension set 3 (p. 1468), Dimension set 4 (p. 1468)
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. 1468), Dimension set 2 (p. 1468)
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. 1468), Dimension set 2 (p. 1468)
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. 1468), Dimension set 2 (p. 1468)

Metric	Description	Valid period	CloudWatch dimension set
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	<a href="#">Dimension set 1 (p. 1468)</a> , <a href="#">Dimension set 2 (p. 1468)</a>

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 \(p. 569\)](#).
- To learn how to configure Amazon Aurora to send events to EventBridge, see [Creating a rule that triggers on an Amazon Aurora event \(p. 587\)](#).

## 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).

Category	RDS event ID	Description
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-0215	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",
  "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 \(p. 1458\)](#).

### Topics

- [Verifying connectivity for a proxy \(p. 1476\)](#)
- [Common issues and solutions \(p. 1477\)](#)

## 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)	Some possible reasons include the following: <ul style="list-style-type: none"> <li>• 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.</li> </ul>
ERROR 1105 (HY000): Unknown error	An unknown error occurred.

Error	Causes or workarounds
ERROR 1231 (42000): Variable 'character_set_client' can't be set to the value of <code>value</code>	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.	You enabled the setting <b>Require Transport Layer Security</b> in the proxy but your connection included the parameter <code>ssl-mode=DISABLED</code> in the MySQL client. Do either of the following: <ul style="list-style-type: none"> <li>Disable the setting <b>Require Transport Layer Security</b> for the proxy.</li> <li>Connect to the database using the minimum setting of <code>ssl-mode=REQUIRED</code> in the MySQL client.</li> </ul>
ERROR 2026 (HY000): SSL connection error: Internal Server Error	The TLS handshake to the proxy failed. Some possible reasons include the following: <ul style="list-style-type: none"> <li>SSL is required but the server doesn't support it.</li> <li>An internal server error occurred.</li> <li>A bad handshake occurred.</li> </ul>
ERROR 9501 (HY000): 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"> <li>The proxy is unable to establish a database connection because the maximum connections have been reached</li> <li>The proxy is unable to establish a database connection because the database is unavailable.</li> </ul>

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 <a href="#">PostgreSQL SSL support</a> documentation.
This RDS Proxy requires TLS connections.	The user enabled the option <b>Require Transport Layer Security</b> 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"> <li>Disable the proxy's <b>Require Transport Layer Security</b> option.</li> <li>Connect to the database using the minimum setting of <code>sslmode=allow</code> in the PostgreSQL client.</li> </ul>
IAM authentication failed for user <code>user_name</code> . Check the IAM	This error might be due to the following reasons:	To fix this error, do the following:

Error	Cause	Solution
token for this user and try again.	<ul style="list-style-type: none"> <li>The client supplied the incorrect IAM user name.</li> <li>The client supplied an incorrect IAM authorization token for the user.</li> <li>The client is using an IAM policy that does not have the necessary permissions.</li> <li>The client supplied an expired IAM authorization token for the user.</li> </ul>	<ol style="list-style-type: none"> <li>Confirm that the provided IAM user exists.</li> <li>Confirm that the IAM authorization token belongs to the provided IAM user.</li> <li>Confirm that the IAM policy has adequate permissions for RDS.</li> <li>Check the validity of the IAM authorization token used.</li> </ol>
This RDS proxy has no credentials for the role <b>role_name</b> . 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.
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 psql 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 <b>option_name</b> .	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.

Error	Cause	Solution
The maximum number of client connections to the proxy exceeded <i>number_value</i> .	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 <i>number_value</i> .	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 <i>role_name</i> 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 <i>role_name</i> . 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.
Unknown error.	An unknown error occurred.	Reach out to AWS Support to investigate the issue.
Timed-out waiting to acquire database connection.	<p>The proxy timed-out waiting to acquire a database connection. Some possible reasons include the following:</p> <ul style="list-style-type: none"> <li>The proxy can't establish a database connection because the maximum connections have been reached.</li> <li>The proxy can't establish a database connection because the database is unavailable.</li> </ul>	<p>Possible solutions are the following:</p> <ul style="list-style-type: none"> <li>Check the target of the RDS DB instance or Aurora DB cluster status to see if it's unavailable.</li> <li>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.</li> </ul>
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](#).

# Using Aurora Serverless v2

Aurora Serverless v2 is an on-demand, autoscaling configuration for Amazon Aurora. Aurora Serverless v2 helps to automate the processes of monitoring the workload and adjusting the capacity for your databases. Capacity is adjusted automatically based on application demand. You're charged only for the resources that your DB clusters consume. Thus, Aurora Serverless v2 can help you to stay within budget and avoid paying for computer resources that you don't use.

This type of automation is especially valuable for multitenant databases, distributed databases, development and test systems, and other environments with highly variable and unpredictable workloads.

## Topics

- [Aurora Serverless v2 use cases \(p. 1482\)](#)
- [Advantages of Aurora Serverless v2 \(p. 1483\)](#)
- [How Aurora Serverless v2 works \(p. 1484\)](#)
- [Requirements for Aurora Serverless v2 \(p. 1490\)](#)
- [Getting started with Aurora Serverless v2 \(p. 1492\)](#)
- [Creating a cluster that uses Aurora Serverless v2 \(p. 1506\)](#)
- [Managing Aurora Serverless v2 \(p. 1509\)](#)
- [Performance and scaling for Aurora Serverless v2 \(p. 1526\)](#)

## Aurora Serverless v2 use cases

Many kinds of workloads can benefit from Aurora Serverless v2. Aurora Serverless v2 is especially useful for the following use cases:

- **Variable workloads** – You're running 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. Another is an e-commerce site with increased traffic when you offer sales or special promotions. With Aurora Serverless v2, your database automatically scales capacity to meet the needs of the application's peak load and scales back down when the surge of activity is over. With Aurora Serverless v2, you no longer need to provision for peak or average capacity. You can specify an upper capacity limit to handle the worst-case situation, and that capacity isn't used unless it's needed.

The granularity of scaling in Aurora Serverless v2 helps you to match capacity closely to your database's needs. For a provisioned cluster, scaling up requires adding a whole new DB instance. For an Aurora Serverless v1 cluster, scaling up requires doubling the number of Aurora capacity units (ACUs) for the cluster, such as from 16 to 32 or 32 to 64. In contrast, Aurora Serverless v2 can add half an ACU when only a little more capacity is needed. It can add 0.5, 1, 1.5, 2, or additional half-ACUs based on the additional capacity needed to handle an increase in workload. And it can remove 0.5, 1, 1.5, 2, or additional half-ACUs when the workload decreases and that capacity is no longer needed.

- **Multi-tenant applications** – With Aurora Serverless v2, you don't have to individually manage database capacity for each application in your fleet. Aurora Serverless v2 manages individual database capacity for you.

You can create a cluster for each tenant. That way, you can use features such as cloning, snapshot restore, and Aurora global databases to enhance high availability and disaster recovery as appropriate for each tenant.

Each tenant might have specific busy and idle periods depending on the time of day, time of year, promotional events, and so on. Each cluster can have a wide capacity range. That way, clusters with

low activity incur minimal DB instance charges. Any cluster can quickly scale up to handle periods of high activity.

- **New applications** – You're deploying a new application and you're unsure about the DB instance size you need. By using Aurora Serverless v2, you can set up a cluster with one or many DB instances and have the database autoscale to the capacity requirements of your application.
- **Development and testing** – With Aurora Serverless v2, you can create Aurora Serverless v2 DB instances with a low minimum capacity instead of using T DB instance classes. You can set the maximum capacity high enough that those DB instances can still run substantial workloads without running low on memory. When the database isn't in use, all the DB instances scale down to avoid unnecessary charges.

**Tip**

To make it convenient to use Aurora Serverless v2 in development and test environments, the AWS Management Console provides the **Easy create** shortcut when you create a new cluster. If you choose the **Dev/Test** option, Aurora creates a cluster with an Aurora Serverless v2 DB instance and a capacity range that's typical for a development and test system.

- **Mixed-use applications** – Suppose that you have an online transaction processing (OLTP) application, but you periodically experience spikes in query traffic. By specifying promotion tiers for the Aurora Serverless v2 DB instances in a cluster, you can configure your cluster so that the reader DB instances can scale independently of the writer DB instance to handle the additional load. When the usage spike subsides, the reader DB instances scale back down to match the capacity of the writer DB instance.
- **Capacity planning** – Suppose that you usually adjust your database capacity, or verify the optimal database capacity for your workload, by modifying the DB instance classes of all the DB instances in a cluster. With Aurora Serverless v2, you can avoid this administrative overhead. You can determine the appropriate minimum and maximum capacity by running the workload and checking how much the DB instances actually scale.

You can modify existing DB instances from provisioned to Aurora Serverless v2 or from Aurora Serverless v2 to provisioned. You don't need to create a new cluster or a new DB instance in such cases.

With an Aurora global database, you might not need as much capacity for the secondary clusters as in the primary cluster. You can use Aurora Serverless v2 DB instances in the secondary clusters. That way, the cluster capacity can scale up if a secondary region is promoted and takes over your application's workload.

## Starting to use Aurora Serverless v2 for existing provisioned workloads

Suppose that you already have an Aurora application running on a provisioned cluster. You can check how the application would work with Aurora Serverless v2 by adding one or more Aurora Serverless v2 DB instances to the existing cluster as reader DB instances. You can check how often the reader DB instances scale up and down. You can use the Aurora failover mechanism to promote an Aurora Serverless v2 DB instance to be the writer and check how it handles the read/write workload. That way, you can switch over with minimal downtime and without changing the endpoint that your client applications use. For details on the procedure to convert existing clusters to Aurora Serverless v2, see [Getting started with Aurora Serverless v2 \(p. 1492\)](#).

## Advantages of Aurora Serverless v2

Aurora Serverless v2 is intended for variable or "spiky" workloads. With such unpredictable workloads, you might have difficulty planning when to change your database capacity. You might also have trouble making capacity changes quickly enough using the familiar mechanisms such as adding DB instances or

changing DB instance classes. Aurora Serverless v2 provides the following advantages to help with such use cases:

- **Simpler capacity management than provisioned** – Aurora Serverless v2 reduces the effort for planning DB instance sizes and resizing DB instances as the workload changes. It also reduces the effort for maintaining consistent capacity for all the DB instances in a cluster.
- **Faster and easier scaling during periods of high activity** – Aurora Serverless v2 scales compute and memory capacity as needed, with no disruption to client transactions or your overall workload. The ability to use reader DB instances with Aurora Serverless v2 helps you to take advantage of horizontal scaling in addition to vertical scaling. The ability to use Aurora global databases means that you can spread your Aurora Serverless v2 read workload across multiple AWS Regions. This capability is more convenient than the scaling mechanisms for provisioned clusters. It's also faster and more granular than the scaling capabilities in Aurora Serverless v1.
- **Cost-effective during periods of low activity** – Aurora Serverless v2 helps you to avoid overprovisioning your DB instances. Aurora Serverless v2 adds resources in granular increments when DB instances scale up. You pay only for the database resources that you consume. Aurora Serverless v2 resource usage is measured on a per-second basis. That way, when a DB instance scales down, the reduced resource usage is registered right away.
- **Greater feature parity with provisioned** – You can use many Aurora features with Aurora Serverless v2 that aren't available for Aurora Serverless v1. For example, with Aurora Serverless v2 you can use reader DB instances, global databases, AWS Identity and Access Management (IAM) database authentication, and Performance Insights. You can also use many more configuration parameters than with Aurora Serverless v1.

In particular, with Aurora Serverless v2 you can take advantage of the following features from provisioned clusters:

- **Reader DB instances** – Aurora Serverless v2 can take advantage of reader DB instances to scale horizontally. When a cluster contains one or more reader DB instances, the cluster can fail over immediately in case of problems with the writer DB instance. This is a capability that isn't available with Aurora Serverless v1.
- **Multi-AZ clusters** – You can distribute the Aurora Serverless v2 DB instances of a cluster across multiple Availability Zones (AZs). Setting up a Multi-AZ cluster helps to ensure business continuity even in the rare case of issues that affect an entire AZ. This is a capability that isn't available with Aurora Serverless v1.
- **Global databases** – You can use Aurora Serverless v2 in combination with Aurora global databases to create additional read-only copies of your cluster in other AWS Regions for disaster recovery purposes.
- **RDS Proxy** – You can use Amazon RDS Proxy to allow your applications to pool and share database connections to improve their ability to scale.
- **Faster, more granular, less disruptive scaling than Aurora Serverless v1** – Aurora Serverless v2 can scale up and down faster. Scaling can change capacity by as little as 0.5 ACUs, instead of doubling or halving the number of ACUs. Scaling typically happens with no pause in processing at all. Scaling doesn't involve an event that you have to be aware of, as with Aurora Serverless v1. Scaling can happen while SQL statements are running and transactions are open, without the need to wait for a quiet point.

## How Aurora Serverless v2 works

Following, you can find an overview that describes how Aurora Serverless v2 works.

### Topics

- [Aurora Serverless v2 overview \(p. 1485\)](#)
- [Configurations for Aurora clusters \(p. 1486\)](#)

- [Aurora Serverless v2 capacity \(p. 1486\)](#)
- [Aurora Serverless v2 scaling \(p. 1487\)](#)
- [Aurora Serverless v2 and high availability \(p. 1488\)](#)
- [Aurora Serverless v2 and storage \(p. 1489\)](#)
- [Configuration parameters for Aurora clusters \(p. 1489\)](#)

## Aurora Serverless v2 overview

*Amazon Aurora Serverless v2* is suitable for the most demanding, highly variable workloads. 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, gaming, or sports websites with periodic promotional events, and databases that produce reports when needed. Others are development and testing environments, and new applications where usage might ramp up quickly. 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.

In contrast, *Aurora provisioned clusters* are suitable for steady workloads. With provisioned clusters, you choose a DB instance class that has a predefined amount of memory, CPU power, I/O bandwidth, and so on. If your workload changes, you manually modify the instance class of your writer and readers. The provisioned model works well when you can adjust capacity in advance of expected consumption patterns and it's acceptable to have brief outages while you change the instance class of the writer and readers in your cluster.

Aurora Serverless v2 is architected from the ground up to support serverless DB clusters that are instantly scalable. Aurora Serverless v2 is engineered to provide the same degree of security and isolation as with provisioned writers and readers. These aspects are crucial in multitenant serverless cloud environments. The dynamic scaling mechanism has very little overhead so that it can respond quickly to changes in the database workload. It's also powerful enough to meet dramatic increases in processing demand.

By using Aurora Serverless v2, you can create an Aurora DB cluster without being locked into a specific database capacity for each writer and reader. You specify the minimum and maximum capacity range. Aurora scales each Aurora Serverless v2 writer or reader in the cluster within that capacity range. By using a Multi-AZ cluster where each writer or reader can scale dynamically, you can take advantage of dynamic scaling and high availability.

Aurora Serverless v2 scales the database resources automatically based on your minimum and maximum capacity specifications. Scaling is fast because most scaling events operations keep the writer or reader on the same host. In the rare cases that an Aurora Serverless v2 writer or reader is moved from one host to another, Aurora Serverless v2 manages the connections automatically. You don't need to change your database client application code or your database connection strings.

With Aurora Serverless v2, as with provisioned clusters, storage capacity and compute capacity are separate. When we refer to Aurora Serverless v2 capacity and scaling, it's always compute capacity that's increasing or decreasing. Thus, your cluster can contain many terabytes of data even when the CPU and memory capacity scale down to low levels.

Instead of provisioning and managing database servers, you specify database capacity. For details about Aurora Serverless v2 capacity, see [Aurora Serverless v2 capacity \(p. 1486\)](#). The actual capacity of each Aurora Serverless v2 writer or reader varies over time, depending on your workload. For details about that mechanism, see [Aurora Serverless v2 scaling \(p. 1487\)](#).

### Important

With Aurora Serverless v1, your cluster has a single measure of compute capacity that can scale between the minimum and maximum capacity values. With Aurora Serverless v2, your cluster can contain readers in addition to the writer. Each Aurora Serverless v2 writer and reader can

scale between the minimum and maximum capacity values. Thus, the total capacity of your Aurora Serverless v2 cluster depends on both the capacity range that you define for your DB cluster and the number of writers and readers in the cluster. At any specific time, you are only charged for the Aurora Serverless v2 capacity that is being actively used in your Aurora DB cluster.

## Configurations for Aurora clusters

For each of your Aurora DB clusters, you can choose any combination of Aurora Serverless v2 capacity, provisioned capacity, or both.

You can set up a cluster that contains both Aurora Serverless v2 and provisioned capacity, called a *mixed-configuration cluster*. For example, suppose that you need more read/write capacity than is available for an Aurora Serverless v2 writer. In this case, you can set up the cluster with a very large provisioned writer. In that case, you can still use Aurora Serverless v2 for the readers. Or suppose that the write workload for your cluster varies but the read workload is steady. In this case, you can set up your cluster with an Aurora Serverless v2 writer and one or more provisioned readers.

You can also set up a DB cluster where all the capacity is managed by Aurora Serverless v2. To do this, you can create a new cluster and use Aurora Serverless v2 from the start. Or you can replace all the provisioned capacity in an existing cluster with Aurora Serverless v2. For example, some of the upgrade paths from older engine versions require starting with a provisioned writer and replacing it with an Aurora Serverless v2 writer. For the procedures to create a new DB cluster with Aurora Serverless v2 or to switch an existing DB cluster to Aurora Serverless v2, see [Creating a cluster that uses Aurora Serverless v2 \(p. 1506\)](#) and [Switching from a provisioned cluster to Aurora Serverless v2 \(p. 1495\)](#).

If you don't use Aurora Serverless v2 at all in a DB cluster, all the writers and readers in the DB cluster are *provisioned*. This is the oldest and most common kind of DB cluster that most users are familiar with. In fact, before Aurora Serverless, there wasn't a special name for this kind of Aurora DB cluster. Provisioned capacity is constant. The charges are relatively easy to forecast. However, you have to predict in advance how much capacity you need. In some cases, your predictions might be inaccurate or your capacity needs might change. In these cases, your DB cluster can become underprovisioned (slower than you want) or overprovisioned (more expensive than you want).

## Aurora Serverless v2 capacity

The unit of measure for Aurora Serverless v2 is the *Aurora capacity unit (ACU)*. Aurora Serverless v2 capacity isn't tied to the DB instance classes that you use for provisioned clusters.

Each ACU is a combination of approximately 2 gibibytes (GiB) of memory, corresponding CPU, and networking. You specify the database capacity range using this unit of measure. The `ServerlessDatabaseCapacity` and `ACUUtilization` metrics help you to determine how much capacity your database is actually using and where that capacity falls within the specified range.

At any moment in time, each Aurora Serverless v2 DB writer or reader has a *capacity*. The capacity is represented as a floating-point number representing ACUs. The capacity increases or decreases whenever the writer or reader scales. This value is measured every second. For each DB cluster where you intend to use Aurora Serverless v2, you define a *capacity range*: the minimum and maximum capacity values that each Aurora Serverless v2 writer or reader can scale between. The capacity range is the same for each Aurora Serverless v2 writer or reader in a DB cluster. Each Aurora Serverless v2 writer or reader has its own capacity, falling somewhere in that range.

The largest Aurora Serverless v2 capacity that you can define is 128 ACUs. For all the considerations when choosing the maximum capacity value, see [Choosing the maximum Aurora Serverless v2 capacity setting for a cluster \(p. 1528\)](#).

The smallest Aurora Serverless v2 capacity that you can define is 0.5 ACUs. You can specify a higher number if it's less than or equal to the maximum capacity value. Setting the minimum capacity to a small

number lets lightly loaded DB clusters consume minimal compute resources. At the same time, they stay ready to accept connections immediately and scale up when they become busy.

We recommend setting the minimum to a value that allows each DB writer or reader to hold the working set of the application in the buffer pool. That way, the contents of the buffer pool aren't discarded during idle periods. For all the considerations when choosing the minimum capacity value, see [Choosing the minimum Aurora Serverless v2 capacity setting for a cluster \(p. 1527\)](#).

Depending on how you configure the readers in a Multi-AZ DB cluster, their capacities can be tied to the capacity of the writer or independently. For details about how to do that, see [Aurora Serverless v2 scaling \(p. 1487\)](#).

Monitoring Aurora Serverless v2 involves measuring the capacity values for the writer and readers in your DB cluster over time. If your database doesn't scale down to the minimum capacity, you can take actions such as adjusting the minimum and optimizing your database application. If your database consistently reaches its maximum capacity, you can take actions such as increasing the maximum. You can also optimize your database application and spread the query load across more readers.

The charges for Aurora Serverless v2 capacity are measured in terms of ACU-hours. For information about how Aurora Serverless v2 charges are calculated, see the [Aurora pricing page](#).

Suppose that the total number of writers and readers in your cluster is  $N$ . In that case, the cluster consumes approximately  $n \times \text{minimum ACUs}$  when you aren't running any database operations. Aurora itself might run monitoring or maintenance operations that cause some small amount of load. That cluster consumes no more than  $n \times \text{maximum ACUs}$  when the database is running at full capacity.

For more details about choosing appropriate minimum and maximum ACU values, see [Choosing the Aurora Serverless v2 capacity range for an Aurora cluster \(p. 1527\)](#). The minimum and maximum ACU values that you specify also affect the way some of the Aurora configuration parameters work for Aurora Serverless v2. For details about the interaction between the capacity range and configuration parameters, see [Working with parameter groups for Aurora Serverless v2 \(p. 1535\)](#).

## Aurora Serverless v2 scaling

For each Aurora Serverless v2 writer or reader, Aurora continuously tracks utilization of resources such as CPU, memory, and network. These measurements collectively are called the *load*. The load includes the database operations performed by your application. It also includes background processing for the database server and Aurora administrative tasks. When capacity is constrained by any of these, Aurora Serverless v2 scales up. Aurora Serverless v2 also scales up when it detects performance issues that it can resolve by doing so. You can monitor resource utilization and how it affects Aurora Serverless v2 scaling by using the procedures in [Important Amazon CloudWatch metrics for Aurora Serverless v2 \(p. 1539\)](#) and [Monitoring Aurora Serverless v2 performance with Performance Insights \(p. 1542\)](#).

The load can vary across the writer and readers in your DB cluster. The writer handles all data definition language (DDL) statements, such as `CREATE TABLE`, `ALTER TABLE`, and `DROP TABLE`. The writer also handles all data manipulation language (DML) statements, such as `INSERT` and `UPDATE`. Readers can process read-only statements, such as `SELECT` queries.

*Scaling* is the operation that increases or decreases Aurora Serverless v2 capacity for your database. With Aurora Serverless v2, each writer and reader has its own current capacity value, measured in ACUs. Aurora Serverless v2 scales a writer or reader up to a higher capacity when its current capacity is too low to handle the load. It scales the writer or reader down to a lower capacity when its current capacity is higher than needed.

Unlike Aurora Serverless v1, which scales by doubling the capacity each time the DB cluster reaches a threshold, Aurora Serverless v2 can increase capacity incrementally. When your workload demand begins to reach the current database capacity of a writer or reader, Aurora Serverless v2 increases the number of ACUs for that writer or reader. Aurora Serverless v2 scales capacity in the increments required to provide

the best performance for the resources consumed. Scaling happens in increments as small as 0.5 ACUs. The larger the current capacity, the larger the scaling increment and thus the faster scaling can happen.

Because Aurora Serverless v2 scaling is so frequent, granular, and nondisruptive, it doesn't cause discrete events in the AWS Management Console the way that Aurora Serverless v1 does. Instead, you can measure the Amazon CloudWatch metrics such as `ServerlessDatabaseCapacity` and `ACUUtilization` and track their minimum, maximum, and average values over time. To learn more about Aurora metrics, see [Monitoring metrics in an Amazon Aurora cluster \(p. 427\)](#). For tips about monitoring Aurora Serverless v2, see [Important Amazon CloudWatch metrics for Aurora Serverless v2 \(p. 1539\)](#).

You can choose to make a reader scale at the same time as the associated writer, or independently from the writer. You do so by specifying the promotion tier for that reader.

- Readers in promotion tiers 0 and 1 scale at the same time as the writer. That scaling behavior makes readers in priority tiers 0 and 1 ideal for availability. That's because they are always sized to the right capacity to take over the workload from the writer in case of failover.
- Readers in promotion tiers 2–15 scale independently from the writer. Each reader remains within the minimum and maximum ACU values that you specified for your cluster. When a reader scales independently of the associated writer DB, it can become idle and scale down while the writer continues to process a high volume of transactions. It's still available as a failover target, if no other readers are available in lower promotion tiers. However, if it's promoted to be the writer, it might need to scale up to handle the full workload of the writer.

For details about promotion tiers, see [Choosing the promotion tier for an Aurora Serverless v2 reader \(p. 1521\)](#).

The notions of scaling points and associated timeout periods from Aurora Serverless v1 don't apply in Aurora Serverless v2. Aurora Serverless v2 scaling can happen while database connections are open, while SQL transactions are in process, while tables are locked, and while temporary tables are in use. Aurora Serverless v2 doesn't wait for a quiet point to begin scaling. Scaling doesn't disrupt any database operations that are underway.

If your workload requires more read capacity than is available with a single writer and a single reader, you can add multiple Aurora Serverless v2 readers to the cluster. Each Aurora Serverless v2 reader can scale within the range of minimum and maximum capacity values that you specified for your DB cluster. You can use the cluster's reader endpoint to direct read-only sessions to the readers and reduce the load on the writer.

Whether Aurora Serverless v2 performs scaling, and how fast scaling occurs once it starts, also depends on the minimum and maximum ACU settings for the cluster. In addition, it depends on whether a reader is configured to scale along with the writer or independently from it. For details about the factors that affect Aurora Serverless v2 scaling, see [Performance and scaling for Aurora Serverless v2 \(p. 1526\)](#).

**Note**

Currently, Aurora Serverless v2 writers and readers don't scale all the way down to zero ACUs. Idle Aurora Serverless v2 writers and readers can scale down to the minimum ACU value that you specified for the cluster.

That behavior is different than Aurora Serverless v1, which can pause after a period of idleness, but then takes some time to resume when you open a new connection. When your DB cluster with Aurora Serverless v2 capacity isn't needed for some time, you can stop and start clusters as with provisioned DB clusters. For details about stopping and starting clusters, see [Stopping and starting an Amazon Aurora DB cluster \(p. 244\)](#).

## Aurora Serverless v2 and high availability

The way to establish high availability for an Aurora DB cluster is to make it a Multi-AZ DB cluster. A *Multi-AZ Aurora DB cluster* has compute capacity available at all times in more than one Availability Zone (AZ).

That configuration keeps your database up and running even in case of a significant outage. Aurora performs an automatic failover in case of an issue that affects the writer or even the entire AZ. With Aurora Serverless v2, you can choose for the standby compute capacity to scale up and down along with the capacity of the writer. That way, the compute capacity in the second AZ is ready to take over the current workload at any time. At the same time, the compute capacity in all AZs can scale down when the database is idle. For details about how Aurora works with AWS Regions and Availability Zones, see [High availability for Aurora DB instances \(p. 71\)](#).

The Aurora Serverless v2 Multi-AZ capability uses *readers* in addition to the writer. Support for readers is new for Aurora Serverless v2 compared to Aurora Serverless v1. You can add up to 15 Aurora Serverless v2 readers spread across 3 AZs to an Aurora DB cluster.

For business-critical applications that must remain available even in case of an issue that affects your entire cluster or the whole AWS Region, you can set up an Aurora global database. You can use Aurora Serverless v2 capacity in the secondary clusters so they're ready to take over during disaster recovery. They can also scale down when the database isn't busy. For details about Aurora global databases, see [Using Amazon Aurora global databases \(p. 151\)](#).

Aurora Serverless v2 works like provisioned for failover and other high availability features. For more information, see [High availability for Amazon Aurora \(p. 71\)](#).

Suppose that you want to ensure maximum availability for your Aurora Serverless v2 cluster. You can create a reader in addition to the writer. If you assign the reader to promotion tier 0 or 1, whatever scaling happens for the writer also happens for the reader. That way, a reader with identical capacity is always ready to take over for the writer in case of a failover.

Suppose that you want to run quarterly reports for your business at the same time as your cluster continues to process transactions. If you add an Aurora Serverless v2 reader to the cluster and assign it to a promotion tier from 2 through 15, you can connect directly to that reader to run the reports. Depending on how memory-intensive and CPU-intensive the reporting queries are, that reader can scale up to accommodate the workload. It can then scale down again when the reports are finished.

## Aurora Serverless v2 and storage

The storage for each Aurora DB cluster consists of six copies of all your data, spread across three AZs. This built-in data replication applies regardless of whether your DB cluster includes any readers in addition to the writer. That way, your data is safe, even from issues that affect the compute capacity of the cluster.

Aurora Serverless v2 storage has the same reliability and durability characteristics as described in [Amazon Aurora storage and reliability \(p. 67\)](#). That's because the storage for Aurora DB clusters works the same whether the compute capacity uses Aurora Serverless v2 or provisioned.

## Configuration parameters for Aurora clusters

You can adjust all the same cluster and database configuration parameters for clusters with Aurora Serverless v2 capacity as for provisioned DB clusters. However, some capacity-related parameters are handled differently for Aurora Serverless v2. In a mixed-configuration cluster, the parameter values that you specify for those capacity-related parameters still apply to any provisioned writers and readers.

Almost all of the parameters work the same way for Aurora Serverless v2 writers and readers as for provisioned ones. The exceptions are some parameters that Aurora automatically adjusts during scaling, and some parameters that Aurora keeps at fixed values that depend on the maximum capacity setting.

For example, the amount of memory reserved for the buffer cache increases as a writer or reader scales up, and decreases as it scales down. That way, memory can be released when your database isn't busy. Conversely, Aurora automatically sets the maximum number of connections to a value that's appropriate

based on the maximum capacity setting. That way, active connections aren't dropped if the load drops and Aurora Serverless v2 scales down. For information about how Aurora Serverless v2 handles specific parameters, see [Working with parameter groups for Aurora Serverless v2 \(p. 1535\)](#).

## Requirements for Aurora Serverless v2

When you create a cluster where you intend to use Aurora Serverless v2 DB instances, pay attention to the following requirements.

### Topics

- [Aurora Serverless v2 is available in certain AWS Regions \(p. 1490\)](#)
- [Aurora Serverless v2 requires minimum engine versions \(p. 1490\)](#)
- [Clusters that use Aurora Serverless v2 must have a capacity range specified \(p. 1491\)](#)
- [Some provisioned features aren't supported in Aurora Serverless v2 \(p. 1491\)](#)
- [Some Aurora Serverless v2 aspects are different from Aurora Serverless v1 \(p. 1492\)](#)

## Aurora Serverless v2 is available in certain AWS Regions

For the AWS Regions where Aurora Serverless v2 DB instances are currently available, see [Aurora Serverless v2 \(p. 31\)](#).

## Aurora Serverless v2 requires minimum engine versions

Aurora clusters that use Aurora Serverless v2 DB instances must be running one of the following DB engine versions:

- MySQL-compatible DB instances for Aurora Serverless v2 require Aurora MySQL 3.02.0 or higher. This Aurora MySQL version is compatible with MySQL 8.0.
- PostgreSQL-compatible DB instances for Aurora Serverless v2 require Aurora PostgreSQL 13.6 or higher.

For the complete list of supported versions and AWS Regions, see [Aurora Serverless v2 \(p. 31\)](#).

The following example shows the AWS CLI commands to confirm the exact DB engine values you can use with Aurora Serverless v2 for a specific AWS Region. The `--db-instance-class` parameter for Aurora Serverless v2 is always `db.serverless`. The `--engine` parameter can be `aurora-mysql` or `aurora-postgresql`. Substitute the appropriate `--region` and `--engine` values to confirm the `--engine-version` values that you can use. If the command doesn't produce any output, Aurora Serverless v2 isn't available for that combination of AWS Region and DB engine.

```
aws rds describe-orderable-db-instance-options --engine aurora-mysql --db-instance-class db.serverless \
--region my_region --query 'OrderableDBInstanceOptions[].[EngineVersion]' --output text

aws rds describe-orderable-db-instance-options --engine aurora-postgresql --db-instance-class db.serverless \
--region my_region --query 'OrderableDBInstanceOptions[].[EngineVersion]' --output text
```

## Clusters that use Aurora Serverless v2 must have a capacity range specified

An Aurora cluster must have a `ScalingConfigurationInfo` attribute before you can add any DB instances that use the `db.serverless` DB instance class. This attribute specifies the capacity range. Aurora Serverless v2 capacity ranges from a minimum of 0.5 Aurora capacity units (ACU) through 128 ACUs, in increments of 0.5 ACU. Each ACU provides the equivalent of approximately 2 gibibytes (GiB) of RAM and associated CPU and networking. For details about how Aurora Serverless v2 uses the capacity range settings, see [How Aurora Serverless v2 works \(p. 1484\)](#).

You can specify the minimum and maximum ACU values in the AWS Management Console when you create a cluster and associated Aurora Serverless v2 DB instance. You can also specify the `--serverless-v2-scaling-configuration` option in the AWS CLI. Or you can specify the `ServerlessV2ScalingConfiguration` parameter with the Amazon RDS API. You can specify this attribute when you create a cluster or modify an existing cluster. For the procedures to set the capacity range, see [Setting the Aurora Serverless v2 capacity range for a cluster \(p. 1510\)](#). For a detailed discussion of how to pick minimum and maximum capacity values and how those settings affect some database parameters, see [Choosing the Aurora Serverless v2 capacity range for an Aurora cluster \(p. 1527\)](#).

## Some provisioned features aren't supported in Aurora Serverless v2

The following features from Aurora provisioned DB instances currently aren't available for Amazon Aurora Serverless v2:

- Database activity streams (DAS).
- Backtracking for Aurora MySQL. This feature currently isn't available for Aurora MySQL version 3. For details of Aurora MySQL version 3 feature support, see [Aurora MySQL version 3 compatible with MySQL 8.0 \(p. 653\)](#).
- Cluster cache management for Aurora PostgreSQL. The `apg_ccm_enabled` configuration parameter doesn't apply to Aurora Serverless v2 DB instances.
- In the AWS Billing and Cost Management console, filtering by custom tags added to Aurora Serverless v2 shows zero usage. Use other filtering options, such as AWS Region, engine, and usageType, to get accurate billed values.

Some Aurora features work with Aurora Serverless v2, but might cause issues if your capacity range is lower than needed for the memory requirements for those features with your specific workload. In that case, your database might not perform as well as usual, or might encounter out-of-memory errors. For recommendations about setting the appropriate capacity range, see [Choosing the Aurora Serverless v2 capacity range for an Aurora cluster \(p. 1527\)](#). For troubleshooting information if your database encounters out-of-memory errors due to a misconfigured capacity range, see [Avoiding out-of-memory errors \(p. 1538\)](#).

Aurora Auto Scaling isn't supported. This type of scaling adds new readers to handle additional read-intensive workload, based on CPU utilization. However, scaling based on CPU utilization isn't meaningful for Aurora Serverless v2. As an alternative, you can create Aurora Serverless v2 reader DB instances in advance and leave them scaled down to low capacity. That's a faster and less disruptive way to scale a cluster's read capacity than adding new DB instances dynamically.

## Some Aurora Serverless v2 aspects are different from Aurora Serverless v1

If you are an Aurora Serverless v1 user and this is your first time using Aurora Serverless v2, consult [differences between Aurora Serverless v2 and Aurora Serverless v1 requirements \(p. 1500\)](#) to understand how requirements are different between Aurora Serverless v1 and Aurora Serverless v2.

## Getting started with Aurora Serverless v2

To convert an existing database to use Aurora Serverless v2, you can do the following:

- Upgrade from a provisioned Aurora cluster.
- Upgrade from an Aurora Serverless v1 cluster.
- Perform a dump and restore from an Aurora Serverless v2 (preview) cluster.
- Migrate from an on-premises database to an Aurora Serverless v2 cluster.

When your upgraded cluster is running the appropriate engine version as listed in [Requirements for Aurora Serverless v2 \(p. 1490\)](#), you can begin adding Aurora Serverless v2 DB instances to it. The first DB instance that you add to the upgraded cluster must be a provisioned DB instance. Then you can switch over the processing for the write workload, the read workload, or both to the Aurora Serverless v2 DB instances.

### Contents

- [Upgrading or switching existing clusters to use Aurora Serverless v2 \(p. 1492\)](#)
  - Upgrade paths for MySQL-compatible clusters to use Aurora Serverless v2 (p. 1493)
  - Upgrade paths for PostgreSQL-compatible clusters to use Aurora Serverless v2 (p. 1494)
- [Switching from a provisioned cluster to Aurora Serverless v2 \(p. 1495\)](#)
- [Moving from Aurora Serverless v1 to Aurora Serverless v2 \(p. 1499\)](#)
  - Comparison of Aurora Serverless v2 and Aurora Serverless v1 (p. 1500)
    - Comparison of Aurora Serverless v2 and Aurora Serverless v1 requirements (p. 1500)
    - Comparison of Aurora Serverless v2 and Aurora Serverless v1 scaling and availability (p. 1501)
    - Comparison of Aurora Serverless v2 and Aurora Serverless v1 feature support (p. 1503)
  - Adapting Aurora Serverless v1 use cases to Aurora Serverless v2 (p. 1504)
  - Upgrading from an Aurora Serverless v1 cluster to Aurora Serverless v2 (p. 1505)
- [Upgrading from Aurora Serverless v2 \(preview\) to Aurora Serverless v2 \(p. 1506\)](#)
- [Migrating from an on-premises database to Aurora Serverless v2 \(p. 1506\)](#)

## Upgrading or switching existing clusters to use Aurora Serverless v2

If your provisioned cluster has an engine version that supports Aurora Serverless v2, switching to Aurora Serverless v2 doesn't require an upgrade. In that case, you can add Aurora Serverless v2 DB instances to your original cluster. You can switch the cluster to use all Aurora Serverless v2 DB instances. You can also use a combination of Aurora Serverless v2 and provisioned DB instances in the same DB cluster. For the

Aurora engine versions that support Aurora Serverless v2, see [Aurora Serverless v2 requires minimum engine versions \(p. 1490\)](#).

If you're running a lower engine version that doesn't support Aurora Serverless v2, you take these general steps:

1. Upgrade the cluster.
2. Create a provisioned writer DB instance for the upgraded cluster.
3. Modify the cluster to use Aurora Serverless v2 DB instances.

Such an upgrade might involve one or more snapshot restore operations. When doing an upgrade across multiple major engine versions, you perform an intermediate upgrade for every major version between the original and final clusters.

**Important**

When you perform a major version upgrade to an Aurora Serverless v2-compatible version by using snapshot restore or cloning, the first DB instance that you add to the new cluster must be a provisioned DB instance. This addition starts the final stage of the upgrade process.

Until that final stage happens, the cluster doesn't have the infrastructure that's required for Aurora Serverless v2 support. Thus, these upgraded clusters always start with a provisioned writer DB instance. Then you can convert or fail over the provisioned DB instance to an Aurora Serverless v2 one.

Upgrading from Aurora Serverless v1 to Aurora Serverless v2 involves creating a provisioned cluster as an intermediate step. Then you perform the same upgrade steps as when you start with a provisioned cluster.

Switching from Aurora Serverless v2 (preview) to Aurora Serverless v2 involves a logical dump and restore.

## Upgrade paths for MySQL-compatible clusters to use Aurora Serverless v2

If your original cluster is running Aurora MySQL, choose the appropriate procedure depending on the engine version and engine mode of your cluster.

If your original Aurora MySQL cluster is this	Do this to switch to Aurora Serverless v2
Provisioned cluster running Aurora MySQL version 3, compatible with MySQL 8.0	This is the final stage for all conversions from existing Aurora MySQL clusters.  If necessary, perform a minor version upgrade to version 3.02.0 or higher. Use a provisioned DB instance for the writer DB instance. Add one Aurora Serverless v2 reader DB instance. Perform a failover to make that the writer DB instance. Optional: Convert other provisioned DB instances in the cluster to Aurora Serverless v2, or add new Aurora Serverless v2 DB instances and remove the provisioned DB instances.  For the full procedure and examples, see <a href="#">Switching from a provisioned cluster to Aurora Serverless v2 (p. 1495)</a> .
Provisioned cluster running Aurora MySQL version 2, compatible with MySQL 5.7	Perform a major version upgrade to Aurora MySQL version 3.02.0 or higher. Then follow

If your original Aurora MySQL cluster is this	Do this to switch to Aurora Serverless v2
	the procedure for Aurora MySQL version 3 to switch the cluster to use Aurora Serverless v2 DB instances.
Provisioned cluster running Aurora MySQL version 1, compatible with MySQL 5.6	Perform an intermediate major version upgrade to Aurora MySQL version 2. Perform another major version upgrade to Aurora MySQL version 3.02.0 or higher. Then follow the procedure for Aurora MySQL version 3 to switch the cluster to use Aurora Serverless v2 DB instances.
Aurora Serverless v1 cluster running Aurora MySQL version 2, compatible with MySQL 5.7	To help plan your conversion from Aurora Serverless v1, consult <a href="#">Moving from Aurora Serverless v1 to Aurora Serverless v2 (p. 1499)</a> first.  Then follow the procedure in <a href="#">Upgrading from an Aurora Serverless v1 cluster to Aurora Serverless v2 (p. 1505)</a> .
Aurora Serverless v1 cluster running Aurora MySQL version 1, compatible with MySQL 5.6	To help plan your conversion from Aurora Serverless v1, consult <a href="#">Moving from Aurora Serverless v1 to Aurora Serverless v2 (p. 1499)</a> first.  Then follow the procedure in <a href="#">Upgrading from an Aurora Serverless v1 cluster to Aurora Serverless v2 (p. 1505)</a> .
Aurora Serverless v2 (preview) cluster	Perform a logical dump and restore to a provisioned cluster that's running the same engine version as the preview cluster. Then follow the procedure to upgrade from that version.

## Upgrade paths for PostgreSQL-compatible clusters to use Aurora Serverless v2

If your original cluster is running Aurora PostgreSQL, choose the appropriate procedure depending on the engine version and engine mode of your cluster.

If your original Aurora PostgreSQL cluster is this	Do this to switch to Aurora Serverless v2
Provisioned cluster running Aurora PostgreSQL version 13	This is the final stage for all conversions from existing Aurora PostgreSQL clusters.  If necessary, perform a minor version upgrade to version 13.6 or higher. Add one provisioned DB instance for the writer DB instance. Add one Aurora Serverless v2 reader DB instance. Perform a failover to make that Aurora Serverless v2 instance the writer DB instance. Optional: Convert other provisioned DB instances in the cluster to Aurora Serverless v2, or add new

If your original Aurora PostgreSQL cluster is this	Do this to switch to Aurora Serverless v2
	Aurora Serverless v2 DB instances and remove the provisioned DB instances.  For the full procedure and examples, see <a href="#">Switching from a provisioned cluster to Aurora Serverless v2 (p. 1495)</a> .
Provisioned cluster running Aurora PostgreSQL version 12	Perform a major version upgrade to Aurora PostgreSQL version 13.6 or higher. Then follow the procedure for Aurora PostgreSQL version 13 to switch the cluster to use Aurora Serverless v2 DB instances.
Provisioned cluster running Aurora PostgreSQL version 11	Perform an intermediate major version upgrade to each successive Aurora PostgreSQL major version until you reach Aurora PostgreSQL version 13.6 or higher. Depending on the starting version, you might be able to upgrade directly to the final Aurora PostgreSQL version. For details about which Aurora PostgreSQL versions can upgrade directly to which other Aurora PostgreSQL major versions, see <a href="#">How to perform a major version upgrade (p. 1417)</a> . Then follow the procedure for Aurora MySQL version 13 to switch the cluster to use Aurora Serverless v2 DB instances.
Provisioned cluster running Aurora PostgreSQL version 10	Perform an intermediate major version upgrade to each successive Aurora PostgreSQL major version until you reach Aurora PostgreSQL version 13.6 or higher. Depending on the starting version, you might be able to upgrade directly to the final Aurora PostgreSQL version. For details about which Aurora PostgreSQL versions can upgrade directly to which other Aurora PostgreSQL major versions, see <a href="#">How to perform a major version upgrade (p. 1417)</a> . Then follow the procedure for Aurora MySQL version 13 to switch the cluster to use Aurora Serverless v2 DB instances.
Aurora Serverless v1 cluster running Aurora PostgreSQL version 10 or 11	To help plan your conversion from Aurora Serverless v1, consult <a href="#">Moving from Aurora Serverless v1 to Aurora Serverless v2 (p. 1499)</a> first.  Then follow the procedure in <a href="#">Upgrading from an Aurora Serverless v1 cluster to Aurora Serverless v2 (p. 1505)</a> .

## Switching from a provisioned cluster to Aurora Serverless v2

To switch a provisioned cluster to use Aurora Serverless v2, follow these steps:

1. Check if the provisioned cluster needs to be upgraded to be used with Aurora Serverless v2 DB instances. For the Aurora versions that are compatible with Aurora Serverless v2, see [Requirements for Aurora Serverless v2 \(p. 1490\)](#).

If the provisioned cluster is running an engine version that isn't available for Aurora Serverless v2, upgrade the engine version of the cluster:

- If you have a MySQL 5.6-compatible or MySQL 5.7-compatible provisioned cluster, follow the upgrade instructions for Aurora MySQL version 3. Use the procedures in [Upgrading to Aurora MySQL version 3 \(p. 666\)](#).
- If you have a PostgreSQL-compatible provisioned cluster running PostgreSQL version 10 through 12, follow the upgrade instructions for Aurora PostgreSQL version 13. Use the procedures in [How to perform a major version upgrade \(p. 1417\)](#).

2. Configure any other cluster properties to match the Aurora Serverless v2 requirements from [Requirements for Aurora Serverless v2 \(p. 1490\)](#).
3. Configure the scaling configuration for the cluster. Follow the procedure in [Setting the Aurora Serverless v2 capacity range for a cluster \(p. 1510\)](#).
4. Add one or more Aurora Serverless v2 DB instances to the cluster. Follow the general procedure in [Adding Aurora Replicas to a DB cluster \(p. 270\)](#). For each new DB instance, specify the special DB instance class name **Serverless** in the AWS Management Console, or `db.serverless` in the AWS CLI or Amazon RDS API.

In some cases, you might already have one or more provisioned reader DB instances in the cluster. If so, you can convert one of the readers to an Aurora Serverless v2 DB instance instead of creating a new DB instance. To do so, follow the procedure in [Converting a provisioned writer or reader to Aurora Serverless v2 \(p. 1518\)](#).

5. Perform a failover operation to make one of the Aurora Serverless v2 DB instances the writer DB instance for the cluster.
6. (Optional) Convert any provisioned DB instances to Aurora Serverless v2, or remove them from the cluster. Follow the general procedure in [Converting a provisioned writer or reader to Aurora Serverless v2 \(p. 1518\)](#) or [Deleting a DB instance from an Aurora DB cluster \(p. 350\)](#).

**Tip**

Removing the provisioned DB instances isn't mandatory. You can set up a cluster containing both Aurora Serverless v2 and provisioned DB instances. However, until you are familiar with the performance and scaling characteristics of Aurora Serverless v2 DB instances, we recommend that you configure your clusters with DB instances all of the same type.

The following AWS CLI example shows the switchover process using a provisioned cluster that's running Aurora MySQL version 3.02.0. The cluster is named `mysql-80`. The cluster starts with two provisioned DB instances named `provisioned-instance-1` and `provisioned-instance-2`, a writer and a reader. They both use the `db.r6g.large` DB instance class.

```
$ aws rds describe-db-clusters --db-cluster-identifier mysql-80 \
  --query '*[].[DBClusterIdentifier,DBClusterMembers[*].
[DBInstanceIdentifier,IsClusterWriter]]' --output text
mysql-80
provisioned-instance-2      False
provisioned-instance-1      True

$ aws rds describe-db-instances --db-instance-identifier provisioned-instance-1 \
  --output text --query '*[].[DBInstanceIdentifier,DBInstanceClass]'
provisioned-instance-1      db.r6g.large

$ aws rds describe-db-instances --db-instance-identifier provisioned-instance-2 \
  --output text --query '*[].[DBInstanceIdentifier,DBInstanceClass]'
provisioned-instance-2      db.r6g.large
```

We create a table with some data. That way, we can confirm that the data and operation of the cluster are the same before and after the switchover.

```
mysql> create database serverless_v2_demo;
mysql> create table serverless_v2_demo.demo (s varchar(128));
mysql> insert into serverless_v2_demo.demo values ('This cluster started with a provisioned writer.');
Query OK, 1 row affected (0.02 sec)
```

First, we add a capacity range to the cluster. Otherwise, we get an error when adding any Aurora Serverless v2 DB instances to the cluster. If we use the AWS Management Console for this procedure, that step is automatic when we add the first Aurora Serverless v2 DB instance.

```
$ aws rds create-db-instance --db-instance-identifier serverless-v2-instance-1 \
--db-cluster-identifier mysql-80 --db-instance-class db.serverless --engine aurora-mysql

An error occurred (InvalidDBClusterStateFault) when calling the CreateDBInstance operation:
Set the Serverless v2 scaling configuration on the parent DB cluster before creating a
Serverless v2 DB instance.

$ # The blank ServerlessV2ScalingConfiguration attribute confirms that the cluster doesn't
have a capacity range set yet.
$ aws rds describe-db-clusters --db-cluster-identifier mysql-80 --query
'DBClusters[*].ServerlessV2ScalingConfiguration'
[]

$ aws rds modify-db-cluster --db-cluster-identifier mysql-80 \
--serverless-v2-scaling-configuration MinCapacity=0.5,MaxCapacity=16
{
  "DBClusterIdentifier": "mysql-80",
  "ServerlessV2ScalingConfiguration": {
    "MinCapacity": 0.5,
    "MaxCapacity": 16
  }
}
```

We create two Aurora Serverless v2 readers to take the place of the original DB instances, by specifying the db.serverless DB instance class for the new DB instances.

```
$ aws rds create-db-instance --db-instance-identifier serverless-v2-instance-1 --db-
cluster-identifier mysql-80 --db-instance-class db.serverless --engine aurora-mysql
{
  "DBInstanceIdentifier": "serverless-v2-instance-1",
  "DBClusterIdentifier": "mysql-80",
  "DBInstanceClass": "db.serverless",
  "DBInstanceState": "creating"
}

$ aws rds create-db-instance --db-instance-identifier serverless-v2-instance-2 \
--db-cluster-identifier mysql-80 --db-instance-class db.serverless --engine aurora-mysql
{
  "DBInstanceIdentifier": "serverless-v2-instance-2",
  "DBClusterIdentifier": "mysql-80",
  "DBInstanceClass": "db.serverless",
  "DBInstanceState": "creating"
}

$ # Wait for both DB instances to finish being created before proceeding.
$ aws rds wait db-instance-available --db-instance-identifier serverless-v2-instance-1 && \
aws rds wait db-instance-available --db-instance-identifier serverless-v2-instance-2
```

We perform a failover to make one of the Aurora Serverless v2 DB instances the new writer for the cluster.

```
$ aws rds failover-db-cluster --db-cluster-identifier mysql-80 \
--target-db-instance-identifier serverless-v2-instance-1
{
  "DBClusterIdentifier": "mysql-80",
  "DBClusterMembers": [
    {
      "DBInstanceIdentifier": "serverless-v2-instance-1",
      "IsClusterWriter": false,
      "DBClusterParameterGroupStatus": "in-sync",
      "PromotionTier": 1
    },
    {
      "DBInstanceIdentifier": "serverless-v2-instance-2",
      "IsClusterWriter": false,
      "DBClusterParameterGroupStatus": "in-sync",
      "PromotionTier": 1
    },
    {
      "DBInstanceIdentifier": "provisioned-instance-2",
      "IsClusterWriter": false,
      "DBClusterParameterGroupStatus": "in-sync",
      "PromotionTier": 1
    },
    {
      "DBInstanceIdentifier": "provisioned-instance-1",
      "IsClusterWriter": true,
      "DBClusterParameterGroupStatus": "in-sync",
      "PromotionTier": 1
    }
  ],
  "Status": "available"
}
```

It takes a few seconds for that change to take effect. At that point, we have an Aurora Serverless v2 writer and an Aurora Serverless v2 reader. Thus, we don't need either of the original provisioned DB instances.

```
$ aws rds describe-db-clusters --db-cluster-identifier mysql-80 \
--query '*[].[DBClusterIdentifier,DBClusterMembers[*].
[DBInstanceIdentifier,IsClusterWriter]]' \
--output text
mysql-80
serverless-v2-instance-1      True
serverless-v2-instance-2      False
provisioned-instance-2       False
provisioned-instance-1       False
```

The last step in the switchover procedure is to delete both of the provisioned DB instances.

```
$ aws rds delete-db-instance --db-instance-identifier provisioned-instance-2 --skip-final-
snapshot
{
  "DBInstanceIdentifier": "provisioned-instance-2",
  "DBInstanceState": "deleting",
  "Engine": "aurora-mysql",
  "EngineVersion": "8.0.mysql_aurora.3.02.0",
  "DBInstanceClass": "db.r6g.large"
}
```

```
$ aws rds delete-db-instance --db-instance-identifier provisioned-instance-1 --skip-final-snapshot
{
    "DBInstanceIdentifier": "provisioned-instance-1",
    "DBInstanceState": "deleting",
    "Engine": "aurora-mysql",
    "EngineVersion": "8.0.mysql_aurora.3.02.0",
    "DBInstanceClass": "db.r6g.large"
}
```

As a final check, we confirm that the original table is accessible and writeable from the Aurora Serverless v2 writer DB instance.

```
mysql> select * from serverless_v2_demo.demo;
+-----+
| s |
+-----+
| This cluster started with a provisioned writer. |
+-----+
1 row in set (0.00 sec)

mysql> insert into serverless_v2_demo.demo values ('And it finished with a Serverless v2 writer.');
Query OK, 1 row affected (0.01 sec)

mysql> select * from serverless_v2_demo.demo;
+-----+
| s |
+-----+
| This cluster started with a provisioned writer. |
| And it finished with a Serverless v2 writer. |
+-----+
2 rows in set (0.01 sec)
```

We also connect to the Aurora Serverless v2 reader DB instance and confirm that the newly written data is available there too.

```
mysql> select * from serverless_v2_demo.demo;
+-----+
| s |
+-----+
| This cluster started with a provisioned writer. |
| And it finished with a Serverless v2 writer. |
+-----+
2 rows in set (0.01 sec)
```

## Moving from Aurora Serverless v1 to Aurora Serverless v2

If you are already using Aurora Serverless v1 and want to use Aurora Serverless v2, familiarize yourself with the differences first. Following, you can learn how Aurora Serverless v1 differs from Aurora Serverless v2. You can also learn how you can move your DB clusters and applications from one to the other.

### Topics

- [Comparison of Aurora Serverless v2 and Aurora Serverless v1 \(p. 1500\)](#)
- [Adapting Aurora Serverless v1 use cases to Aurora Serverless v2 \(p. 1504\)](#)

- [Upgrading from an Aurora Serverless v1 cluster to Aurora Serverless v2 \(p. 1505\)](#)

## Comparison of Aurora Serverless v2 and Aurora Serverless v1

If you are already using Aurora Serverless v1, you can learn the major differences between Aurora Serverless v1 and Aurora Serverless v2. The architectural differences, such as support for reader DB instances, open up new types of use cases.

You can use the following tables to help understand the most important differences between Aurora Serverless v2 and Aurora Serverless v1.

### Topics

- [Comparison of Aurora Serverless v2 and Aurora Serverless v1 requirements \(p. 1500\)](#)
- [Comparison of Aurora Serverless v2 and Aurora Serverless v1 scaling and availability \(p. 1501\)](#)
- [Comparison of Aurora Serverless v2 and Aurora Serverless v1 feature support \(p. 1503\)](#)

### Comparison of Aurora Serverless v2 and Aurora Serverless v1 requirements

The following table summarizes the different requirements to run your database using Aurora Serverless v2 or Aurora Serverless v1. Aurora Serverless v2 offers higher versions of the Aurora MySQL and Aurora PostgreSQL DB engines than Aurora Serverless v1 does.

Feature	Aurora Serverless v2 requirement	Aurora Serverless v1 requirement
DB engines	Aurora MySQL, Aurora PostgreSQL	Aurora MySQL, Aurora PostgreSQL
Supported Aurora MySQL versions	Version 3.02.0, compatible with MySQL 8.0	Versions 1 and 2, compatible with MySQL 5.6 and 5.7
Supported Aurora PostgreSQL versions	Version 13.6, 13.7, or 14.3	Version 10.18 or 11.13
Converting from provisioned DB cluster	<p>You can use the following methods:</p> <ul style="list-style-type: none"><li>• Add one or more Aurora Serverless v2 reader DB instances to an existing provisioned cluster. To use Aurora Serverless v2 for the writer, perform a failover to one of the Aurora Serverless v2 DB instances. For the entire cluster to use Aurora Serverless v2 DB instances, remove any provisioned writer DB instances after promoting the Aurora Serverless v2 DB instance to the writer.</li><li>• Create a new cluster with the appropriate DB engine and engine version. Use any of the standard methods. For example, restore a cluster snapshot or create a clone of an existing cluster. Choose Aurora Serverless v2 for some or</li></ul>	Restore snapshot of provisioned cluster to create new Aurora Serverless v1 cluster.

Feature	Aurora Serverless v2 requirement	Aurora Serverless v1 requirement
	<p>all of the DB instances in the new cluster.</p> <p>If you create the new cluster through cloning, you can't upgrade the engine version at the same time. Make sure that the original cluster is already running an engine version that's compatible with Aurora Serverless v2.</p>	
Converting from Aurora Serverless v1 cluster	Follow the procedure in <a href="#">Upgrading from an Aurora Serverless v1 cluster to Aurora Serverless v2 (p. 1505)</a> .	Not applicable
Available DB instance classes	The special DB instance class <code>db.serverless</code> . In the AWS Management Console, it's labeled as <b>Serverless</b> .	Not applicable.
Port	Any port that's compatible with MySQL or PostgreSQL	Default MySQL or PostgreSQL port only
Public IP address allowed?	Yes	No
Virtual private cloud (VPC) required?	Yes	Yes. Each Aurora Serverless v1 cluster consumes 2 interface and Gateway Load Balancer endpoints allocated to your VPC.

## Comparison of Aurora Serverless v2 and Aurora Serverless v1 scaling and availability

The following table summarizes differences between Aurora Serverless v2 and Aurora Serverless v1 for scalability and availability.

Aurora Serverless v2 scaling is more responsive, more granular, and less disruptive than the scaling in Aurora Serverless v1. Aurora Serverless v2 can scale both by changing the size of the DB instance and by adding more DB instances to the DB cluster. It can also scale by adding clusters in other AWS Regions to an Aurora global database. In contrast, Aurora Serverless v1 only scales by increasing or decreasing the capacity of the writer. All the compute for an Aurora Serverless v1 cluster runs in a single Availability Zone and a single AWS Region.

Feature	Aurora Serverless v2 scaling and high availability behavior	Aurora Serverless v1 scaling and high availability behavior
Minimum Aurora capacity units (ACUs) (Aurora MySQL)	0.5	1 when the cluster is running, 0 when the cluster is paused.
Minimum ACUs (Aurora PostgreSQL)	0.5	2 when the cluster is running, 0 when the cluster is paused.
Maximum ACUs (Aurora MySQL)	128	256

Feature	Aurora Serverless v2 scaling and high availability behavior	Aurora Serverless v1 scaling and high availability behavior
Maximum ACUs (Aurora PostgreSQL)	128	384
Stopping a cluster	You can manually stop and start the cluster by using <a href="#">the same cluster stop and start feature</a> as provisioned clusters.	The cluster pauses automatically after a timeout. It takes some time to become available when activity resumes.
Scaling for DB instances	Scale up and down with minimum increment of 0.5 ACUs	Scale up and down by doubling or halving the ACUs
Number of DB instances	Same as a provisioned cluster: 1 writer DB instance, up to 15 reader DB instances.	1 DB instance handling both reads and writes.
Scaling can happen while SQL statements are running?	Yes. Aurora Serverless v2 doesn't require waiting for a quiet point.	No. For example, scaling waits for completion of long-running transactions, temporary tables, and table locks.
Reader DB instances scale along with writer	Optional.	Not applicable.
Maximum storage	128 TiB	128 TiB or 64 TiB, depending on database engine and version.
Buffer cache preserved when scaling	Yes. Buffer cache is resized dynamically.	No. Buffer cache is rewarmed after scaling.
Failover	Yes, same as for provisioned clusters.	Best effort only, subject to capacity availability. Slower than in Aurora Serverless v2.
Multi-AZ capability	Yes, same as for provisioned. A Multi-AZ cluster requires a reader DB instance in a second Availability Zone (AZ). For a Multi-AZ cluster, Aurora performs Multi-AZ failover in case of an AZ failure.	Aurora Serverless v1 clusters run all their compute in a single AZ. Recovery in case of AZ failure is best effort only and subject to capacity availability.
Aurora global databases	Yes	No
Scaling based on memory pressure	Yes	No
Scaling based on CPU load	Yes	Yes
Scaling based on network traffic	Yes, based on memory and CPU overhead of network traffic. The <code>max_connections</code> parameter remains constant to avoid dropping connections when scaling down.	Yes, based on number of connections.
Timeout action for scaling events	No	Yes

Feature	Aurora Serverless v2 scaling and high availability behavior	Aurora Serverless v1 scaling and high availability behavior
Adding new DB instances to cluster through AWS Auto Scaling	Not applicable. You can create Aurora Serverless v2 reader DB instances in promotion tiers 2–15 and leave them scaled down to low capacity.	No. Reader DB instances aren't available.

## Comparison of Aurora Serverless v2 and Aurora Serverless v1 feature support

The following table summarizes these:

- Features that are available in Aurora Serverless v2 but not Aurora Serverless v1
- Features that work differently between Aurora Serverless v1 and Aurora Serverless v2
- Features that aren't currently available in Aurora Serverless v2

Aurora Serverless v2 includes many features from provisioned clusters that aren't available for Aurora Serverless v1.

Feature	Aurora Serverless v2 features	Aurora Serverless v1 features
Cluster topology	Aurora Serverless v2 is a property of individual DB instances. A cluster can contain multiple Aurora Serverless v2 DB instances, or a combination of Aurora Serverless v2 and provisioned DB instances.	Aurora Serverless v1 clusters don't use the notion of DB instances. You can't change the Aurora Serverless v1 property after you create the cluster.
Configuration parameters	Almost all the same parameters can be modified as in provisioned clusters. For details, see <a href="#">Working with parameter groups for Aurora Serverless v2 (p. 1535)</a> .	Only a subset of parameters can be modified.
Parameter groups	Cluster parameter group and DB parameter groups. Parameters with provisioned value in <code>SupportedEngineModes</code> attribute are available. That's many more parameters than in Aurora Serverless v1.	Cluster parameter group only. Parameters with <code>serverless</code> value in <code>SupportedEngineModes</code> attribute are available.
Encryption for cluster volume	Optional.	Required. The limitations in <a href="#">Limitations of Amazon Aurora encrypted DB clusters (p. 1640)</a> apply to all Aurora Serverless v1 clusters.
Cross-Region snapshots	Yes	Snapshot must be encrypted with your own AWS Key Management Service (AWS KMS) key.
TLS/SSL	Yes. The support is the same as for provisioned clusters. For usage	Yes. There are some differences from TLS support for provisioned clusters.

Feature	Aurora Serverless v2 features	Aurora Serverless v1 features
	information, see <a href="#">Using TLS/SSL with Aurora Serverless v2 (p. 1521)</a> .	For usage information, see <a href="#">Using TLS/SSL with Aurora Serverless v1 (p. 1546)</a> .
Cloning	Only from and to DB engine versions that are compatible with Aurora Serverless v2. You can't use cloning to upgrade from Aurora Serverless v1 or from an earlier version of a provisioned cluster.	Only from and to DB engine versions that are compatible with Aurora Serverless v1.
Integration with Amazon S3	Yes	Yes
Exporting DB cluster snapshots to S3	Yes	No
Associating an IAM role	Yes	No
Uploading logs to Amazon CloudWatch	Optional. You choose which logs to turn on and which logs to upload to CloudWatch.	All logs that are turned on are uploaded to CloudWatch automatically.
Data API available	No	Yes
Query editor available	No	Yes
Performance Insights	Yes	No
Amazon RDS Proxy available	Yes	No
Babelfish for Aurora PostgreSQL available	Yes. Supported for Aurora PostgreSQL versions that are compatible with both Babelfish and Aurora Serverless v2.	No

## Adapting Aurora Serverless v1 use cases to Aurora Serverless v2

Depending on your use case for Aurora Serverless v1, you might adapt that approach to take advantage of Aurora Serverless v2 features as follows.

Suppose that you have an Aurora Serverless v1 cluster that is lightly loaded and your priority is maintaining continuous availability while minimizing costs. With Aurora Serverless v2, you can configure a smaller minimum ACU setting of 0.5, compared with a minimum of 1 ACU for Aurora Serverless v1. You can increase availability by creating a Multi-AZ configuration, with the reader DB instance also having a minimum of 0.5 ACUs.

Suppose that you have an Aurora Serverless v1 cluster that you use in a development and test scenario. In this case, cost is also a high priority but the cluster doesn't need to be available at all times. Currently, Aurora Serverless v2 doesn't automatically pause when the cluster is completely idle. Instead, you can manually stop the cluster when it's not needed, and start it when it's time for the next test or development cycle.

Suppose that you have an Aurora Serverless v1 cluster with a heavy workload. An equivalent cluster using Aurora Serverless v2 can scale with more granularity. For example, Aurora Serverless v1 scales by doubling the capacity, for example from 64 to 128 ACUs. In contrast, your Aurora Serverless v2 DB instance can scale to a value somewhere between those numbers.

Suppose that your workload requires a higher total capacity than is available in Aurora Serverless v1. You can use multiple Aurora Serverless v2 reader DB instances to offload the read-intensive parts of the workload from the writer DB instance. You can also divide the read-intensive workload among multiple reader DB instances.

For a write-intensive workload, you might configure the cluster with a large provisioned DB instance as the writer, alongside one or more Aurora Serverless v2 reader DB instances.

## Upgrading from an Aurora Serverless v1 cluster to Aurora Serverless v2

The process of upgrading a DB cluster from Aurora Serverless v1 to Aurora Serverless v2 involves several steps. That's because Aurora Serverless v1 isn't available for the same major Aurora MySQL and Aurora PostgreSQL versions as Aurora Serverless v2 is. Thus, going from Aurora Serverless v1 to Aurora Serverless v2 always involves at least one major version upgrade.

### To upgrade an Aurora Serverless v1 cluster running Aurora MySQL version 2 (MySQL 5.7–compatible)

1. Create a DB cluster snapshot of the Aurora Serverless v1 cluster. Follow the procedure in [Creating a DB cluster snapshot \(p. 373\)](#).
2. Restore the snapshot to create a new, provisioned (that is, *not* Aurora Serverless) DB cluster running Aurora MySQL version 2. Follow the procedure in [Restoring from a DB cluster snapshot \(p. 375\)](#).  
Choose the latest minor engine version available for the new cluster, for example, 2.10.2.
3. Create a DB cluster snapshot of the provisioned Aurora MySQL version 2 cluster.
4. Restore the snapshot to create a new, provisioned DB cluster running Aurora MySQL version 3 that's compatible with Aurora Serverless v2, for example, 3.02.0.  
For compatible versions, see [Aurora Serverless v2 \(p. 31\)](#).
5. Modify the writer DB instance of the provisioned DB cluster to use the **Serverless v2** DB instance class.

For details, see [Converting a provisioned writer or reader to Aurora Serverless v2 \(p. 1518\)](#).

### To upgrade an Aurora Serverless v1 cluster running Aurora MySQL version 1 (MySQL 5.6–compatible)

1. Perform an in-place upgrade of the Aurora Serverless v1 cluster. Follow the procedure in [Modifying an Aurora Serverless v1 DB cluster \(p. 1570\)](#).
2. Follow the steps in the previous procedure for upgrading from an Aurora Serverless v1 DB cluster running Aurora MySQL version 2.

### To upgrade an Aurora Serverless v1 cluster running Aurora PostgreSQL

1. Create a DB cluster snapshot of the Aurora Serverless v1 cluster. Follow the procedure in [Creating a DB cluster snapshot \(p. 373\)](#).
2. Restore the snapshot to create a new, provisioned (that is, *not* Aurora Serverless) DB cluster. Follow the procedure in [Restoring from a DB cluster snapshot \(p. 375\)](#).  
Choose the latest minor engine version available for the new cluster, for example, 10.21.
3. Modify the provisioned DB cluster to upgrade it to an Aurora PostgreSQL version that's compatible with Aurora Serverless v2, for example, 13.7.

Follow the procedure in [Upgrading the Aurora PostgreSQL engine to a new major version \(p. 1422\)](#). For compatible versions, see [Aurora Serverless v2 \(p. 31\)](#).

4. Modify the writer DB instance of the provisioned DB cluster to use the **Serverless v2** DB instance class.

For details, see [Converting a provisioned writer or reader to Aurora Serverless v2 \(p. 1518\)](#).

## Upgrading from Aurora Serverless v2 (preview) to Aurora Serverless v2

Any Aurora MySQL clusters that you created using the Aurora Serverless v2 preview can't be upgraded using the snapshot restore mechanism. The preview is intended for testing only. Your preview clusters shouldn't contain any production or business-critical data. If you need to bring any data from an Aurora Serverless v2 preview cluster to Aurora Serverless v2, perform a logical dump and restore. Use the `mysqldump` command as described in [Migrating from MySQL to Amazon Aurora by using mysqldump \(p. 705\)](#).

## Migrating from an on-premises database to Aurora Serverless v2

You can migrate your on-premises databases to Aurora Serverless v2, just as with provisioned Aurora MySQL and Aurora PostgreSQL.

- For MySQL databases, you can use the `mysqldump` command. For more information, see [Importing data to a MySQL or MariaDB DB instance with reduced downtime](#).
- For PostgreSQL databases, you can use the `pg_dump` and `pg_restore` commands. For more information, see the blog post [Best practices for migrating PostgreSQL databases to Amazon RDS and Amazon Aurora](#).

## Creating a cluster that uses Aurora Serverless v2

To create an Aurora cluster where you can add Aurora Serverless v2 DB instances, you follow the same procedure as in [Creating an Amazon Aurora DB cluster \(p. 127\)](#). With Aurora Serverless v2, your clusters are interchangeable with provisioned clusters. You can have clusters where some DB instances use Aurora Serverless v2 and some DB instances are provisioned.

If you intend to use Aurora Serverless v2 DB instances in a cluster, make sure that the cluster's initial settings meet the requirements that are listed in [Requirements for Aurora Serverless v2 \(p. 1490\)](#). Specify the following settings to make sure that you can add Aurora Serverless v2 DB instances to the cluster:

### AWS Region

Create the cluster in an AWS Region where Aurora Serverless v2 DB instances are available. For details about available Regions, see [Aurora Serverless v2 \(p. 31\)](#).

### DB engine version

Choose an engine version that's compatible with Aurora Serverless v2. For information about the Aurora Serverless v2 version requirements, see [Requirements for Aurora Serverless v2 \(p. 1490\)](#). On

the **Create database** page in the AWS Management Console, you can choose the filter setting **Show versions that support Serverless v2**. You can also see the applicable versions in the **Info** panel on the **Create database** page.

#### DB instance class

If you create a cluster using the AWS Management Console, you choose the DB instance class for the writer DB instance at the same time. Choose the **Serverless** DB instance class. When you choose that DB instance class, you also specify the capacity range for the writer DB instance. That same capacity range applies to all other Aurora Serverless v2 DB instances that you add to that cluster. If you don't see the **Serverless** choice for the DB instance class, make sure that you chose a DB engine version that's compatible with Aurora Serverless v2.

When you use the AWS CLI or the Amazon RDS API, the parameter that you specify for the DB instance class parameter is `db.serverless`.

#### capacity range

Fill in the minimum and maximum Aurora capacity unit (ACU) values that apply to all the DB instances in the cluster. This option is available on both the **Create cluster** and **Add reader** console pages when you choose **Serverless** for the DB instance class.

If you don't see the minimum and maximum ACU boxes, make sure that you chose the **Serverless** DB instance class for the writer DB instance.

#### Tip

A simple way to set up a new cluster where you can use Aurora Serverless v2 is to choose the **Easy create** setting on the console **Create database** page. With that setting, you only make a single other choice: whether the cluster is intended for development and test use or for production. Both of those options set up a cluster with an Aurora Serverless v2 writer DB instance.

If you initially create the cluster with a provisioned DB instance, you don't specify the minimum and maximum ACUs. In that case you can modify the cluster afterward to add that setting. You can also add an Aurora Serverless v2 reader DB instance to the cluster. You specify the capacity range as part of that process.

Until you specify the capacity range for your cluster, you can't add any Aurora Serverless v2 DB instances to the cluster using the AWS CLI or RDS API. If you try to add a Aurora Serverless v2 DB instance, you get an error. In the AWS CLI or the RDS API procedures, the capacity range is represented by the `ServerlessV2ScalingConfiguration` attribute.

For clusters containing more than one reader DB instance, the failover priority of each Aurora Serverless v2 reader DB instance plays an important part in how that DB instance scales up and down. You can't specify the priority when you initially create the cluster. Keep this property in mind when you add a second or later reader DB instance to your cluster. For more information, see [Choosing the promotion tier for an Aurora Serverless v2 reader \(p. 1521\)](#).

## Console

### To create a cluster with an Aurora Serverless v2 writer

1. Sign in using the AWS Management Console and open the Amazon RDS console.
2. Choose **Create Database**. On the page that appears, choose the following options:
  - For **Engine type**, choose Aurora.
  - For **Edition**, choose Aurora MySQL or Aurora PostgreSQL.

- For **Version**, choose one of the compatible versions from [Aurora Serverless v2 \(p. 31\)](#). To see the available versions, choose the filter **Show versions that support Serverless v2**.
- For **Capacity settings**, you can accept the default range. Or you can choose other values for minimum and maximum capacity units. You can choose from 0.5 ACUs minimum through 128 ACUs maximum, in increments of 0.5 ACU.

For more information about Aurora Serverless v2 capacity units, see [Aurora Serverless v2 capacity \(p. 1486\)](#) and [Performance and scaling for Aurora Serverless v2 \(p. 1526\)](#).

Capacity range [Info](#)  
Database capacity is measured in Aurora Capacity Units (ACUs). 1 ACU provides 2 GiB of memory and corresponding compute and networking.

Minimum	Maximum
<input type="text" value="1"/> ACUs (2 GiB)	<input type="text" value="64"/> ACUs (128 GiB)
0.5 to 128 in increments of 0.5	0.5 to 128 in increments of 0.5

- Choose any other cluster settings, as described in [Creating an Amazon Aurora DB cluster \(p. 127\)](#).
- Choose **Create database** to create your Aurora cluster with an Aurora Serverless v2 DB instance as the writer instance, also known as the primary DB instance.

## CLI

To create a DB cluster that's compatible with Aurora Serverless v2 DB instances using the AWS CLI, you follow the CLI procedure in [Creating an Amazon Aurora DB cluster \(p. 127\)](#). Include the following parameters in your `create-db-cluster` command:

- `--region AWS_Region_where_Aurora_Serverless_v2_instances_are_available`
- `--engine-version serverless_v2_compatible_engine_version`
- `--serverless-v2-scaling-configuration MinCapacity=minimum_capacity,MaxCapacity=maximum_capacity`

The following example shows the creation of an Aurora Serverless v2 DB cluster.

```
aws rds create-db-cluster \
  --db-cluster-identifier my-serverless-v2-cluster \
  --region eu-central-1 \
  --engine aurora-mysql \
  --engine-version 8.0.mysql_aurora.3.02.0 \
  --serverless-v2-scaling-configuration MinCapacity=1,MaxCapacity=4 \
  --master-username myuser \
  --master-user-password mypassword \
  --backup-retention 1
```

### Note

When you create an Aurora Serverless v2 DB cluster using the AWS CLI, the engine mode appears in the output as provisioned rather than serverless. The serverless engine mode refers to Aurora Serverless v1.

For information about the Aurora Serverless v2 version requirements, see [Requirements for Aurora Serverless v2 \(p. 1490\)](#). For information about the allowed numbers for the capacity range and what those numbers represent, see [Aurora Serverless v2 capacity \(p. 1486\)](#) and [Performance and scaling for Aurora Serverless v2 \(p. 1526\)](#).

To verify whether an existing cluster has the capacity settings specified, check the output of the `describe-db-clusters` command for the `ServerlessV2ScalingConfiguration` attribute. That attribute looks similar to the following.

```
"ServerlessV2ScalingConfiguration": {  
    "MinCapacity": 1.5,  
    "MaxCapacity": 24.0  
}
```

**Tip**

If you don't specify the minimum and maximum ACUs when you create the cluster, you can use the `modify-db-cluster` command afterward to add that setting. Until you do, you can't add any Aurora Serverless v2 DB instances to the cluster. If you try to add a `db.serverless` DB instance, you get an error.

## API

To create a DB cluster that's compatible with Aurora Serverless v2 DB instances using the RDS API, you follow the API procedure in [Creating an Amazon Aurora DB cluster \(p. 127\)](#). Choose the following settings. Make sure that your `CreateDBCluster` operation includes the following parameters:

```
EngineVersion serverless_v2_compatible_engine_version  
ServerlessV2ScalingConfiguration with MinCapacity=minimum_capacity and  
MaxCapacity=maximum_capacity
```

For information about the Aurora Serverless v2 version requirements, see [Requirements for Aurora Serverless v2 \(p. 1490\)](#). For information about the allowed numbers for the capacity range and what those numbers represent, see [Aurora Serverless v2 capacity \(p. 1486\)](#) and [Performance and scaling for Aurora Serverless v2 \(p. 1526\)](#).

To check if an existing cluster has the capacity settings specified, check the output of the `DescribeDBClusters` operation for the `ServerlessV2ScalingConfiguration` attribute. That attribute looks similar to the following.

```
"ServerlessV2ScalingConfiguration": {  
    "MinCapacity": 1.5,  
    "MaxCapacity": 24.0  
}
```

**Tip**

If you don't specify the minimum and maximum ACUs when you create the cluster, you can use the `ModifyDBCluster` operation afterward to add that setting. Until you do, you can't add any Aurora Serverless v2 DB instances to the cluster. If you try to add a `db.serverless` DB instance, you get an error.

# Managing Aurora Serverless v2

With Aurora Serverless v2, your clusters are interchangeable with provisioned clusters. The Aurora Serverless v2 properties apply to one or more DB instances within a cluster. Thus, the procedures for creating clusters, modifying clusters, creating and restoring snapshots, and so on, are basically the same as for other kinds of Aurora clusters. For general procedures for managing Aurora clusters and DB instances, see [Managing an Amazon Aurora DB cluster \(p. 243\)](#).

Following, you can learn about management considerations for clusters that contain Aurora Serverless v2 DB instances.

### Topics

- [Setting the Aurora Serverless v2 capacity range for a cluster \(p. 1510\)](#)
- [Checking the capacity range for Aurora Serverless v2 \(p. 1513\)](#)
- [Creating an Aurora Serverless v2 writer \(p. 1515\)](#)
- [Adding an Aurora Serverless v2 reader \(p. 1517\)](#)
- [Converting a provisioned writer or reader to Aurora Serverless v2 \(p. 1518\)](#)
- [Converting an Aurora Serverless v2 writer or reader to provisioned \(p. 1520\)](#)
- [Choosing the promotion tier for an Aurora Serverless v2 reader \(p. 1521\)](#)
- [Using TLS/SSL with Aurora Serverless v2 \(p. 1521\)](#)
- [Viewing Aurora Serverless v2 writers and readers \(p. 1523\)](#)
- [Logging for Aurora Serverless v2 \(p. 1523\)](#)

## Setting the Aurora Serverless v2 capacity range for a cluster

To modify configuration parameters or other settings for clusters containing Aurora Serverless v2 DB instances, or the DB instances themselves, follow the same general procedures as for provisioned clusters. For details, see [Modifying an Amazon Aurora DB cluster \(p. 248\)](#).

The most important setting that's unique to Aurora Serverless v2 is the capacity range. After you set the minimum and maximum Aurora capacity unit (ACU) values for an Aurora cluster, you don't need to actively adjust the capacity of the Aurora Serverless v2 DB instances in the cluster. Aurora does that for you. This setting is managed at the cluster level. The same minimum and maximum ACU values apply to each Aurora Serverless v2 DB instance in the cluster.

You can set the following specific values:

- **Minimum ACUs** – The Aurora Serverless v2 DB instance can reduce capacity down to this number of ACUs.
- **Maximum ACUs** – The Aurora Serverless v2 DB instance can increase capacity up to this number of ACUs.

For details about the effects of the capacity range and how to monitor and fine-tune it, see [Important Amazon CloudWatch metrics for Aurora Serverless v2 \(p. 1539\)](#) and [Performance and scaling for Aurora Serverless v2 \(p. 1526\)](#). Your goal is to make sure that the maximum capacity for the cluster is high enough to handle spikes in workload, and the minimum is low enough to minimize costs when the cluster isn't busy.

Suppose that you determine based on your monitoring that the ACU range for the cluster should be higher, lower, wider, or narrower. You can set the capacity of an Aurora cluster to a specific range of ACUs with the AWS Management Console, the AWS CLI, or the Amazon RDS API. This capacity range applies to every Aurora Serverless v2 DB instance in the cluster.

For example, suppose that your cluster has a capacity range of 1–16 ACUs and contains two Aurora Serverless v2 DB instances. Then the cluster as a whole consumes somewhere between 2 ACUs (when idle) and 32 ACUs (when fully utilized). If you change the capacity range from 8 to 20.5 ACUs, now the cluster consumes 16 ACUs when idle, and up to 41 ACUs when fully utilized.

Aurora automatically sets certain parameters for Aurora Serverless v2 DB instances to values that depend on the maximum ACU value in the capacity range. For the list of such parameters, see [Parameters that Aurora computes based on Aurora Serverless v2 maximum capacity \(p. 1537\)](#). For static parameters that rely on this type of calculation, the value is evaluated again when you reboot the DB

instance. Thus, you can update the value of such parameters by rebooting the DB instance after changing the capacity range. To check whether you need to reboot your DB instance to pick up such parameter changes, check the `ParameterApplyStatus` attribute of the DB instance. A value of `pending-reboot` indicates that rebooting will apply changes to some parameter values.

## Console

You can set the capacity range of a cluster that contains Aurora Serverless v2 DB instances with the AWS Management Console.

When you use the console, you set the capacity range for the cluster at the time that you add the first Aurora Serverless v2 DB instance to that cluster. You might do so when you choose the **Serverless v2** DB instance class for the writer DB instance when you create the cluster. Or you might do so when you choose the **Serverless** DB instance class when you add an Aurora Serverless v2 reader DB instance to the cluster. Or you might do so when you convert an existing provisioned DB instance in the cluster to the **Serverless** DB instance class. For the full versions of those procedures, see [Creating an Aurora Serverless v2 writer \(p. 1515\)](#), [Adding an Aurora Serverless v2 reader \(p. 1517\)](#), and [Converting a provisioned writer or reader to Aurora Serverless v2 \(p. 1518\)](#)

Whatever capacity range that you set at the cluster level applies to all Aurora Serverless v2 DB instances in your cluster. The following image shows a cluster with multiple Aurora Serverless v2 reader DB instances. Each has an identical capacity range of 2–64 ACUs.

Databases					
<input type="text"/> Filter by databases					
DB identifier	Role	Engine	Engine version	Region & AZ	
serverless-v2-cluster	Regional cluster	Aurora MySQL	8.0.mysql_aurora.3.02.0	eu-central-1	
serverless-v2-cluster-reader-1	Reader instance	Aurora MySQL	8.0.mysql_aurora.3.02.0	eu-central-1c	
serverless-v2-cluster-reader-2	Reader instance	Aurora MySQL	8.0.mysql_aurora.3.02.0	eu-central-1c	
serverless-v2-cluster-instance-1	Writer instance	Aurora MySQL	8.0.mysql_aurora.3.02.0	eu-central-1c	

## To modify the capacity range of an Aurora Serverless v2 cluster

1. Open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. In the navigation pane, choose **Databases**.
3. Choose the cluster containing your Aurora Serverless v2 DB instances from the list. The cluster must already contain at least one Aurora Serverless v2 DB instance. Otherwise, Aurora doesn't show the **Capacity range** section.
4. For **Actions**, choose **Modify**.
5. In the **Capacity range** section, choose the following:
  - a. Enter a value for **Minimum ACUs**. The console shows the allowed range of values. You can choose from 0.5 ACUs minimum to 128 ACUs maximum, in increments of 0.5 ACU.
  - b. Enter a value for **Maximum ACUs**. This value must be greater than or equal to **Minimum ACUs**. The console shows the allowed range of values. The following screenshot shows that choice.

## Serverless v2 capacity settings

### Capacity range Info

Database capacity is measured in Aurora Capacity Units (ACUs). 1 ACU provides 2 GiB of memory and 1 GiB of networking.

Minimum ACUs

2

(4 GiB)

0.5 to 128 in increments of 0.5

Maximum ACUs

64

(128 GiB)

0.5 to 128 in increments of 0.5

 The capacity range applies to all Serverless v2 instances in your cluster, including `serverless-v2-another-reader`, `serverless-v2-reader`.

6. Choose **Continue**. The **Summary of modifications** page appears.
7. Choose whether to apply the capacity change immediately, or during the next scheduled maintenance window.
8. Choose **Modify cluster** to accept the summary of modifications. You can also choose **Back** to modify your changes or **Cancel** to discard your changes.

### AWS CLI

To set the capacity of a cluster where you intend to use Aurora Serverless v2 DB instances using the AWS CLI, run the `modify-db-cluster` AWS CLI command. Specify the `--serverless-v2-scaling-configuration` option. The cluster might already contain one or more Aurora Serverless v2 DB instances, or you can add the DB instances later. Valid values for the `MinCapacity` and `MaxCapacity` fields include the following:

- 0.5, 1, 1.5, 2, and so on, in steps of 0.5, up to a maximum of 128.

In this example, you set the ACU range of an Aurora DB cluster named `sample-cluster` to a minimum of 48.5 and a maximum of 64.

```
aws rds modify-db-cluster --db-cluster-identifier sample-cluster --serverless-v2-scaling-configuration MinCapacity=48.5,MaxCapacity=64
```

After doing so, you can add Aurora Serverless v2 DB instances to the cluster and each new DB instance can scale between 48.5 and 64 ACUs. The new capacity range also applies to any Aurora Serverless v2 DB instances that were already in the cluster. The DB instances scale up or down if necessary to fall within the new capacity range.

For additional examples of setting the capacity range using the CLI, see [Choosing the Aurora Serverless v2 capacity range for an Aurora cluster \(p. 1527\)](#).

To modify the scaling configuration of an Aurora Serverless DB cluster using the AWS CLI, run the [modify-db-cluster](#) AWS CLI command. Specify the `--serverless-v2-scaling-configuration` option to configure the minimum capacity and maximum capacity. Valid capacity values include the following:

- Aurora MySQL: 0.5, 1, 1.5, 2, and so on, in increments of 0.5 ACUs up to a maximum of 128.
- Aurora PostgreSQL: 0.5, 1, 1.5, 2, and so on, in increments of 0.5 ACUs up to a maximum of 128.

In the following example, you modify the scaling configuration of an Aurora Serverless v2 DB instance named `sample-instance` that's part of a cluster named `sample-cluster`.

For Linux, macOS, or Unix:

```
aws rds modify-db-cluster --db-cluster-identifier sample-cluster \
--serverless-v2-scaling-configuration MinCapacity=8,MaxCapacity=64
```

For Windows:

```
aws rds modify-db-cluster --db-cluster-identifier sample-cluster ^
--serverless-v2-scaling-configuration MinCapacity=8,MaxCapacity=64
```

## RDS API

You can set the capacity of an Aurora DB instance with the [ModifyDBCluster](#) API operation. Specify the `ServerlessV2ScalingConfiguration` parameter. Valid values for the `MinCapacity` and `MaxCapacity` fields include the following:

- 0.5, 1, 1.5, 2, and so on, in steps of 0.5, up to a maximum of 128.

You can modify the scaling configuration of a cluster containing Aurora Serverless v2 DB instances with the [ModifyDBCluster](#) API operation. Specify the `ServerlessV2ScalingConfiguration` parameter to configure the minimum capacity and the maximum capacity. Valid capacity values include the following:

- Aurora MySQL: 0.5, 1, 1.5, 2, and so on, in increments of 0.5 ACUs up to a maximum of 128.
- Aurora PostgreSQL: 0.5, 1, 1.5, 2, and so on, in increments of 0.5 ACUs up to a maximum of 128.

## Checking the capacity range for Aurora Serverless v2

The procedure to check the capacity range for your Aurora Serverless v2 cluster requires that you first set a capacity range. If you haven't done so, follow the procedure in [Setting the Aurora Serverless v2 capacity range for a cluster \(p. 1510\)](#).

Whatever capacity range you set at the cluster level applies to all Aurora Serverless v2 DB instances in your cluster. The following image shows a cluster with multiple Aurora Serverless v2 DB instances. Each has an identical capacity range.

Databases					
<input type="text"/> Filter by databases					
DB identifier	Role	Engine	Engine version	Region & AZ	
serverless-v2-cluster	Regional cluster	Aurora MySQL	8.0.mysql_aurora.3.02.0	eu-central-1	
serverless-v2-cluster-reader-1	Reader instance	Aurora MySQL	8.0.mysql_aurora.3.02.0	eu-central-1c	
serverless-v2-cluster-reader-2	Reader instance	Aurora MySQL	8.0.mysql_aurora.3.02.0	eu-central-1c	
serverless-v2-cluster-instance-1	Writer instance	Aurora MySQL	8.0.mysql_aurora.3.02.0	eu-central-1c	

You can also view the details page for any Aurora Serverless v2 DB instance in the cluster. DB instances' capacity range appears on the **Configuration** tab.

## Instance configuration

Instance type

Serverless v2

Minimum capacity

2 ACUs (4 GiB)

Maximum capacity

64 ACUs (128 GiB)

You can also see the current capacity range for the cluster on the **Modify** page for the cluster. The following image shows how. At that point, you can change the capacity range. For all the ways that you can set or change the capacity range, see [Setting the Aurora Serverless v2 capacity range for a cluster \(p. 1510\)](#).

## Serverless v2 capacity settings

### Capacity range [Info](#)

Database capacity is measured in Aurora Capacity Units (ACUs). 1 ACU provides 2 GiB of memory and networking.

Minimum ACUs

2

(4 GiB)

0.5 to 128 in increments of 0.5

Maximum ACUs

64

(128 GiB)

0.5 to 128 in increments of 0.5

 The capacity range applies to all Serverless v2 instances in your cluster. Any existing reader instances, such as serverless-v2-another-reader,serverless-v2-reader, will inherit the capacity range.

## Checking the current capacity range for an Aurora cluster

You can check the capacity range that's configured for Aurora Serverless v2 DB instances in a cluster by examining the `ServerlessV2ScalingConfiguration` attribute for the cluster. The following AWS CLI example shows a cluster with a minimum capacity of 0.5 Aurora capacity units (ACUs) and a maximum capacity of 16 ACUs.

```
$ aws rds describe-db-clusters --db-cluster-identifier serverless-v2-64-acu-cluster \
--query 'DBClusters[*].[ServerlessV2ScalingConfiguration]'
[
  [
    {
      "MinCapacity": 0.5,
      "MaxCapacity": 16.0
    }
  ]
]
```

## Creating an Aurora Serverless v2 writer

### Console

When you create a cluster using the AWS Management Console, you specify the properties of the writer DB instance at the same time. To make the writer DB instance use Aurora Serverless v2, choose the **Serverless** DB instance class. Then you set the capacity range for the cluster by specifying the minimum and maximum Aurora capacity unit (ACU) values. These minimum and maximum values apply to each Aurora Serverless v2 DB instance in the cluster.

## Instance configuration

The DB instance configuration options below are limited to those supported by the engine.

### DB instance class [Info](#)

- Serverless v2
- Memory optimized classes (includes r classes)
- Burstable classes (includes t classes)

### Capacity range [Info](#)

Database capacity is measured in Aurora Capacity Units (ACUs). 1 ACU provides 2 GiB of memory and 1 GiB of networking.

#### Minimum ACUs

4.5  (8 GiB)

0.5 to 128 in increments of 0.5

#### Maximum ACUs

16  (32 GiB)

0.5 to 128 in increments of 0.5

If you don't create an Aurora Serverless v2 DB instance when you first create the cluster, you can add one or more Aurora Serverless v2 DB instances later. To do so, follow the procedures in [Adding an Aurora Serverless v2 reader \(p. 1517\)](#) and [Converting a provisioned writer or reader to Aurora Serverless v2 \(p. 1518\)](#). You specify the capacity range at the time that you add the first Aurora Serverless v2 DB instance to the cluster. You can change the capacity range later by following the procedure in [Setting the Aurora Serverless v2 capacity range for a cluster \(p. 1510\)](#).

### CLI

When you create a Aurora Serverless v2 DB cluster using the AWS CLI, you explicitly add the writer DB instance using the `create-db-instance` command. Include the following parameter:

- `--db-instance-class db.serverless`

The following example shows the creation of an Aurora Serverless v2 writer DB instance.

```
aws rds create-db-instance \
--db-cluster-identifier my-serverless-v2-cluster \
--db-instance-identifier my-serverless-v2-instance \
--db-instance-class db.serverless \
--engine aurora-mysql
```

## Adding an Aurora Serverless v2 reader

To add an Aurora Serverless v2 reader DB instance to your cluster, you follow the same general procedure as in [Adding Aurora Replicas to a DB cluster \(p. 270\)](#). Choose the **Serverless v2** instance class for the new DB instance.

If the reader DB instance is the first Aurora Serverless v2 DB instance in the cluster, you also choose the capacity range. The following image shows the controls that you use to specify the minimum and maximum Aurora capacity units (ACUs). This setting applies to this reader DB instance and to any other Aurora Serverless v2 DB instances that you add to the cluster. Each Aurora Serverless v2 DB instance can scale between the minimum and maximum ACU values.

### Instance configuration

The DB instance configuration options below are limited to those supported by the engine.

#### DB instance class [Info](#)

- Serverless v2**
- Memory optimized classes (includes r classes)**
- Burstable classes (includes t classes)**

#### Capacity range [Info](#)

Database capacity is measured in Aurora Capacity Units (ACUs). 1 ACU provides 2 GiB of memory and networking.

##### Minimum ACUs

4.5	▼	(8 GiB)
-----	---	---------

0.5 to 128 in increments of 0.5

##### Maximum ACUs

16	(32 GiB)
----	----------

0.5 to 128 in increments of 0.5

If you already added any Aurora Serverless v2 DB instances to the cluster, adding another Aurora Serverless v2 reader DB instance shows you the current capacity range. The following image shows those read-only controls.

## Instance configuration

The DB instance configuration options below are limited to those supported by the engine that you selected.

### DB instance class [Info](#)

- Serverless v2
- Memory optimized classes (includes r classes)
- Burstable classes (includes t classes)

### Capacity range [Info](#)

Database capacity is measured in Aurora Capacity Units (ACUs). 1 ACU provides 2 GiB of memory and networking.

Minimum ACUs

2 ACUs (4 GiB)

Maximum ACUs

64 ACUs (128 GiB)

If you want to change the capacity range for the cluster, follow the procedure in [Setting the Aurora Serverless v2 capacity range for a cluster \(p. 1510\)](#).

For clusters containing more than one reader DB instance, the failover priority of each Aurora Serverless v2 reader DB instance plays an important part in how that DB instance scales up and down. You can't specify the priority when you initially create the cluster. Keep this property in mind when you add a second reader or later DB instance to your cluster. For more information, see [Choosing the promotion tier for an Aurora Serverless v2 reader \(p. 1521\)](#).

For other ways that you can see the current capacity range for a cluster, see [Checking the capacity range for Aurora Serverless v2 \(p. 1513\)](#).

## Converting a provisioned writer or reader to Aurora Serverless v2

You can convert a provisioned DB instance to use Aurora Serverless v2. To do so, you follow the procedure in [Modify a DB instance in a DB cluster \(p. 249\)](#). The cluster must meet the requirements in [Requirements for Aurora Serverless v2 \(p. 1490\)](#). For example, Aurora Serverless v2 DB instances require that the cluster be running certain minimum engine versions.

Suppose that you are converting a running provisioned cluster to take advantage of Aurora Serverless v2. In that case, you can minimize downtime by converting a DB instance to Aurora Serverless v2 as the first step in the switchover process. For the full procedure, see [Switching from a provisioned cluster to Aurora Serverless v2 \(p. 1495\)](#).

If the DB instance that you convert is the first Aurora Serverless v2 DB instance in the cluster, you choose the capacity range for the cluster as part of the **Modify** operation. This capacity range applies to each Aurora Serverless v2 DB instance that you add to the cluster. The following image shows the page where you specify the minimum and maximum Aurora capacity units (ACUs).

## Instance configuration

The DB instance configuration options below are limited to those supported by the engine.

### DB instance class [Info](#)

- Serverless v2
- Memory optimized classes (includes r classes)
- Burstable classes (includes t classes)

### Capacity range [Info](#)

Database capacity is measured in Aurora Capacity Units (ACUs). 1 ACU provides 2 GiB of memory and 1 GiB of networking.

Minimum ACUs

4.5  (8 GiB)

0.5 to 128 in increments of 0.5

Maximum ACUs

16 (32 GiB)

0.5 to 128 in increments of 0.5

For details about the significance of the capacity range, see [Aurora Serverless v2 capacity \(p. 1486\)](#).

If the cluster already contains one or more Aurora Serverless v2 DB instances, you see the existing capacity range during the **Modify** operation. The following image shows an example of that information panel.

## Instance configuration

The DB instance configuration options below are limited to those supported by the engine that you selected.

### DB instance class [Info](#)

- Serverless v2
- Memory optimized classes (includes r classes)
- Burstable classes (includes t classes)

### Capacity range [Info](#)

Database capacity is measured in Aurora Capacity Units (ACUs). 1 ACU provides 2 GiB of memory and networking.

Minimum ACUs

2 ACUs (4 GiB)

Maximum ACUs

64 ACUs (128 GiB)

If you want to change the capacity range for the cluster after you add more Aurora Serverless v2 DB instances, follow the procedure in [Setting the Aurora Serverless v2 capacity range for a cluster \(p. 1510\)](#).

## Converting an Aurora Serverless v2 writer or reader to provisioned

You can convert an Aurora Serverless v2 DB instance to a provisioned DB instance. To do so, you follow the procedure in [Modify a DB instance in a DB cluster \(p. 249\)](#). Choose a DB instance class other than **Serverless**.

You might convert an Aurora Serverless v2 DB instance to provisioned if it needs a larger capacity than is available with the maximum Aurora capacity units (ACUs) of an Aurora Serverless v2 DB instance. For example, the largest db.r5 and db.r6g DB instance classes have a larger memory capacity than an Aurora Serverless v2 DB instance can scale to.

### Tip

Some of the older DB instance classes such as db.r3 and db.t2 aren't available for the Aurora versions that you use with Aurora Serverless v2. To see which DB instance classes you can use when converting an Aurora Serverless v2 DB instance to a provisioned one, see [Supported DB engines for DB instance classes \(p. 57\)](#).

If you are converting the writer DB instance of your cluster from Aurora Serverless v2 to provisioned, you can follow the procedure in [Switching from a provisioned cluster to Aurora Serverless v2 \(p. 1495\)](#) but in reverse. Switch one of the reader DB instances in the cluster from Aurora Serverless v2 to provisioned. Then perform a failover to make that provisioned DB instance into the writer.

Any capacity range that you previously specified for the cluster remains in place, even if all Aurora Serverless v2 DB instances are removed from the cluster. If you want to change the capacity range, you can modify the cluster, as explained in [Setting the Aurora Serverless v2 capacity range for a cluster \(p. 1510\)](#).

## Choosing the promotion tier for an Aurora Serverless v2 reader

For clusters containing multiple Aurora Serverless v2 DB instances or a mixture of provisioned and Aurora Serverless v2 DB instances, pay attention to the promotion tier setting for each Aurora Serverless v2 DB instance. This setting controls more behavior for Aurora Serverless v2 DB instances than for provisioned DB instances.

In the AWS Management Console, you specify this setting using the **Failover priority** choice under **Additional configuration** for the **Create database**, **Modify instance**, and **Add reader** pages. You see this property for existing DB instances in the optional **Priority tier** column on the **Databases** page. You can also see this property on the details page for a DB cluster or DB instance.

For provisioned DB instances, the choice of tier 0–15 determines only the order in which Aurora chooses which reader DB instance to promote to the writer during a failover operation. For Aurora Serverless v2 reader DB instances, the tier number also determines whether the DB instance scales up to match the capacity of the writer DB instance or scales independently based on its own workload. Aurora Serverless v2 reader DB instances in tier 0 or 1 are kept at a minimum capacity at least as high as the writer DB instance. That way, they are ready to take over from the writer DB instance in case of a failover. If the writer DB instance is a provisioned DB instance, Aurora estimates the equivalent Aurora Serverless v2 capacity. It uses that estimate as the minimum capacity for the Aurora Serverless v2 reader DB instance.

Aurora Serverless v2 reader DB instances in tiers 2–15 don't have the same constraint on their minimum capacity. When they are idle, they can scale down to the minimum Aurora capacity unit (ACU) value specified in the cluster's capacity range.

The following Linux AWS CLI example shows how to examine the promotion tiers of all the DB instances in your cluster. The final field includes a value of `True` for the writer DB instance and `False` for all the reader DB instances.

```
$ aws rds describe-db-clusters --db-cluster-identifier promotion-tier-demo \
--query 'DBClusters[*].DBClusterMembers[*]' \
[PromotionTier,DBInstanceIdentifier,IsClusterWriter]' \
--output text

1  instance-192  True
1  tier-01-4840  False
10 tier-10-7425  False
15 tier-15-6694  False
```

The following Linux AWS CLI example shows how to change the promotion tier of a specific DB instance in your cluster. The commands first modify the DB instance with a new promotion tier. Then they wait for the DB instance to become available again and confirm the new promotion tier for the DB instance.

```
$ aws rds modify-db-instance --db-instance-identifier instance-192 --promotion-tier 0
$ aws rds wait db-instance-available --db-instance-identifier instance-192
$ aws rds describe-db-instances --db-instance-identifier instance-192 \
--query '*[].[PromotionTier]' --output text
0
```

For more guidance about specifying promotion tiers for different use cases, see [Aurora Serverless v2 scaling \(p. 1487\)](#).

## Using TLS/SSL with Aurora Serverless v2

Aurora Serverless v2 can use the Transport Layer Security/Secure Sockets Layer (TLS/SSL) protocol to encrypt communications between clients and your Aurora Serverless v2 DB instances. It supports TLS/

SSL versions 1.0, 1.1, and 1.2. For general information about using TLS/SSL with Aurora, see [Using SSL/TLS with Aurora MySQL DB clusters \(p. 683\)](#).

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 v2 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 certificate authority (CA).	VERIFY_CA	verify-ca
Enforce TLS/SSL, verify the CA, and verify the CA hostname.	VERIFY_IDENTITY	verify-full

Aurora Serverless v2 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, do the following.

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 \(p. 207\)](#).

## Supported cipher suites for connections to Aurora Serverless v2 DB clusters

By using configurable cipher suites, you can have more control over the security of your database connections. You can specify a list of cipher suites that you want to allow to secure client TLS/SSL connections to your database. With configurable cipher suites, you can control the connection encryption that your database server accepts. Doing this prevents the use of ciphers that aren't secure or that are no longer used.

Aurora Serverless v2 DB clusters that are based on Aurora MySQL support the same cipher suites as Aurora MySQL provisioned DB clusters. For information about these cipher suites, see [Configuring cipher suites for connections to Aurora MySQL DB clusters \(p. 685\)](#).

Aurora Serverless v2 DB clusters that are based on Aurora PostgreSQL support the same cipher suites as Aurora PostgreSQL provisioned DB clusters. For information about these cipher suites, see [Configuring cipher suites for connections to Aurora PostgreSQL DB clusters \(p. 1031\)](#).

## Viewing Aurora Serverless v2 writers and readers

You can view the details of Aurora Serverless v2 DB instances in the same way that you do for provisioned DB instances. To do so, follow the general procedure from [Viewing an Amazon Aurora DB cluster \(p. 433\)](#). A cluster might contain all Aurora Serverless v2 DB instances, all provisioned DB instances, or some of each.

After you create one or more Aurora Serverless v2 DB instances, you can view which DB instances are type **Serverless** and which are type **Instance**. You can also view the minimum and maximum Aurora capacity units (ACUs) that the Aurora Serverless v2 DB instance can use. Each ACU is a combination of processing (CPU) and memory (RAM) capacity. This capacity range applies to each Aurora Serverless v2 DB instance in the cluster. For the procedure to check the capacity range of a cluster or any Aurora Serverless v2 DB instance in the cluster, see [Checking the capacity range for Aurora Serverless v2 \(p. 1513\)](#).

In the AWS Management Console, Aurora Serverless v2 DB instances are marked under the **Size** column in the **Databases** page. Provisioned DB instances show the name of a DB instance class such as r6g.xlarge. The Aurora Serverless DB instances show **Serverless** for the DB instance class, along with the DB instance's minimum and maximum capacity. For example, you might see **Serverless v2 (4–64 ACUs)** or **Serverless v2 (1–40 ACUs)**.

You can find the same information on the **Configuration** tab for each Aurora Serverless v2 DB instance in the console. For example, you might see an **Instance type** section such as the following. Here, the **Instance type** value is **Serverless v2**, the **Minimum capacity** value is **2 ACUs (4 GiB)**, and the **Maximum capacity** value is **64 ACUs (128 GiB)**.

---

### Instance configuration

Instance type

Serverless v2

Minimum capacity

2 ACUs (4 GiB)

Maximum capacity

64 ACUs (128 GiB)

You can monitor the capacity of each Aurora Serverless v2 DB instance over time. That way, you can check the minimum, maximum, and average ACUs consumed by each DB instance. You can also check how close the DB instance came to its minimum or maximum capacity. To see such details in the AWS Management Console, examine the graphs of Amazon CloudWatch metrics on the **Monitoring** tab for the DB instance. For information about the metrics to watch and how to interpret them, see [Important Amazon CloudWatch metrics for Aurora Serverless v2 \(p. 1539\)](#).

## Logging for Aurora Serverless v2

To turn on database logging, you specify the logs to enable using configuration parameters in your custom parameter group.

For Aurora MySQL, you can enable the following logs.

Aurora MySQL	Description
<code>general_log</code>	Creates the general log. Set to 1 to turn on. Default is off (0).
<code>log_queries_not_using_indexes</code>	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.
<code>long_query_time</code>	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).
<code>server_audit_events</code>	The list of events to capture in the logs. Supported values are CONNECT, QUERY, QUERY_DCL, QUERY_DDL, QUERY_DML, and TABLE.
<code>server_audit_logging</code>	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 <code>server_audit_events</code> parameter.
<code>slow_query_log</code>	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 \(p. 823\)](#).

For Aurora PostgreSQL, you can enable the following logs on your Aurora Serverless v2 DB instances.

Aurora PostgreSQL	Description
<code>log_connections</code>	Logs each successful connection.
<code>log_disconnections</code>	Logs end of a session including duration.
<code>log_lock_waits</code>	Default is 0 (off). Set to 1 to log lock waits.
<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 debug5, debug4, debug3, debug2, debug1, info, notice, warning, error, log, fatal, panic. To log performance data to the <code>postgres</code> log, set the value to debug1.
<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 none, ddl, mod, and all. Default is none.

## Topics

- [Logging with Amazon CloudWatch \(p. 1525\)](#)
- [Viewing Aurora Serverless v2 logs in Amazon CloudWatch \(p. 1525\)](#)
- [Monitoring capacity with Amazon CloudWatch \(p. 1526\)](#)

## Logging with Amazon CloudWatch

After you use the procedure in [Logging for Aurora Serverless v2 \(p. 1523\)](#) to choose which database logs to turn on, you can choose which logs to upload ("publish") to Amazon CloudWatch.

You can use Amazon CloudWatch to analyze log data, create alarms, and view metrics. By default, error logs for Aurora Serverless v2 are enabled and automatically uploaded to CloudWatch. You can also upload other logs from Aurora Serverless v2 DB instances to CloudWatch.

Then you choose which of those logs to upload to CloudWatch, by using the **Log exports** settings in the AWS Management Console or the `--enable-cloudwatch-logs-exports` option in the AWS CLI.

You can choose which of your Aurora Serverless v2 logs to upload to CloudWatch.

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 and Aurora Serverless, 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 DB cluster, so that it's available to choose when you create a database on the console.

For more information, see [Using Advanced Auditing with an Amazon Aurora MySQL DB cluster \(p. 823\)](#).

After you apply your modified DB cluster parameter group to your Aurora Serverless v2 DB cluster, you can view the logs in CloudWatch.

### To view Aurora Serverless v2 logs

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. Choose **Log groups**.
3. Choose your Aurora Serverless v2 DB cluster log from the list. For error logs, the naming pattern is as follows.

```
/aws/rds/cluster/cluster_name/log_type
```

For more information on viewing details on your logs, see [Monitoring log events in Amazon CloudWatch \(p. 927\)](#).

## Viewing Aurora Serverless v2 logs in Amazon CloudWatch

After you use the procedure in [Logging for Aurora Serverless v2 \(p. 1523\)](#) to choose which database logs to turn on, you can view the contents of the logs.

For more information on using CloudWatch with Aurora MySQL and Aurora PostgreSQL logs, see [Monitoring log events in Amazon CloudWatch \(p. 927\)](#) and [Publishing Aurora PostgreSQL logs to Amazon CloudWatch Logs \(p. 1265\)](#).

### To view logs for your Aurora Serverless v2 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 v2 DB cluster log from the list. For error logs, the naming pattern is as follows.

```
/aws/rds/cluster/cluster-name/error
```

## Monitoring capacity with Amazon CloudWatch

With Aurora Serverless v2, you can use CloudWatch to monitor the capacity and utilization of all the Aurora Serverless v2 DB instances in your cluster. You can view instance-level metrics to check the capacity of each Aurora Serverless v2 DB instance as it scales up and down. You can also compare the capacity-related metrics to other metrics to see how changes in workloads affect resource consumption. For example, you can compare `ServerlessDatabaseCapacity` to `DatabaseUsedMemory`, `DatabaseConnections`, and `DMLThroughput` to assess how your DB cluster is responding during operations. For details about the capacity-related metrics that apply to Aurora Serverless v2, see [Important Amazon CloudWatch metrics for Aurora Serverless v2 \(p. 1539\)](#).

### To monitor your Aurora Serverless v2 DB cluster's capacity

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. Choose **Metrics**. All available metrics appear as cards in the console, grouped by service name.
3. Choose **RDS**.
4. (Optional) Use the **Search** box to find the metrics that are especially important for Aurora Serverless v2: `ServerlessDatabaseCapacity`, `ACUUtilization`, `CPUUtilization`, and `FreeableMemory`.

We recommend that you set up a CloudWatch dashboard to monitor your Aurora Serverless v2 DB cluster capacity using the capacity-related metrics. To learn how, see [Building dashboards with CloudWatch](#).

To learn more about using Amazon CloudWatch with Amazon Aurora, see [Publishing Amazon Aurora MySQL logs to Amazon CloudWatch Logs \(p. 924\)](#).

## Performance and scaling for Aurora Serverless v2

The following procedures and examples show how you can set the capacity range for Aurora Serverless v2 clusters and their associated DB instances. You can also use procedures following to monitor how busy your DB instances are. Then you can use your findings to determine if you need to adjust the capacity range upward or downward.

Before you use these procedures, make sure that you are familiar with how Aurora Serverless v2 scaling works. The scaling mechanism is different than in Aurora Serverless v1. For details, see [Aurora Serverless v2 scaling \(p. 1487\)](#).

### Contents

- [Choosing the Aurora Serverless v2 capacity range for an Aurora cluster \(p. 1527\)](#)
  - [Choosing the minimum Aurora Serverless v2 capacity setting for a cluster \(p. 1527\)](#)
  - [Choosing the maximum Aurora Serverless v2 capacity setting for a cluster \(p. 1528\)](#)
  - [Example: Change the Aurora Serverless v2 capacity range of an Aurora MySQL cluster \(p. 1530\)](#)
  - [Example: Change the Aurora Serverless v2 capacity range of an Aurora PostgreSQL cluster \(p. 1532\)](#)

- [Working with parameter groups for Aurora Serverless v2 \(p. 1535\)](#)
  - [Default parameter values \(p. 1536\)](#)
  - [Parameters that Aurora adjusts as Aurora Serverless v2 scales up and down \(p. 1537\)](#)
  - [Parameters that Aurora computes based on Aurora Serverless v2 maximum capacity \(p. 1537\)](#)
- [Avoiding out-of-memory errors \(p. 1538\)](#)
- [Important Amazon CloudWatch metrics for Aurora Serverless v2 \(p. 1539\)](#)
  - [How Aurora Serverless v2 metrics apply to your AWS bill \(p. 1540\)](#)
  - [Examples of CloudWatch commands for Aurora Serverless v2 metrics \(p. 1540\)](#)
- [Monitoring Aurora Serverless v2 performance with Performance Insights \(p. 1542\)](#)

## Choosing the Aurora Serverless v2 capacity range for an Aurora cluster

With Aurora Serverless v2 DB instances, you set the capacity range that applies to all the DB instances in your DB cluster at the same time that you add the first Aurora Serverless v2 DB instance to the DB cluster. For the procedure to do so, see [Setting the Aurora Serverless v2 capacity range for a cluster \(p. 1510\)](#).

You can also change the capacity range for an existing cluster. The following sections discuss in more detail how to choose appropriate minimum and maximum values and what happens when you make a change to the capacity range. For example, changing the capacity range can modify the default values of some configuration parameters. Applying all the parameter changes can require rebooting each Aurora Serverless v2 DB instance.

### Topics

- [Choosing the minimum Aurora Serverless v2 capacity setting for a cluster \(p. 1527\)](#)
- [Choosing the maximum Aurora Serverless v2 capacity setting for a cluster \(p. 1528\)](#)
- [Example: Change the Aurora Serverless v2 capacity range of an Aurora MySQL cluster \(p. 1530\)](#)
- [Example: Change the Aurora Serverless v2 capacity range of an Aurora PostgreSQL cluster \(p. 1532\)](#)

## Choosing the minimum Aurora Serverless v2 capacity setting for a cluster

It's tempting to always choose 0.5 for the minimum Aurora Serverless v2 capacity setting. That value allows the DB instance to scale down the most when it's completely idle. However, depending on how you use that cluster and the other settings that you configure, a different value might be the most effective. Consider the following factors when choosing the minimum capacity setting:

- The scaling rate for an Aurora Serverless v2 DB instance depends on its current capacity. The higher the current capacity, the faster it can scale up. If you need the DB instance to quickly scale up to a very high capacity, consider setting the minimum capacity to a value where the scaling rate meets your requirement.
- If you typically modify the DB instance class of your DB instances in anticipation of especially high or low workload, you can use that experience to make a rough estimate of the equivalent Aurora Serverless v2 capacity range. To determine the memory size to use in times of low traffic, consult [Hardware specifications for DB instance classes for Aurora \(p. 64\)](#).

For example, suppose that you use the db.r6g.xlarge DB instance class when your cluster has a low workload. That DB instance class has 32 GiB of memory. Thus, you can specify a minimum Aurora capacity unit (ACU) setting of 16 to set up an Aurora Serverless v2 DB instance that can scale down

to approximately that same capacity. That's because each ACU corresponds to approximately 2 GiB of memory. You might specify a somewhat lower value to let the DB instance scale down further in case your db.r6g.xlarge DB instance was sometimes underutilized.

- If your application works most efficiently when the DB instances have a certain amount of data in the buffer cache, consider specifying a minimum ACU setting where the memory is large enough to hold the frequently accessed data. Otherwise, some data is evicted from the buffer cache when the Aurora Serverless v2 DB instances scale down to a lower memory size. Then when the DB instances scale back up, the information is read back into the buffer cache over time. If the amount of I/O to bring the data back into the buffer cache is substantial, it might be more effective to choose a higher minimum ACU value.
- If your Aurora Serverless v2 DB instances run most of the time at a particular capacity, consider specifying a minimum capacity setting that's lower than that baseline, but not too much lower. Aurora Serverless v2 DB instances can most effectively estimate how much and how fast to scale up when the current capacity isn't drastically lower than the required capacity.
- If your provisioned workload has memory requirements that are too high for small DB instance classes such as T3 or T4g, choose a minimum ACU setting that provides memory comparable to an R5 or R6g DB instance.

In particular, we recommend the following minimum capacity for use with the specified features (these recommendations are subject to change):

- **Performance Insights** – 2 ACUs.
- In some cases, your cluster might contain Aurora Serverless v2 reader DB instances that scale independently from the writer. If so, choose a minimum capacity setting that's high enough that when the writer DB instance is busy with a write-intensive workload, the reader DB instances can apply the changes from the writer without falling behind. If you observe replica lag in readers that are in promotion tiers 2–15, consider increasing the minimum capacity setting for your cluster. For details on choosing whether reader DB instances scale along with the writer or independently, see [Choosing the promotion tier for an Aurora Serverless v2 reader \(p. 1521\)](#).
- If you have a mixed-configuration cluster with a provisioned writer and Aurora Serverless v2 readers, the readers can't scale along with the writer. In that case, setting a low minimum capacity can result in excessive replication lag. That's because the readers might not have enough capacity to apply changes from the writer when the database is busy. When your cluster uses a provisioned writer, set the minimum capacity to a value that represents a comparable amount of memory and CPU to the writer.
- For Aurora PostgreSQL, when you specify a minimum Aurora Serverless v2 capacity of 0.5, the `max_connections` setting is permanently capped at 2000. If you intend to use the Aurora PostgreSQL cluster for a high-connection workload, consider using a minimum ACU setting of 1 or higher. For details about how Aurora Serverless v2 handles the `max_connections` configuration parameter, see [Parameters that Aurora computes based on Aurora Serverless v2 maximum capacity \(p. 1537\)](#).
- The time it takes for an Aurora Serverless v2 DB instance to scale from its minimum capacity to its maximum capacity depends on the difference between its minimum and maximum ACU values. When the current capacity of the DB instance is large, Aurora Serverless v2 scales up in larger increments than when the DB instance starts from a small capacity. Thus, if you specify a relatively large maximum capacity and the DB instance spends most of its time near that capacity, consider increasing the minimum ACU setting. That way, an idle DB instance can scale back up to maximum capacity more quickly.

## Choosing the maximum Aurora Serverless v2 capacity setting for a cluster

It's tempting to always choose some high value for the maximum Aurora Serverless v2 capacity setting. A large maximum capacity allows the DB instance to scale up the most when it's running an intensive

workload. A low value avoids the possibility of unexpected charges. Depending on how you use that cluster and the other settings that you configure, the most effective value might be higher or lower than you originally thought. Consider the following factors when choosing the maximum capacity setting:

- The maximum capacity must be at least as high as the minimum capacity. You can set the minimum and maximum capacity to be identical. However, in that case the capacity never scales up or down. Thus, using identical values for minimum and maximum capacity isn't appropriate outside of testing situations.
- The maximum capacity must be higher than 0.5 ACUs. You can set the minimum and maximum capacity to be the same in most cases. However, you can't specify 0.5 for both the minimum and maximum. Use a value of 1 or higher for the maximum capacity.
- If you typically modify the DB instance class of your DB instances in anticipation of especially high or low workload, you can use that experience to estimate the equivalent Aurora Serverless v2 capacity range. To determine the memory size to use in times of high traffic, consult [Hardware specifications for DB instance classes for Aurora \(p. 64\)](#).

For example, suppose that you use the db.r6g.4xlarge DB instance class when your cluster has a high workload. That DB instance class has 128 GiB of memory. Thus, you can specify a maximum ACU setting of 64 to set up an Aurora Serverless v2 DB instance that can scale up to approximately that same capacity. That's because each ACU corresponds to approximately 2 GiB of memory. You might specify a somewhat higher value to let the DB instance scale up farther in case your db.r6g.4xlarge DB instance sometimes doesn't have enough capacity to handle the workload effectively.

- If you have a budgetary cap on your database usage, choose a value that stays within that cap even if all your Aurora Serverless v2 DB instances run at maximum capacity all the time. Remember that when you have  $n$  Aurora Serverless v2 DB instances in your cluster, the theoretical maximum Aurora Serverless v2 capacity that the cluster can consume at any moment is  $n$  times the maximum ACU setting for the cluster. (The actual amount consumed might be less, for example if some readers scale independently from the writer.)
- If you make use of Aurora Serverless v2 reader DB instances to offload some of the read-only workload from the writer DB instance, you might be able to choose a lower maximum capacity setting. You do this to reflect that each reader DB instance doesn't need to scale as high as if the cluster contains only a single DB instance.
- Suppose that you want to protect against excessive usage due to misconfigured database parameters or inefficient queries in your application. In that case, you might avoid accidental overuse by choosing a maximum capacity setting that's lower than the absolute highest that you could set.
- If spikes due to real user activity are rare but do happen, you can take those occasions into account when choosing the maximum capacity setting. If the priority is for the application to keep running with full performance and scalability, you can specify a maximum capacity setting that's higher than you observe in normal usage. If it's OK for the application to run with reduced throughput during very extreme spikes in activity, you can choose a slightly lower maximum capacity setting. Make sure that you choose a setting that still has enough memory and CPU resources to keep the application running.
- If you turn on settings in your cluster that increase the memory usage for each DB instance, take that memory into account when deciding on the maximum ACU value. Such settings include those for Performance Insights, Aurora MySQL parallel queries, Aurora MySQL performance schema, and Aurora MySQL binary log replication. Make sure that the maximum ACU value allows the Aurora Serverless v2 DB instances to scale up enough to handle the workload when those feature are being used. For information about troubleshooting problems caused by the combination of a low maximum ACU setting and Aurora features that impose memory overhead, see [Avoiding out-of-memory errors \(p. 1538\)](#).

## Example: Change the Aurora Serverless v2 capacity range of an Aurora MySQL cluster

The following AWS CLI example shows how to update the ACU range for Aurora Serverless v2 DB instances in an existing Aurora MySQL cluster. Initially, the ACU range for the cluster 8–32.

```
$ aws rds describe-db-clusters --db-cluster-identifier serverless-v2-cluster \
--query 'DBClusters[*].ServerlessV2ScalingConfiguration|[0]'

{
    "MinCapacity": 8.0,
    "MaxCapacity": 32.0
}
```

The following capacity-related settings apply to the DB instance at this point. The DB instance is idle and scaled down to 8 ACUs. To represent the size of the buffer pool in easily readable units, we divide it by 2 to the power of 30, yielding a measurement in gibibytes (GiB). That's because memory-related measurements for Aurora use units based on powers of 2, not powers of 10.

```
mysql> select @@max_connections;
+-----+
| @@max_connections |
+-----+
|          3000 |
+-----+
1 row in set (0.00 sec)

mysql> select @@innodb_buffer_pool_size;
+-----+
| @@innodb_buffer_pool_size |
+-----+
|      9294577664 |
+-----+
1 row in set (0.00 sec)

mysql> select @@innodb_buffer_pool_size / pow(2,30) as gibibytes;
+-----+
| gibibytes |
+-----+
|     8.65625 |
+-----+
1 row in set (0.00 sec)
```

Next, we change the capacity range for the cluster. After the `modify-db-cluster` command finishes, the ACU range for the cluster is 12.5–80.

```
$ aws rds modify-db-cluster --db-cluster-identifier serverless-v2-cluster \
--serverless-v2-scaling-configuration MinCapacity=12.5,MaxCapacity=80

$ aws rds describe-db-clusters --db-cluster-identifier serverless-v2-cluster \
--query 'DBClusters[*].ServerlessV2ScalingConfiguration|[0]'

{
    "MinCapacity": 12.5,
    "MaxCapacity": 80.0
}
```

Changing the capacity range caused changes to the default values of some configuration parameters. Aurora can apply some of those new defaults immediately. However, some of the parameter changes take effect only after a reboot. The pending-reboot status indicates that a reboot is needed to apply some parameter changes.

**Tip**

You can reboot the DB instances yourself to apply these parameter changes. Or you can wait for Aurora to do the reboot and apply the parameter changes during your next scheduled maintenance window.

```
$ aws rds describe-db-clusters --db-cluster-identifier serverless-v2-cluster \
--query '*[].[{DBClusterMembers:DBClusterMembers[*]}, \
{DBInstanceIdentifier:DBInstanceIdentifier,DBClusterParameterGroupStatus:DBClusterParameterGroupStatus}][0]' \
{
    "DBClusterMembers": [
        {
            "DBInstanceIdentifier": "serverless-v2-instance-1",
            "DBClusterParameterGroupStatus": "pending-reboot"
        }
    ]
}
```

The following example shows how the `innodb_buffer_pool_size` parameter is already adjusted based on the current capacity of the DB instance. At this point, the cluster is idle and the DB instance `serverless-v2-instance-1` is consuming 12.5 ACUs. The `max_connections` parameter still reflects the value from the former capacity range. Resetting that value requires rebooting the DB instance.

```
mysql> select @@max_connections;
+-----+
| @@max_connections |
+-----+
|          3000 |
+-----+
1 row in set (0.00 sec)

mysql> select @@innodb_buffer_pool_size;
+-----+
| @@innodb_buffer_pool_size |
+-----+
|      15572402176 |
+-----+
1 row in set (0.00 sec)

mysql> select @@innodb_buffer_pool_size / pow(2,30) as gibibytes;
+-----+
| gibibytes   |
+-----+
| 14.5029296875 |
+-----+
1 row in set (0.00 sec)
```

Now we reboot the DB instance and wait for it to become available again.

```
$ aws rds reboot-db-instance --db-instance-identifier serverless-v2-instance-1 \
{
    "DBInstanceIdentifier": "serverless-v2-instance-1",
    "DBInstanceState": "rebooting"
}
$ aws rds wait db-instance-available --db-instance-identifier serverless-v2-instance-1
```

Now that the DB instance is rebooted, the `pending-reboot` status is cleared. The value `in-sync` confirms that Aurora has applied all the pending parameter changes.

```
$ aws rds describe-db-clusters --db-cluster-identifier serverless-v2-cluster \
```

```
--query '*[].[DBClusterMembers:DBClusterMembers[*].{DBInstanceIdentifier:DBInstanceIdentifier,DBClusterParameterGroupStatus:DBClusterParameterGroupStatus}[0]'.
{
    "DBClusterMembers": [
        {
            "DBInstanceIdentifier": "serverless-v2-instance-1",
            "DBClusterParameterGroupStatus": "in-sync"
        }
    ]
}
```

The following example checks the same parameters as before the reboot. The `innodb_buffer_pool_size` parameter has increased to its final size for an idle DB instance. The `max_connections` parameter has increased to reflect a value derived from the maximum ACU value. The formula that Aurora uses for `max_connections` causes an increase of 1,000 when the memory size doubles.

```
mysql> select @@innodb_buffer_pool_size;
+-----+
| @@innodb_buffer_pool_size |
+-----+
| 16139681792 |
+-----+
1 row in set (0.00 sec)

mysql> select @@innodb_buffer_pool_size / pow(2,30) as gibibytes;
+-----+
| gibibytes |
+-----+
| 15.03125 |
+-----+
1 row in set (0.00 sec)

mysql> select @@max_connections;
+-----+
| @@max_connections |
+-----+
| 4000 |
+-----+
1 row in set (0.00 sec)
```

In the following example, we used the same procedure as before to set the capacity range at 0.5–128 ACUs. We rebooted the DB instance to apply any resulting changes to static parameters. Now the idle DB instance has a buffer cache size that's less than 1 GiB, so we measure it in mebibytes (MiB). The `max_connections` value of 5000 is derived from the memory size of the maximum capacity setting.

```
mysql> select @@innodb_buffer_pool_size / pow(2,20) as mebibytes, @@max_connections;
+-----+
| mebibytes | @@max_connections |
+-----+
| 672 | 5000 |
+-----+
1 row in set (0.00 sec)
```

## Example: Change the Aurora Serverless v2 capacity range of an Aurora PostgreSQL cluster

The following CLI example shows how to update the ACU range for Aurora Serverless v2 DB instances in an existing Aurora PostgreSQL cluster. Initially, the ACU range for the cluster is 8–32.

```
$ aws rds describe-db-clusters --db-cluster-identifier serverless-v2-cluster \
--query 'DBClusters[*].ServerlessV2ScalingConfiguration|[0]'
{
    "MinCapacity": 8.0,
    "MaxCapacity": 32.0
}
```

The following capacity-related settings apply to the DB instance at this point. The DB instance is idle and scaled down to 8 ACUs.

```
postgres=> show shared_buffers;
shared_buffers
-----
1327104
(1 row)

postgres=> show max_connections;
max_connections
-----
2000
(1 row)
```

Next, we change the capacity range for the cluster. After the `modify-db-cluster` command finishes, the ACU range for the cluster is 12.5–80.

```
$ aws rds modify-db-cluster --db-cluster-identifier serverless-v2-cluster \
--serverless-v2-scaling-configuration MinCapacity=12.5,MaxCapacity=80

$ aws rds describe-db-clusters --db-cluster-identifier serverless-v2-cluster \
--query 'DBClusters[*].ServerlessV2ScalingConfiguration|[0]'
{
    "MinCapacity": 12.5,
    "MaxCapacity": 80.0
}
```

Changing the capacity range causes changes to the default values of some configuration parameters. Aurora can apply some of those new defaults immediately. However, some of the parameter changes take effect only after a reboot. The `pending-reboot` status indicates that you need a reboot to apply some parameter changes.

**Tip**

You can reboot the DB instances yourself to apply these parameter changes. Or you can wait for Aurora to do the reboot and apply the parameter changes during your next scheduled maintenance window.

```
$ aws rds describe-db-clusters --db-cluster-identifier serverless-v2-cluster \
--query '*[].[DBClusterMembers:DBClusterMembers[*].
{DBInstanceIdentifier:DBInstanceIdentifier,DBClusterParameterGroupStatus:DBClusterParameterGroupStatus}
[0]]'
{
    "DBClusterMembers": [
        {
            "DBInstanceIdentifier": "serverless-v2-instance-1",
            "DBClusterParameterGroupStatus": "pending-reboot"
        }
    ]
}
```

The following example shows how the `shared_buffers` parameter is already adjusted based on the current capacity of the DB instance. At this point, the cluster is idle and the DB instance `serverless-`

v2-instance-1 is consuming 12.5 ACUs. The `max_connections` parameter still reflects the value from the former capacity range. Resetting that value requires rebooting the DB instance.

```
postgres=> show shared_buffers;
shared_buffers
-----
344064
(1 row)

postgres=> show max_connections;
max_connections
-----
1034
(1 row)

postgres=> select name as parameter_name, setting, unit,
  (((setting::BIGINT)*8)/1024)::BIGINT as "size_MiB",
  postgres->   (((setting::BIGINT)*8)/1024/1024)::BIGINT as "size_GiB",
  pg_size.pretty(((setting::BIGINT)*8)*1024)::BIGINT)
postgres-> from pg_settings where name in ('shared_buffers');
parameter_name | setting | unit | size_MiB | size_GiB | pg_size.pretty
-----+-----+-----+-----+-----+
shared_buffers | 32768 | 8kB | 256 | 0 | 256 MB
(1 row)
```

Now we reboot the DB instance and wait for it to become available again.

```
$ aws rds reboot-db-instance --db-instance-identifier serverless-v2-instance-1
{
  "DBInstanceIdentifier": "serverless-v2-instance-1",
  "DBInstanceState": "rebooting"
}
$ aws rds wait db-instance-available --db-instance-identifier serverless-v2-instance-1
```

Now that the DB instance is rebooted, the pending-reboot status is cleared. The value `in-sync` confirms that Aurora has applied all the pending parameter changes.

```
$ aws rds describe-db-clusters --db-cluster-identifier serverless-v2-cluster \
--query '*[].[DBClusterMembers:DBClusterMembers[*].
{DBInstanceIdentifier:DBInstanceIdentifier,DBClusterParameterGroupStatus:DBClusterParameterGroupStatus}[0]'.
{
  "DBClusterMembers": [
    {
      "DBInstanceIdentifier": "serverless-v2-instance-1",
      "DBClusterParameterGroupStatus": "in-sync"
    }
  ]
}
```

The following example checks the same parameters as before the reboot. The `shared_buffers` parameter has increased to its final size for an idle DB instance. The `max_connections` parameter has increased to reflect a value derived from the maximum ACU value.

```
postgres=> show shared_buffers;
shared_buffers
-----
1425408
```

```
(1 row)

postgres=> show max_connections;
max_connections
-----
1034
(1 row)

postgres=> select name as parameter_name, setting, unit,
  pg_size.pretty(((setting::BIGINT)*8)*1024)::BIGINT)
postgres->   from pg_settings where name in ('shared_buffers');
  parameter_name | setting | unit | pg_size.pretty
-----+-----+-----+
  shared_buffers | 1425408 | 8kB  | 11 GB
(1 row)
```

In the following example, we used the same procedure as before to set the capacity range from 0.5 to 128 ACUs. We rebooted the DB instance to apply any resulting changes to static parameters. The `max_connections` value of 2000 isn't derived from the maximum ACU setting. When the minimum ACU setting is 0.5, PostgreSQL-compatible Aurora Serverless v2 DB instances use a default `max_connections` value of 2000 regardless of the maximum ACU value.

```
postgres=> show shared_buffers;
shared_buffers
-----
2228224
(1 row)

postgres=> show max_connections;
max_connections
-----
1034
(1 row)

postgres=> select name as parameter_name, setting, unit,
  pg_size.pretty(((setting::BIGINT)*8)*1024)::BIGINT) from pg_settings where name in
  ('shared_buffers');
  parameter_name | setting | unit | pg_size.pretty
-----+-----+-----+
  shared_buffers | 2228224 | 8kB  | 17 GB
(1 row)
```

## Working with parameter groups for Aurora Serverless v2

When you create your Aurora Serverless v2 DB cluster, you choose a specific Aurora DB engine and an associated DB cluster parameter group. If you aren't familiar with how Aurora uses parameter groups to apply configuration settings consistently across clusters, see [Working with parameter groups \(p. 215\)](#). All of those procedures for creating, modifying, applying, and other actions for parameter groups apply to Aurora Serverless v2.

The parameter group feature works generally the same between provisioned clusters and clusters containing Aurora Serverless v2 DB instances:

- The default parameter values for all DB instances in the cluster are defined by the cluster parameter group.
- You can override some parameters for specific DB instances by specifying a custom DB parameter group for those DB instances. You might do so during debugging or performance tuning for specific DB instances. For example, suppose that you have a cluster containing some Aurora Serverless v2

DB instances and some provisioned DB instances. In this case, you might specify some different parameters for the provisioned DB instances by using a custom DB parameter group.

- For Aurora Serverless v2, you can use all the parameters that have the value `provisioned` in the `SupportedEngineModes` attribute in the parameter group. In Aurora Serverless v1, you can only use the subset of parameters that have `serverless` in the `SupportedEngineModes` attribute.

### Topics

- [Default parameter values \(p. 1536\)](#)
- [Parameters that Aurora adjusts as Aurora Serverless v2 scales up and down \(p. 1537\)](#)
- [Parameters that Aurora computes based on Aurora Serverless v2 maximum capacity \(p. 1537\)](#)

## Default parameter values

The crucial difference between provisioned DB instances and Aurora Serverless v2 DB instances is that Aurora overrides any custom parameter values for certain parameters that are related to DB instance capacity. The custom parameter values still apply to any provisioned DB instances in your cluster. For more details about how Aurora Serverless v2 DB instances interpret the parameters from Aurora parameter groups, see [Configuration parameters for Aurora clusters \(p. 1489\)](#). For the specific parameters that Aurora Serverless v2 overrides, see [Parameters that Aurora adjusts as Aurora Serverless v2 scales up and down \(p. 1537\)](#) and [Parameters that Aurora computes based on Aurora Serverless v2 maximum capacity \(p. 1537\)](#).

You can get a list of default values for the default parameter groups for the various Aurora DB engines by using the `describe-db-cluster-parameters` CLI command and querying the AWS Region. The following are values that you can use for the `--db-parameter-group-family` and `-db-parameter-group-name` options for engine versions that are compatible with Aurora Serverless v2.

Database engine and version	Parameter group family	Default parameter group name
Aurora MySQL version 3	<code>aurora-mysql8.0</code>	<code>default.aurora-mysql8.0</code>
Aurora PostgreSQL version 13.x	<code>aurora-postgresql13</code>	<code>default.aurora-postgresql13</code>

The following example gets a list of parameters from the default DB cluster group for Aurora MySQL version 3 and Aurora PostgreSQL 13. Those are the Aurora MySQL and Aurora PostgreSQL versions that you use with Aurora Serverless v2.

For Linux, macOS, or Unix:

```
aws rds describe-db-cluster-parameters \
--db-cluster-parameter-group-name default.aurora-mysql8.0 \
--query 'Parameters[*]' \
{ParameterName:ParameterName,SupportedEngineModes:SupportedEngineModes} |
[?contains(SupportedEngineModes, `provisioned`) == `true`] | [*].[ParameterName]` \
--output text

aws rds describe-db-cluster-parameters \
--db-cluster-parameter-group-name default.aurora-postgresql13 \
--query 'Parameters[*]' \
{ParameterName:ParameterName,SupportedEngineModes:SupportedEngineModes} |
[?contains(SupportedEngineModes, `provisioned`) == `true`] | [*].[ParameterName]` \
--output text
```

For Windows:

```
aws rds describe-db-cluster-parameters ^
--db-cluster-parameter-group-name default.aurora-mysql8.0 ^
--query 'Parameters[*].{ParameterName:ParameterName,SupportedEngineModes:SupportedEngineModes} | 
[?contains(SupportedEngineModes, `provisioned` ) == `true`] | [*].[ParameterName]' ^
--output text

aws rds describe-db-cluster-parameters ^
--db-cluster-parameter-group-name default.aurora-postgresql13 ^
--query 'Parameters[*].{ParameterName:ParameterName,SupportedEngineModes:SupportedEngineModes} | 
[?contains(SupportedEngineModes, `provisioned` ) == `true`] | [*].[ParameterName]' ^
--output text
```

## Parameters that Aurora adjusts as Aurora Serverless v2 scales up and down

During autoscaling, Aurora Serverless v2 needs to be able to change parameters for each DB instance to work best for the increased or decreased capacity. Thus, you can't override some parameters related to capacity. For some parameters that you can override, avoid hardcoding fixed values. The following considerations apply to these settings that are related to capacity.

For Aurora MySQL, Aurora Serverless v2 resizes some parameters dynamically during scaling. For the following parameters, Aurora Serverless v2 doesn't use any custom parameter values that you specify:

- `innodb_buffer_pool_size`
- `innodb_purge_threads`
- `table_definition_cache`
- `table_open_cache`

For Aurora PostgreSQL, Aurora Serverless v2 resizes the following parameters parameter dynamically during scaling. For the following parameters, Aurora Serverless v2 doesn't use any custom parameter values that you specify:

- `shared_buffers`

For all parameters other than those listed here and in [Parameters that Aurora adjusts as Aurora Serverless v2 scales up and down \(p. 1537\)](#), Aurora Serverless v2 DB instances work the same as provisioned DB instances. The default parameter value is inherited from the cluster parameter group. You can modify the default for the whole cluster by using a custom cluster parameter group. Or you can modify the default for certain DB instances by using a custom DB parameter group. Dynamic parameters are updated immediately. Changes to static parameters only take effect after you reboot the DB instance.

## Parameters that Aurora computes based on Aurora Serverless v2 maximum capacity

For both Aurora MySQL and Aurora PostgreSQL, Aurora Serverless v2 DB instances hold the `max_connections` parameter constant so that connections aren't dropped when the DB instance scales down. The default value for this parameter is derived from a formula that refers to the memory size of the DB instance. For details about the formula and the default values for provisioned DB instance classes, see [Maximum connections to an Aurora MySQL DB instance \(p. 722\)](#) and [Maximum connections to an Aurora PostgreSQL DB instance \(p. 1144\)](#).

When Aurora Serverless v2 evaluates the formula, it uses the memory size based on the maximum Aurora capacity units (ACUs) for the DB instance, not the current ACU value. If you change the default

value, we recommend using a variation of the formula instead of specifying a constant value. That way, Aurora Serverless v2 can use a setting that's sized appropriately based on the maximum capacity.

When you specify a minimum capacity of 0.5 ACUs, PostgreSQL-compatible Aurora Serverless v2 DB instances set an upper limit of 2000 on the `max_connections` setting.

For the following parameters, Aurora PostgreSQL also uses default values that are derived from the memory size based on the maximum ACU setting, the same as with `max_connections`:

- `autovacuum_max_workers`
- `autovacuum_vacuum_cost_limit`
- `autovacuum_work_mem`
- `effective_cache_size`
- `maintenance_work_mem`

## Avoiding out-of-memory errors

If one of your Aurora Serverless v2 DB instances consistently reaches the limit of its maximum capacity, Aurora indicates this condition by setting the DB instance to a status of `incompatible-parameters`. While the DB instance has the `incompatible-parameters` status, some operations are blocked. For example, you can't upgrade the engine version.

Typically, your DB instance goes into this status when it restarts frequently due to out-of-memory errors. Aurora records an event when this type of restart happens. You can view the event by following the procedure in [Viewing Amazon RDS events \(p. 569\)](#). Unusually high memory usage can happen because of overhead from turning on settings such as Performance Insights and IAM authentication. It can also come from a heavy workload on your DB instance or from managing the metadata associated with a large number of schema objects.

If the memory pressure becomes lower so that the DB instance doesn't reach its maximum capacity very often, Aurora automatically changes the DB instance status back to `available`.

To recover from this condition, you can take some or all of the following actions:

- Increase the lower limit on capacity for Aurora Serverless v2 DB instances by changing the minimum Aurora capacity unit (ACU) value for the cluster. Doing so avoids issues where an idle database scales down to a capacity with less memory than is needed for the features that are turned on in your cluster. After changing the ACU settings for the cluster, reboot the Aurora Serverless v2 DB instance. Doing so evaluates whether Aurora can reset the status back to `available`.
- Increase the upper limit on capacity for Aurora Serverless v2 DB instances by changing the maximum ACU value for the cluster. Doing so avoids issues where a busy database can't scale up to a capacity with enough memory for the features that are turned on in your cluster and the database workload. After changing the ACU settings for the cluster, reboot the Aurora Serverless v2 DB instance. Doing so evaluates whether Aurora can reset the status back to `available`.
- Turn off configuration settings that require memory overhead. For example, suppose that you have features such as AWS Identity and Access Management (IAM), Performance Insights, or Aurora MySQL binary log replication turned on but don't use them. If so, you can turn them off. Or you can adjust the minimum and maximum capacity values for the cluster higher to account for the memory used by those features. For guidelines about choosing minimum and maximum capacity settings, see [Choosing the Aurora Serverless v2 capacity range for an Aurora cluster \(p. 1527\)](#).
- Reduce the workload on the DB instance. For example, you can add reader DB instances to the cluster to spread the load from read-only queries across more DB instances.
- Tune the SQL code used by your application to use fewer resources. For example, you can examine your query plans, check the slow query log, or adjust the indexes on your tables. You can also perform other traditional kinds of SQL tuning.

## Important Amazon CloudWatch metrics for Aurora Serverless v2

To get started with Amazon CloudWatch for your Aurora Serverless v2 DB instance, see [Viewing Aurora Serverless v2 logs in Amazon CloudWatch \(p. 1525\)](#). To learn more about how to monitor Aurora DB clusters through CloudWatch, see [Monitoring log events in Amazon CloudWatch \(p. 927\)](#).

You can view your Aurora Serverless v2 DB instances in CloudWatch to monitor the capacity consumed by each DB instance with the `ServerlessDatabaseCapacity` metric. You can also monitor all of the standard Aurora CloudWatch metrics, such as `DatabaseConnections` and `Queries`. For the full list of CloudWatch metrics that you can monitor for Aurora, see [Amazon CloudWatch metrics for Amazon Aurora \(p. 525\)](#). The metrics are divided into cluster-level and instance-level metrics, in [Cluster-level metrics for Amazon Aurora \(p. 525\)](#) and [Instance-level metrics for Amazon Aurora \(p. 531\)](#).

The following CloudWatch instance-level metrics are important to monitor for you to understand how your Aurora Serverless v2 DB instances are scaling up and down. All of these metrics are calculated every second. That way, you can monitor the current status of your Aurora Serverless v2 DB instances. You can set alarms to notify you if any Aurora Serverless v2 DB instance approaches a threshold for metrics related to capacity. You can determine if the minimum and maximum capacity settings are appropriate, or if you need to adjust them. You can determine where to focus your efforts for optimizing the efficiency of your database.

- `ServerlessDatabaseCapacity`. As an instance-level metric, it reports the number of ACUs represented by the current DB instance capacity. As a cluster-level metric, it represents the average of the `ServerlessDatabaseCapacity` values of all the Aurora Serverless v2 DB instances in the cluster. This metric is only a cluster-level metric in Aurora Serverless v1. In Aurora Serverless v2, it's available at the DB instance level and at the cluster level.
- `ACUUtilization`. This metric is new in Aurora Serverless v2. This value is represented as a percentage. It's calculated as the value of the `ServerlessDatabaseCapacity` metric divided by the maximum ACU value of the DB cluster. Consider the following guidelines to interpret this metric and take action:
  - If this metric approaches a value of 100 . 0, the DB instance has scaled up as high as it can. Consider increasing the maximum ACU setting for the cluster. That way, both writer and reader DB instances can scale to a higher capacity.
  - Suppose that a read-only workload causes a reader DB instance to approach an `ACUUtilization` of 100 . 0, while the writer DB instance isn't close to its maximum capacity. In that case, consider adding additional reader DB instances to the cluster. That way, you can spread the read-only part of the workload spread across more DB instances, reducing the load on each reader DB instance.
  - Suppose that you are running a production application, where performance and scalability are the primary considerations. In that case, you can set the maximum ACU value for the cluster to a high number. Your goal is for the `ACUUtilization` metric to always be below 100 . 0. With a high maximum ACU value, you can be confident that there's enough room in case there are unexpected spikes in database activity. You are only charged for the database capacity that's actually consumed.
- `CPUUtilization`. This metric is interpreted differently in Aurora Serverless v2 than in provisioned DB instances. For Aurora Serverless v2, this value is a percentage that's calculated as the amount of CPU currently being used divided by the CPU capacity that's available under the maximum ACU value of the DB cluster. Aurora monitors this value automatically and scales up your Aurora Serverless v2 DB instance when the DB instance consistently uses a high proportion of its CPU capacity.

If this metric approaches a value of 100 . 0, the DB instance has reached its maximum CPU capacity. Consider increasing the maximum ACU setting for the cluster. If this metric approaches a value of 100 . 0 on a reader DB instance, consider adding additional reader DB instances to the cluster. That way, you can spread the read-only part of the workload spread across more DB instances, reducing the load on each reader DB instance.

- **FreeableMemory.** This value represents the amount of unused memory that is available when the Aurora Serverless v2 DB instance is scaled to its maximum capacity. For every ACU that the current capacity is below the maximum capacity, this value increases by approximately 2 GiB. Thus, this metric doesn't approach zero until the DB instance is scaled up as high as it can.

If this metric approaches a value of 0, the DB instance has scaled up as much as it can and is nearing the limit of its available memory. Consider increasing the maximum ACU setting for the cluster. If this metric approaches a value of 0 on a reader DB instance, consider adding additional reader DB instances to the cluster. That way, the read-only part of the workload can be spread across more DB instances, reducing the memory usage on each reader DB instance.

- **TempStorageIops.** The number of IOPS done on local storage attached to the DB instance. It includes the IOPS for both reads and writes. This metric represents a count and is measured once per second. This is a new metric for Aurora Serverless v2. For details, see [Instance-level metrics for Amazon Aurora \(p. 531\)](#).
- **TempStorageThroughput.** The amount of data transferred to and from local storage associated with the DB instance. This metric represents bytes and is measured once per second. This is a new metric for Aurora Serverless v2. For details, see [Instance-level metrics for Amazon Aurora \(p. 531\)](#).

Typically, most scaling up for Aurora Serverless v2 DB instances is caused by memory usage and CPU activity. The **TempStorageIops** and **TempStorageThroughput** metrics can help you to diagnose the rare cases where network activity for transfers between your DB instance and local storage devices is responsible for unexpected capacity increases. To monitor other network activity, you can use these existing metrics:

- **NetworkReceiveThroughput**
- **NetworkThroughput**
- **NetworkTransmitThroughput**
- **StorageNetworkReceiveThroughput**
- **StorageNetworkThroughput**
- **StorageNetworkTransmitThroughput**

You can have Aurora publish some or all database logs to CloudWatch. You select the logs to publish by turning on the [configuration parameters such as general\\_log and slow\\_query\\_log in the DB cluster parameter group \(p. 1535\)](#) associated with the cluster that contains your Aurora Serverless v2 DB instances. When you turn off a log configuration parameter, publishing that log to CloudWatch stops. You can also delete the logs in CloudWatch if they are no longer needed.

## How Aurora Serverless v2 metrics apply to your AWS bill

The Aurora Serverless v2 charges on your AWS bill are calculated based on the same **ServerlessDatabaseCapacity** metric that you can monitor. The billing mechanism can differ from the computed CloudWatch average for this metric in cases where you use Aurora Serverless v2 capacity for only part of an hour. It can also differ if system issues make the CloudWatch metric unavailable for brief periods. Thus, you might see a slightly different value of ACU-hours on your bill than if you compute the number yourself from the **ServerlessDatabaseCapacity** average value.

## Examples of CloudWatch commands for Aurora Serverless v2 metrics

The following AWS CLI examples demonstrate how you can monitor the most important CloudWatch metrics related to Aurora Serverless v2. In each case, replace the `Value=` string for the `--dimensions` parameter with the identifier of your own Aurora Serverless v2 DB instance.

The following Linux example displays the minimum, maximum, and average capacity values for a DB instance, measured every 10 minutes over one hour. The `Linux date` command specifies the start and end times relative to the current date and time. The `sort_by` function in the `--query` parameter sorts the results chronologically based on the `Timestamp` field.

```
aws cloudwatch get-metric-statistics --metric-name "ServerlessDatabaseCapacity" \
--start-time "$(date -d '1 hour ago')" --end-time "$(date -d 'now')" --period 600 \
--namespace "AWS/RDS" --statistics Minimum Maximum Average \
--dimensions Name=DBInstanceIdentifier,Value=my_instance \
--query 'sort_by(Datapoints[*].{min:Minimum,max:Maximum,avg:Average,ts:Timestamp},&ts)' \
--output table
```

The following Linux examples demonstrate monitoring the capacity of each DB instance in a cluster. They measure the minimum, maximum, and average capacity utilization of each DB instance. The measurements are taken once each hour over a three-hour period. These examples use the `ACUUtilization` metric representing a percentage of the upper limit on ACUs, instead of `ServerlessDatabaseCapacity` representing a fixed number of ACUs. That way, you don't need to know the actual numbers for the minimum and maximum ACU values in the capacity range. You can see percentages ranging from 0 to 100.

```
aws cloudwatch get-metric-statistics --metric-name "ACUUtilization" \
--start-time "$(date -d '3 hours ago')" --end-time "$(date -d 'now')" --period 3600 \
--namespace "AWS/RDS" --statistics Minimum Maximum Average \
--dimensions Name=DBInstanceIdentifier,Value=my_writer_instance \
--query 'sort_by(Datapoints[*].{min:Minimum,max:Maximum,avg:Average,ts:Timestamp},&ts)' \
--output table

aws cloudwatch get-metric-statistics --metric-name "ACUUtilization" \
--start-time "$(date -d '3 hours ago')" --end-time "$(date -d 'now')" --period 3600 \
--namespace "AWS/RDS" --statistics Minimum Maximum Average \
--dimensions Name=DBInstanceIdentifier,Value=my_reader_instance \
--query 'sort_by(Datapoints[*].{min:Minimum,max:Maximum,avg:Average,ts:Timestamp},&ts)' \
--output table
```

The following Linux example does similar measurements as the previous ones. In this case, the measurements are for the `CPUUtilization` metric. The measurements are taken every 10 minutes over a 1-hour period. The numbers represent the percentage of available CPU used, based on the CPU resources available to the maximum capacity setting for the DB instance.

```
aws cloudwatch get-metric-statistics --metric-name "CPUUtilization" \
--start-time "$(date -d '1 hour ago')" --end-time "$(date -d 'now')" --period 600 \
--namespace "AWS/RDS" --statistics Minimum Maximum Average \
--dimensions Name=DBInstanceIdentifier,Value=my_instance \
--query 'sort_by(Datapoints[*].{min:Minimum,max:Maximum,avg:Average,ts:Timestamp},&ts)' \
--output table
```

The following Linux example does similar measurements as the previous ones. In this case, the measurements are for the `FreeableMemory` metric. The measurements are taken every 10 minutes over a 1-hour period.

```
aws cloudwatch get-metric-statistics --metric-name "FreeableMemory" \
--start-time "$(date -d '1 hour ago')" --end-time "$(date -d 'now')" --period 600 \
--namespace "AWS/RDS" --statistics Minimum Maximum Average \
--dimensions Name=DBInstanceIdentifier,Value=my_instance \
--query 'sort_by(Datapoints[*].{min:Minimum,max:Maximum,avg:Average,ts:Timestamp},&ts)' \
--output table
```

## Monitoring Aurora Serverless v2 performance with Performance Insights

You can use Performance Insights to monitor the performance of Aurora Serverless v2 DB instances. For Performance Insights procedures, see [Monitoring DB load with Performance Insights on Amazon Aurora \(p. 461\)](#).

The following new Performance Insights counters apply to Aurora Serverless v2 DB instances:

- `os.general.serverlessDatabaseCapacity` – The current capacity of the DB instance in ACUs. The value corresponds to the `ServerlessDatabaseCapacity` CloudWatch metric for the DB instance.
- `os.general.acuUtilization` – The percentage of current capacity out of the maximum configured capacity. The value corresponds to the `ACUUtilization` CloudWatch metric for the DB instance.
- `os.general.maxConfiguredAcu` – The maximum capacity that you configured for this Aurora Serverless v2 DB instance. It's measured in ACUs.
- `os.general.minConfiguredAcu` – The minimum capacity that you configured for this Aurora Serverless v2 DB instance. It's measured in ACUs

For the full list of Performance Insights counters, see [Performance Insights counter metrics \(p. 546\)](#).

When vCPU values are shown for an Aurora Serverless v2 DB instance in Performance Insights, those values represent estimates based on the ACU value for the DB instance. At the default interval of one minute, any fractional vCPU values are rounded up to the nearest whole number. For longer intervals, the vCPU value shown is the average of the integer vCPU values for each minute.

# 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 v1 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.

For an Aurora Serverless v2 cluster, you can choose whether to encrypt the cluster volume.

For an Aurora Serverless v1 cluster, the cluster volume 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 \(p. 1559\)](#).

## Topics

- [Advantages of Aurora Serverless v1 \(p. 1543\)](#)
- [Use cases for Aurora Serverless v1 \(p. 1544\)](#)
- [Limitations of Aurora Serverless v1 \(p. 1544\)](#)
- [Configuration requirements for Aurora Serverless v1 \(p. 1545\)](#)
- [Using TLS/SSL with Aurora Serverless v1 \(p. 1546\)](#)
- [How Aurora Serverless v1 works \(p. 1548\)](#)
- [Creating an Aurora Serverless v1 DB cluster \(p. 1559\)](#)
- [Restoring an Aurora Serverless v1 DB cluster \(p. 1566\)](#)
- [Modifying an Aurora Serverless v1 DB cluster \(p. 1570\)](#)
- [Scaling Aurora Serverless v1 DB cluster capacity manually \(p. 1574\)](#)
- [Viewing Aurora Serverless v1 DB clusters \(p. 1576\)](#)
- [Deleting an Aurora Serverless v1 DB cluster \(p. 1578\)](#)
- [Aurora Serverless v1 and Aurora database engine versions \(p. 1580\)](#)

## Important

Aurora has two generations of serverless technology, Aurora Serverless v2 and Aurora Serverless v1. If your application can run on MySQL 8.0 or PostgreSQL 13, we recommend that you use Aurora Serverless v2. Aurora Serverless v2 scales more quickly and in a more granular way. Aurora Serverless v2 also has more compatibility with other Aurora features such as reader DB instances. Thus, if you're already familiar with Aurora, you don't have to learn as many new procedures or limitations to use Aurora Serverless v2 as with Aurora Serverless v1. You can learn about Aurora Serverless v2 in [Using Aurora Serverless v2 \(p. 1482\)](#).

## 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:

- 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 \(p. 32\)](#).
- 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

- Viewing logs in the AWS Management Console
- 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 v1 DB clusters. Only provisioned Aurora DB clusters support CDC with AWS DMS as a source.
  - You can't load text file data to Aurora Serverless v1 from Amazon S3. However, you can load data to Aurora Serverless v1 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 data from Amazon S3 into an Aurora PostgreSQL DB cluster \(p. 1230\)](#).
- 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 Serverless v1 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](#).
  - Creating users with host-based access ('`username`'@'`IP_address`'). This is because Aurora Serverless v1 uses a router fleet between the client and the database host for seamless scaling. The IP address that the Aurora Serverless DB cluster sees is that of the router host and not your client. For more information, see [Aurora Serverless v1 architecture \(p. 1549\)](#).

Instead, use the wildcard ('`username`'@'%').

- 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` commands prevent scaling and aren't recommended.

For more information, see [Autoscaling for Aurora Serverless v1 \(p. 1549\)](#).

- 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 formerly weren't supported for Aurora Serverless v1. These parameters are available now for Aurora Serverless v1. For a complete list of parameters supported by Aurora Serverless v1, call the [DescribeEngineDefaultClusterParameters](#) API operation. For more information on parameter groups and Aurora Serverless v1, see [Parameter groups for Aurora Serverless v1 \(p. 1554\)](#).

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

Description of TLS/SSL mode	mysql	psql
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 \(p. 207\)](#).

## Supported cipher suites for connections to Aurora Serverless v1 DB clusters

By using configurable cipher suites, you can have more control over the security of your database connections. You can specify a list of cipher suites that you want to allow to secure client TLS/SSL connections to your database. With configurable cipher suites, you can control the connection encryption that your database server accepts. Doing this prevents the use of ciphers that aren't secure or that are no longer used.

Aurora Serverless v1 DB clusters that are based on Aurora MySQL support the same cipher suites as Aurora MySQL provisioned DB clusters. For information about these cipher suites, see [Configuring cipher suites for connections to Aurora MySQL DB clusters \(p. 685\)](#).

Aurora Serverless v1 DB clusters that are based on Aurora PostgreSQL don't support cipher suites.

## How Aurora Serverless v1 works

Following, you can learn how Aurora Serverless v1 works.

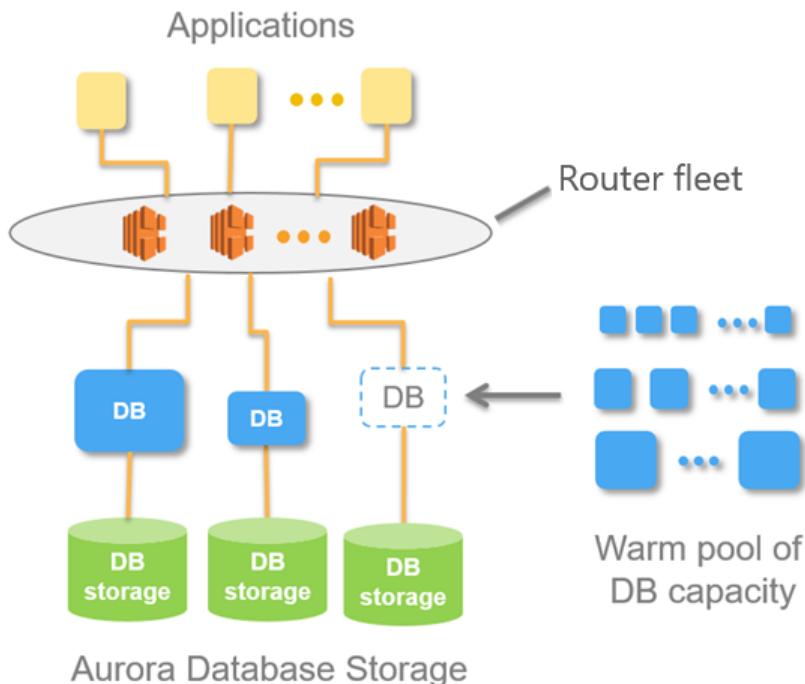
### Topics

- [Aurora Serverless v1 architecture \(p. 1549\)](#)
- [Autoscaling for Aurora Serverless v1 \(p. 1549\)](#)
- [Timeout action for capacity changes \(p. 1550\)](#)
- [Pause and resume for Aurora Serverless v1 \(p. 1551\)](#)
- [Determining the maximum number of database connections for Aurora Serverless v1 \(p. 1552\)](#)
- [Parameter groups for Aurora Serverless v1 \(p. 1554\)](#)
- [Logging for Aurora Serverless v1 \(p. 1556\)](#)
- [Aurora Serverless v1 and maintenance \(p. 1558\)](#)

- [Aurora Serverless v1 and failover \(p. 1559\)](#)
- [Aurora Serverless v1 and snapshots \(p. 1559\)](#)

## Aurora Serverless v1 architecture

The following image shows an overview of 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 charged only 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 v1 also scales up when it detects performance issues that can be resolved by doing so.

You can view scaling events for your Aurora Serverless v1 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 or that Aurora Serverless v1 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 \(p. 427\)](#).

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 \(p. 1551\)](#).

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 v1 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 v1 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 operations that block scaling and how to avoid them, see [Best practices for working with Aurora Serverless v1](#).

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 v1 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 v1 DB cluster by selecting the **Force the capacity change** option. For more information, see [Timeout action for capacity changes \(p. 1550\)](#).

## 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 selecting 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.

By default, the **Force the capacity change** option isn't selected. Keep this option clear to have your Aurora Serverless v1 DB cluster's capacity remain unchanged if the scaling operation times out without finding a scaling point.

Selecting this option causes your Aurora Serverless v1 DB cluster to enforce the capacity change, even without a scaling point. Before selecting this option, be aware of the consequences of this selection:

- 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.

We recommend that you select the **Force the capacity change** option only if your application can recover from dropped connections or incomplete transactions.

The choices that you make in the AWS Management Console when you create an Aurora Serverless v1 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 select the **Roll back the capacity change** option. This is the default behavior.
- `ForceApplyCapacityChange` – This value is set when you select the **Force the capacity change** option.

You can get the value of this property on an existing Aurora Serverless v1 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 \(p. 1559\)](#). 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 \(p. 1570\)](#).

## 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 v1 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.

## Determining the maximum number of database connections for Aurora Serverless v1

The following examples are for an Aurora Serverless v1 DB cluster that's compatible with MySQL 5.7. You can use a MySQL client or the query editor, if you've configured access to it. For more information, see [Running queries in the query editor \(p. 1614\)](#).

### To find the maximum number of database connections

- Find the capacity range for your Aurora Serverless v1 DB cluster using the AWS CLI.

```
aws rds describe-db-clusters \
--db-cluster-identifier my-serverless-57-cluster \
--query 'DBClusters[*].ScalingConfigurationInfo[0]'
```

The result shows that its capacity range is 1–4 ACUs.

```
{
  "MinCapacity": 1,
  "AutoPause": true,
  "MaxCapacity": 4,
  "TimeoutAction": "RollbackCapacityChange",
  "SecondsUntilAutoPause": 3600
}
```

- Run the following SQL query to find the maximum number of connections.

```
select @@max_connections;
```

The result shown is for the minimum capacity of the cluster, 1 ACU.

```
@@max_connections
90
```

- Scale the cluster to 8–32 ACUs.

For more information on scaling, see [Modifying an Aurora Serverless v1 DB cluster \(p. 1570\)](#).

- Confirm the capacity range.

```
{
  "MinCapacity": 8,
  "AutoPause": true,
  "MaxCapacity": 32,
  "TimeoutAction": "RollbackCapacityChange",
```

```
        "SecondsUntilAutoPause": 3600
    }
```

5. Find the maximum number of connections.

```
select @@max_connections;
```

The result shown is for the minimum capacity of the cluster, 8 ACUs.

```
@@max_connections
1000
```

6. Scale the cluster to the maximum possible, 256–256 ACUs.
7. Confirm the capacity range.

```
{
    "MinCapacity": 256,
    "AutoPause": true,
    "MaxCapacity": 256,
    "TimeoutAction": "RollbackCapacityChange",
    "SecondsUntilAutoPause": 3600
}
```

8. Find the maximum number of connections.

```
select @@max_connections;
```

The result shown is for 256 ACUs.

```
@@max_connections
6000
```

**Note**

The `max_connections` value doesn't scale linearly with the number of ACUs.

9. Scale the cluster back down to 1–4 ACUs.

```
{
    "MinCapacity": 1,
    "AutoPause": true,
    "MaxCapacity": 4,
    "TimeoutAction": "RollbackCapacityChange",
    "SecondsUntilAutoPause": 3600
}
```

This time, the `max_connections` value is for 4 ACUs.

```
@@max_connections
270
```

10. Let the cluster scale down to 2 ACUs.

```
@@max_connections
180
```

If you've configured the cluster to pause after a certain amount of time idle, it scales down to 0 ACUs. However, `max_connections` doesn't drop below the value for 1 ACU.

```
@@max_connections
90
```

## Parameter groups for 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 v1 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 v1 needs to be able to change parameters for the cluster to work best for the increased or decreased capacity. Thus, with an Aurora Serverless v1 DB cluster, some of the changes that you might make to parameters for a particular DB engine type might not apply.

For example, an Aurora PostgreSQL-based Aurora Serverless v1 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 that 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 parameter groups \(p. 215\)](#), 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 v1 DB cluster to [log queries or to upload DB engine specific logs \(p. 1556\)](#) 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 that you choose the following options:
  - a. For **Parameter group family**, choose the appropriate family for your chosen DB engine. Be sure that your choice 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 v1 DB clusters. You can also modify an existing Aurora Serverless v1 DB cluster to use your custom DB cluster parameter group. After your Aurora Serverless v1 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.

## Parameters comparison

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 v1 starts a seamless scale in order to apply the parameter changes. If your Aurora Serverless v1 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 a capacity change \(p. 1550\)](#), 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 v1 are enabled and automatically uploaded to Amazon CloudWatch. You can also have your Aurora Serverless v1 DB cluster upload Aurora database-engine specific logs to CloudWatch. To do this, enable configuration parameters in your custom DB cluster parameter group. Your Aurora Serverless v1 DB cluster then uploads all available logs to Amazon CloudWatch. At this point, you can use CloudWatch to analyze log data, create alarms, and view metrics.

For Aurora MySQL, you can turn on the following logs to have them automatically uploaded from your Aurora Serverless v1 DB cluster to Amazon CloudWatch.

Aurora MySQL log	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 3,1536,000. Default is 0 (not active).
server_audit_events	The list of events to capture in the logs. Supported values are CONNECT, QUERY,

Aurora MySQL log	Description
	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 \(p. 823\)](#).

For Aurora PostgreSQL, you can enable the following logs on your Aurora Serverless v1 DB cluster and have them automatically uploaded to Amazon CloudWatch along with the regular error logs.

Aurora PostgreSQL log	Description
log_connections	Turned on by default and can't be changed. It logs details for all new client connections.
log_disconnections	Turned on 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.
log_min_duration_statement	The minimum duration (in milliseconds) for a statement to run before it's logged.
log_min_messages	Sets the message levels that are logged. Supported values are debug5, debug4, debug3, debug2, debug1, info, notice, warning, error, log, fatal, panic. To log performance data to the postgres log, set the value to debug1.
log_temp_files	Logs the use of temporary files that are above the specified kilobytes (kB).
log_statement	Controls the specific SQL statements that get logged. Supported values are none, ddl, mod, and all. Default is none.

After you turn on logs for Aurora MySQL 5.6, Aurora MySQL 5.7, or Aurora PostgreSQL for your Aurora Serverless v1 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 \(p. 927\)](#).

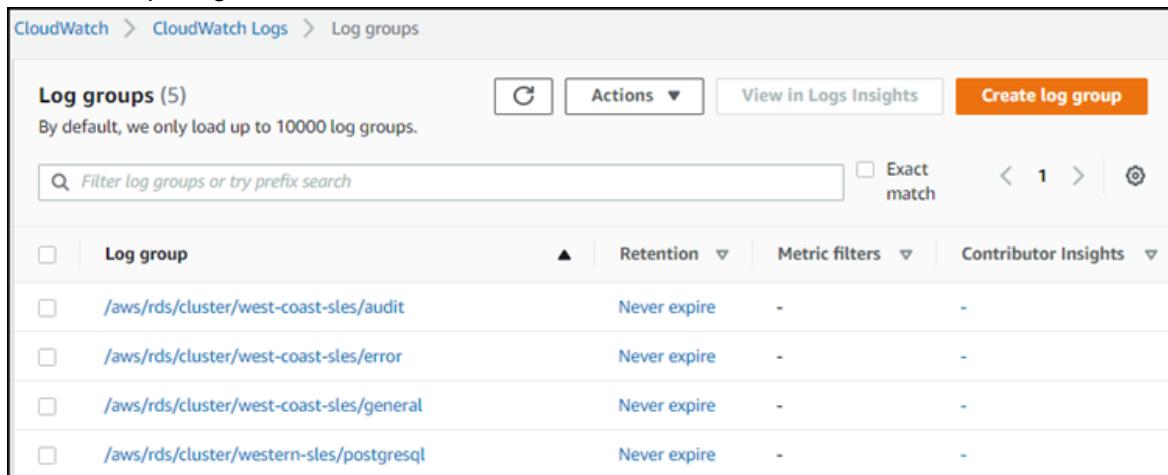
For more information about CloudWatch and Aurora PostgreSQL, see [Publishing Aurora PostgreSQL logs to Amazon CloudWatch Logs \(p. 1265\)](#).

### To view logs for your Aurora Serverless v1 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 v1 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 v1 DB cluster named `western-sles`. You can also find several listings for Aurora MySQL Aurora Serverless v1 DB cluster, `west-coast-sles`. Choose the log that you're interested in to start exploring its content.

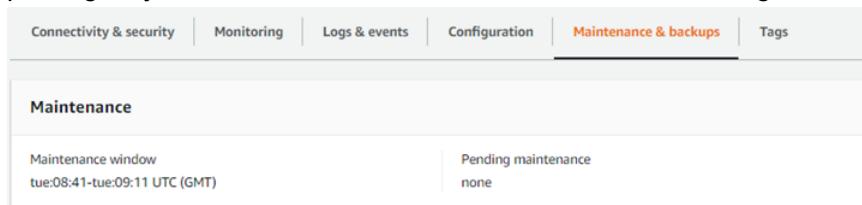


The screenshot shows the AWS CloudWatch Log Groups interface. At the top, there are navigation links: CloudWatch > CloudWatch Logs > Log groups. Below this is a search bar labeled "Filter log groups or try prefix search". To the right of the search bar are buttons for "Actions", "View in Logs Insights", and "Create log group". A checkbox labeled "Exact match" is checked. The main area displays a table of log groups:

<input type="checkbox"/>	Log group	Retention	Metric filters	Contributor Insights
<input type="checkbox"/>	/aws/rds/cluster/west-coast-sles/audit	Never expire	-	-
<input type="checkbox"/>	/aws/rds/cluster/west-coast-sles/error	Never expire	-	-
<input type="checkbox"/>	/aws/rds/cluster/west-coast-sles/general	Never expire	-	-
<input type="checkbox"/>	/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 v1 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 v1 DB cluster. You can find the date and time that maintenance might be performed and if any maintenance is pending for your Aurora Serverless v1 DB cluster, as shown following.



The screenshot shows the "Maintenance & backups" tab selected in the AWS Management Console. The interface includes tabs for Connectivity & security, Monitoring, Logs & events, Configuration, Maintenance & backups (which is highlighted), and Tags. Below the tabs, there is a section titled "Maintenance" with the following details:

Maintenance window	Pending maintenance
tue:08:41-tue:09:11 UTC (GMT)	none

Whenever possible, Aurora Serverless v1 performs maintenance in a nondisruptive manner. When maintenance is required, your Aurora Serverless v1 DB cluster scales its capacity to handle the necessary

operations. Before scaling, Aurora Serverless v1 looks for a scaling point. It does so for up to seven days if necessary.

At the end of each day that Aurora Serverless v1 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 Aurora Serverless v1 can force the DB cluster to scale.

Until that time, your Aurora Serverless v1 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 \(p. 1550\)](#).

## Aurora Serverless v1 and failover

If the DB instance for an Aurora Serverless v1 DB cluster becomes unavailable or the Availability Zone (AZ) it's in fails, Aurora recreates the DB instance in a different AZ. However, the Aurora Serverless v1 cluster isn't a Multi-AZ cluster. That's because it consists of a single DB instance in a single AZ. Thus, this failover mechanism takes longer than for an Aurora cluster with provisioned or Aurora Serverless v2 instances. The Aurora Serverless v1 failover time is 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, encrypt the snapshot using your own AWS KMS key. For more information, see [Copying a DB cluster snapshot \(p. 378\)](#). To learn more about encryption and Amazon Aurora, see [Encrypting an Amazon Aurora DB cluster \(p. 1638\)](#)

## Creating an Aurora Serverless v1 DB cluster

The following procedure creates an Aurora Serverless v1 cluster without any of your schema objects or data. If you want to create an Aurora Serverless v1 cluster that's a duplicate of an existing provisioned or Aurora Serverless v1 cluster, you can perform a snapshot restore or cloning operation instead. For those details, see [Restoring from a DB cluster snapshot \(p. 375\)](#) and [Cloning a volume for an Amazon Aurora DB cluster \(p. 280\)](#). You can't convert an existing provisioned cluster to Aurora Serverless v1. You also can't convert an existing Aurora Serverless v1 cluster back to a provisioned 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 v1 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 \(p. 1549\)](#).

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:

- **Force scaling the capacity to the specified values when the timeout is reached** – You can choose this setting if you want Aurora Serverless v1 to force Aurora Serverless v1 to scale even if it can't find a scaling point before it times out. If you want Aurora Serverless v1 to cancel capacity changes if it can't find a scaling point, don't choose this setting. For more information, see [Timeout action for capacity changes \(p. 1550\)](#).
- **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 \(p. 1551\)](#).

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 \(p. 87\)](#). You also need to complete other preliminary steps for creating any Aurora DB cluster. To learn more, see [Creating an Amazon Aurora DB cluster \(p. 127\)](#).

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 \(p. 32\)](#).

**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. With Aurora Serverless v2, you can choose whether to encrypt the cluster volume.

You can create an Aurora Serverless v1 DB cluster with the AWS Management Console, the AWS CLI, or the RDS API.

**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 \(p. 1724\)](#) 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 \(p. 207\)](#).

## Console

Use the following general procedure. For more information on creating an Aurora DB cluster using the AWS Management Console, see [Creating an Amazon Aurora DB cluster \(p. 127\)](#).

### To create a new Aurora Serverless v1 DB cluster

1. Sign in to the AWS Management Console.
2. Choose an AWS Region that supports Aurora Serverless v1.
3. Choose Amazon RDS from the AWS Services list.
4. Choose **Create database**.

5. On the **Create database** page:
  - a. Choose **Standard Create** for the database creation method.
  - b. Choose **Amazon Aurora** for the Engine type in the **Engine options** section.
  - c. 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.

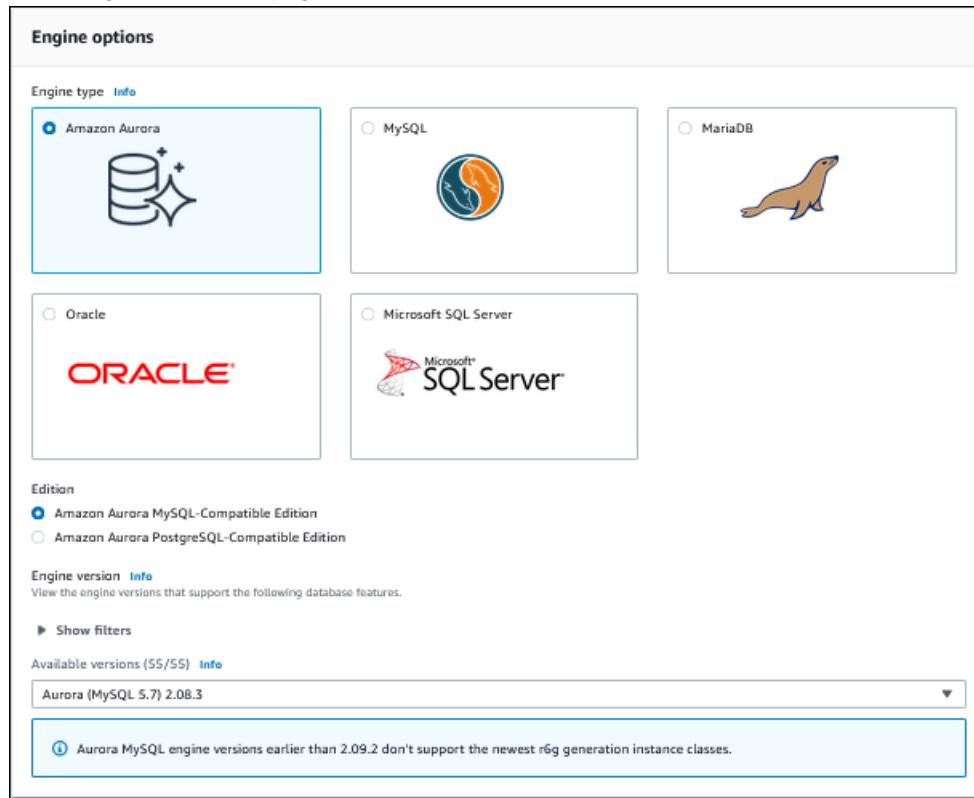
If you choose a version of the DB engine that doesn't support Aurora Serverless v1, the **Serverless** option doesn't display.

## Example for Aurora MySQL

Use the following procedure.

### To create an Aurora Serverless v1 DB cluster for Aurora MySQL

1. Choose **Amazon Aurora with MySQL Compatibility** for the Edition.
2. Choose the Aurora MySQL version you want for your DB cluster. The supported versions are shown on the right side of the page.



3. For **DB instance class**, confirm that **Serverless** is chosen.
4. Set the **Capacity range** for the DB cluster.
5. Adjust values as needed in the **Additional scaling configuration** section of the page. To learn more about capacity settings, see [Autoscaling for Aurora Serverless v1 \(p. 1549\)](#).

**Instance configuration**

The DB instance configuration options below are limited to those supported by the engine that you selected above.

DB instance class [Info](#)

Serverless

Memory optimized classes (includes r classes)

Burstable classes (includes t classes)

Serverless v1  
The previous generation of Aurora Serverless.

Include previous generation classes

Capacity range [Info](#)

Database capacity is measured in Aurora Capacity Units (ACUs). 1 ACU provides 2 GiB of memory and corresponding compute and networking.

Minimum ACUs

1 ACU  
2 GiB RAM

Maximum ACUs

64 ACU  
122 GiB RAM

▼ Additional scaling configuration

Autoscaling timeout and action [Info](#)

Specify the amount of time to allow Aurora to look for a scaling point before the timeout action.

00:05:00

Max: 10 minutes. Min: 1 minute.

If the timeout expires before a scaling point is found, do this:

Roll back the capacity change  
Your Aurora Serverless cluster's capacity isn't changed. It stays as its current capacity.

Force the capacity change  
Your Aurora Serverless cluster's capacity is changed without a scaling point. This can interrupt in-progress transactions, requiring resubmission.

Pause after inactivity [Info](#)

Scale the capacity to 0 ACUs when cluster is idle  
This optional setting allows your Aurora Serverless cluster to scale its capacity to 0 ACUs while inactive. When database traffic resumes, your Aurora Serverless cluster resumes processing capacity and scales to handle the traffic.

6. To enable the Data API for your Aurora Serverless v1 DB cluster, select the **Data API** check box under **Additional configuration** in the **Connectivity** section.

To learn more about the Data API, see [Using the Data API for Aurora Serverless v1 \(p. 1581\)](#).

7. Choose other database settings as needed, then choose **Create database**.

## Example for Aurora PostgreSQL

Use the following procedure.

### To create an Aurora Serverless v1 DB cluster for Aurora PostgreSQL

1. Choose **Amazon Aurora with Postgres; Compatibility** for the Edition.
2. Choose the Aurora PostgreSQL version you want for your DB cluster. The supported versions are shown on the right side of the page.

**Engine options**

Engine type [Info](#)

Amazon Aurora 

MySQL 

MariaDB 

PostgreSQL 

Oracle 

Microsoft SQL Server 

Edition

Amazon Aurora MySQL-Compatible Edition

Amazon Aurora PostgreSQL-Compatible Edition

Engine version [Info](#)  
View the engine versions that support the following database features.

[Show filters](#)

Available versions (22/22) [Info](#)

Aurora PostgreSQL (Compatible with PostgreSQL 10.18) ▾

 Aurora PostgreSQL engine versions earlier than 11.9 don't support the newest r6g generation instance classes.

3. For **DB instance class**, confirm that **Serverless** is chosen.
4. Set the **Capacity range** for the DB cluster.
5. Adjust values as needed in the **Additional scaling configuration** section of the page. To learn more about capacity settings, see [Autoscaling for Aurora Serverless v1 \(p. 1549\)](#).

**Instance configuration**

The DB instance configuration options below are limited to those supported by the engine that you selected above.

**DB instance class** [Info](#)

Serverless  
 Memory optimized classes (includes r classes)  
 Burstable classes (includes t classes)

**Serverless v1**  
The previous generation of Aurora Serverless.

Include previous generation classes

**Capacity range** [Info](#)

Database capacity is measured in Aurora Capacity Units (ACUs). 1 ACU provides 2 GiB of memory and corresponding compute and networking.

**Minimum ACUs** **Maximum ACUs**

2 ACU 384 ACU  
4 GiB RAM 768GB RAM

**Additional scaling configuration**

**Autoscaling timeout and action** [Info](#)

Specify the amount of time to allow Aurora to look for a scaling point before the timeout action.

00:05:00

Max: 10 minutes. Min: 1 minute.

If the timeout expires before a scaling point is found, do this:

Roll back the capacity change  
Your Aurora Serverless cluster's capacity isn't changed. It stays as its current capacity.

Force the capacity change  
Your Aurora Serverless cluster's capacity is changed without a scaling point. This can interrupt in-progress transactions, requiring resubmission.

**Pause after inactivity** [Info](#)

Scale the capacity to 0 ACUs when cluster is idle  
This optional setting allows your Aurora Serverless cluster to scale its capacity to 0 ACUs while inactive. When database traffic resumes, your Aurora Serverless cluster resumes processing capacity and scales to handle the traffic.

6. To enable the Data API for your Aurora Serverless v1 DB cluster, select the **Data API** check box under **Additional configuration** in the **Connectivity** section.  
To learn more about the Data API, see [Using the Data API for Aurora Serverless v1 \(p. 1581\)](#).
7. Choose other database settings as needed, then choose **Create database**.

## 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.mysql_aurora.1.22.3 \
--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.2.08.3 \
--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.mysql_aurora.1.22.3 ^
--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.2.08.3 ^
--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 \(p. 1550\)](#).
- **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 \(p. 375\)](#).

## 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**.

## Restore DB Cluster

### Instance specifications

#### DB Engine

Name of the Database Engine

Aurora MySQL

#### DB Engine Version

Version Number of the Database Engine to be used for this instance

Aurora (MySQL)-5.6.10a (default)

#### Capacity type [Info](#)

**Provisioned**

You provision and manage the server instance sizes.

**Serverless [Info](#)**

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.

**Instance configuration**  
The DB instance configuration options below are limited to those supported by the engine that you selected above.

**DB instance class** [Info](#)  
 Serverless  
 Memory optimized classes (includes r classes)  
 Burstable classes (includes t classes)

**Serverless v1**  
The previous generation of Aurora Serverless.  
 Include previous generation classes

**Capacity range** [Info](#)  
Database capacity is measured in Aurora Capacity Units (ACUs). 1 ACU provides 2 GiB of memory and corresponding compute and networking.

**Minimum ACUs**  
1 ACU  
2 GiB RAM

**Maximum ACUs**  
64 ACU  
122 GiB RAM

**▼ Additional scaling configuration**

**Autoscaling timeout and action** [Info](#)  
Specify the amount of time to allow Aurora to look for a scaling point before the timeout action.  
00:05:00  
Max: 10 minutes. Min: 1 minute.

If the timeout expires before a scaling point is found, do this:

Roll back the capacity change  
Your Aurora Serverless cluster's capacity isn't changed. It stays as its current capacity.

Force the capacity change  
Your Aurora Serverless cluster's capacity is changed without a scaling point. This can interrupt in-progress transactions, requiring resubmission.

**Pause after inactivity** [Info](#)  
 Scale the capacity to 0 ACUs when cluster is idle  
This optional setting allows your Aurora Serverless cluster to scale its capacity to 0 ACUs while inactive. When database traffic resumes, your Aurora Serverless cluster resumes processing capacity and scales to handle the traffic.

## 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 \(p. 207\)](#).

### 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 \(p. 1724\)](#).

## AWS CLI

You can configure an Aurora Serverless 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 DB cluster, you can set the following specific values:

- **Minimum Aurora capacity unit** – Aurora Serverless can reduce capacity down to this capacity unit.
- **Maximum Aurora capacity unit** – Aurora Serverless 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 \(p. 1550\)](#).

- **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.

**Note**

The version of the DB cluster snapshot must be compatible with Aurora Serverless v1. For the list of supported versions, see [Aurora Serverless v1 \(p. 32\)](#).

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 v1 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 certain properties with the AWS Management Console, the AWS CLI, or the RDS API. Most of the properties you can modify are the same as for other kinds of Aurora clusters. The properties that are most relevant for Aurora Serverless v1 are the scaling configuration of the cluster, and the major version for Aurora Serverless v1 clusters that are compatible with MySQL 5.6 and PostgreSQL 10.

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 automatically creates scaling rules for thresholds for CPU utilization, connections, and available memory. You can also set whether Aurora Serverless pauses the database when there's no activity and then resumes when activity begins again.

You can set the following specific values for the scaling configuration:

- **Minimum Aurora capacity unit** – Aurora Serverless can reduce capacity down to this capacity unit.
- **Maximum Aurora capacity unit** – Aurora Serverless 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 \(p. 1550\)](#).
- **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.

You can choose the major version for certain Aurora Serverless v1 DB clusters:

- DB clusters compatible with MySQL 5.6 – Choose a corresponding MySQL 5.7–compatible version number.
- DB clusters compatible with PostgreSQL 10 – Choose a corresponding PostgreSQL 11–compatible version number.

Doing so performs an in-place upgrade of the Aurora Serverless v1 DB cluster.

## 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. For **Version**:
  - Choose an Aurora MySQL version 2 version number if you want to perform an in-place upgrade from a MySQL 5.6–compatible cluster to a MySQL 5.7–compatible one.
  - Choose an Aurora PostgreSQL version 11 version number if you want to perform an in-place upgrade from a PostgreSQL 10–compatible cluster to a PostgreSQL 11–compatible one.

The following example shows an in-place upgrade from Aurora MySQL 1.22.3 to 2.08.3.

The screenshot shows the 'Modify DB cluster' page for a MySQL database cluster named 'serverless-v1-ams-cluster'. The 'Version' section lists three options: 'Aurora MySQL (compatible with MySQL 5.6.1.22.3)' (selected), 'Aurora MySQL (compatible with MySQL 5.6.1.22.3)', and 'Aurora MySQL (compatible with MySQL 5.7.2.08.3)'. A note at the bottom of this section says 'To avoid a required upgrade soon, choose Aurora MySQL version 2 or higher.' The 'DB cluster identifier' field contains 'serverless-v1-ams-cluster'. The 'New master password' field is empty. In the 'Capacity settings' section, the 'Minimum Aurora capacity units' is set to '1 ACU' (2 GB RAM) and the 'Maximum Aurora capacity units' is set to '16 ACU' (32 GB RAM). The 'Additional scaling configuration' section is collapsed. The 'Connectivity' section shows a VPC security group named 'default' selected from a dropdown menu. The 'Web Service Data API' section has a checked checkbox for 'Data API' with a note explaining it enables the SQL HTTP endpoint for running SQL queries. The 'Additional configuration' section is collapsed. At the bottom right, there are 'Cancel' and 'Continue' buttons.

If you perform a major version upgrade, leave all the other properties the same. To change any of the other properties, do another **Modify** operation after the upgrade finishes.

6. In the **Capacity settings** section, modify the scaling configuration.
7. Choose **Continue**.
8. On the **Modify DB cluster** page, review your modifications, then choose one of the following:
  - **Apply during the next scheduled maintenance window**
  - **Apply immediately**
9. Choose **Modify cluster**.

## 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
```

To perform an in-place upgrade from a MySQL 5.6-compatible Aurora Serverless v1 DB cluster to a MySQL 5.7-compatible one, specify the `--engine-version` parameter with an Aurora MySQL version 2 version number that's compatible with Aurora Serverless v1. Also include the `--allow-major-version-upgrade` parameter.

In this example, you modify the major version of a MySQL 5.6-compatible Aurora Serverless v1 DB cluster named *sample-cluster*. Doing so performs an in-place upgrade to a MySQL 5.7-compatible Aurora Serverless v1 DB cluster.

For Linux, macOS, or Unix:

```
aws rds modify-db-cluster \
    --db-cluster-identifier sample-cluster \
    --engine-version 5.7.mysql_aurora.2.08.3 \
    --allow-major-version-upgrade
```

For Windows:

```
aws rds modify-db-cluster ^
    --db-cluster-identifier sample-cluster ^
    --engine-version 5.7.mysql_aurora.2.08.3 ^
    --allow-major-version-upgrade
```

To perform an in-place upgrade from a PostgreSQL 10-compatible Aurora Serverless v1 DB cluster to a PostgreSQL 11-compatible one, specify the `--engine-version` parameter with an Aurora PostgreSQL version 11 version number that's compatible with Aurora Serverless v1. Also include the `--allow-major-version-upgrade` parameter.

In this example, you modify the major version of a PostgreSQL 10-compatible Aurora Serverless v1 DB cluster named *sample-cluster*. Doing so performs an in-place upgrade to a PostgreSQL 11-compatible Aurora Serverless v1 DB cluster.

For Linux, macOS, or Unix:

```
aws rds modify-db-cluster \
    --db-cluster-identifier sample-cluster \
```

```
--engine-version 11.13 \
--allow-major-version-upgrade
```

For Windows:

```
aws rds modify-db-cluster ^
--db-cluster-identifier sample-cluster ^
--engine-version 11.13 ^
--allow-major-version-upgrade
```

## 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.

To perform an in-place upgrade from a MySQL 5.6-compatible Aurora Serverless v1 DB cluster to a MySQL 5.7-compatible one, specify the `EngineVersion` parameter with an Aurora MySQL version 2 version number that's compatible with Aurora Serverless v1. Also include the `AllowMajorVersionUpgrade` parameter.

To perform an in-place upgrade from a PostgreSQL 10-compatible Aurora Serverless v1 DB cluster to a PostgreSQL 11-compatible one, specify the `EngineVersion` parameter with an Aurora PostgreSQL version 11 version number that's compatible with Aurora Serverless v1. Also include the `AllowMajorVersionUpgrade` parameter.

# 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.

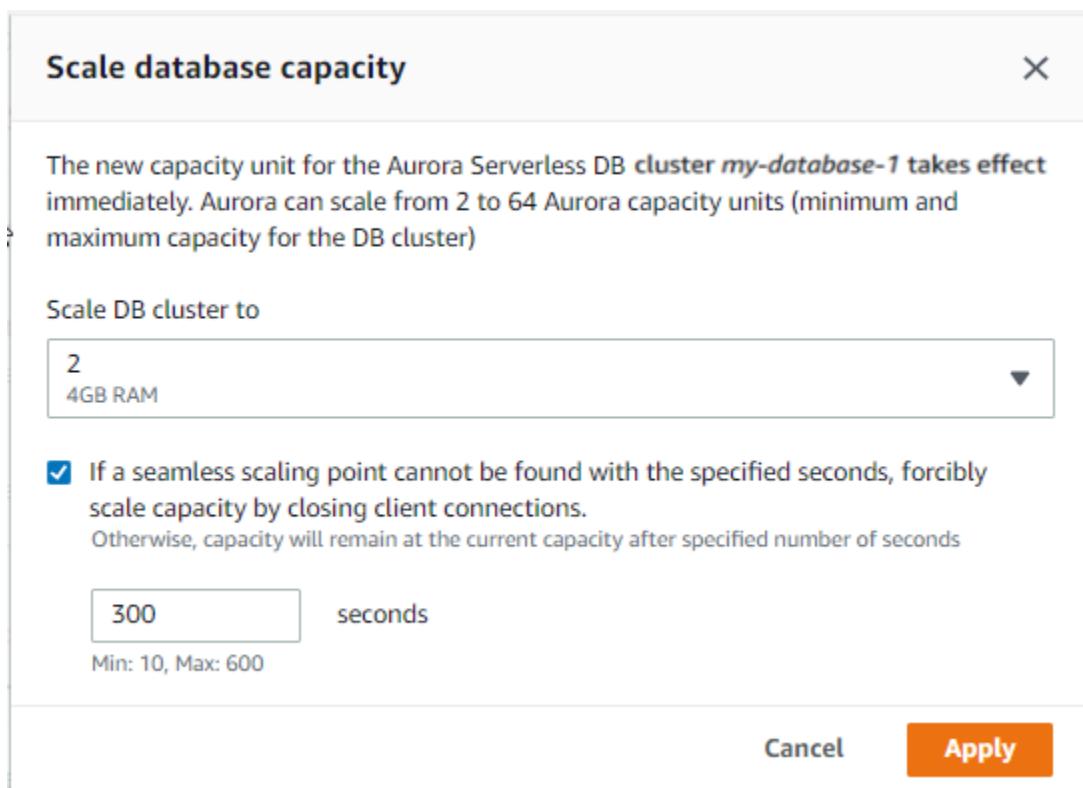
## Console

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** check box, choose the behavior that you want for your Aurora Serverless v1 DB cluster's `TimeoutAction` setting, as follows:
    - Clear this option if you want your capacity to remain unchanged if Aurora Serverless v1 can't find a scaling point before timing out.
    - Select 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. For **seconds**, 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 10 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 \(p. 1549\)](#).

## 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

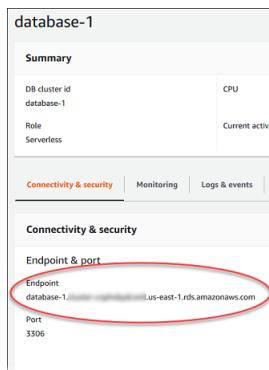
1. Sign in to the AWS Management Console and open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. In the upper-right corner of the AWS Management Console, choose the AWS Region in which you created the Aurora Serverless v1 DB clusters.
3. In the navigation pane, choose **Databases**.

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**.

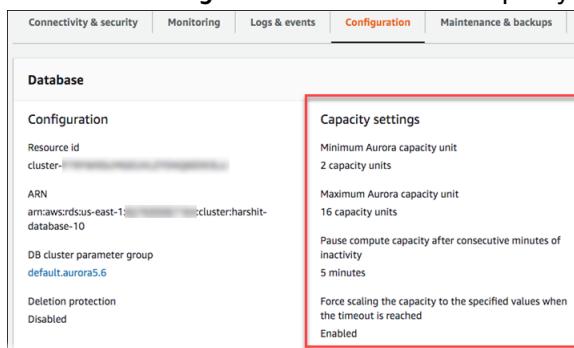
DB identifier	Role	Engine	Region & AZ	Size	Status
database-1	Serverless	Aurora MySQL	us-east-1	0 capacity units	<span>Green</span>
my-database-1	Serverless	Aurora PostgreSQL	us-east-1	0 capacity units	<span>Green</span>
mysqls3	Instance	MySQL	us-east-1c	db.t3.medium	<span>Green</span>
testsAurora	Regional	Aurora MySQL	us-east-1	2 instances	<span>Green</span>
testsAurora	Writer	Aurora MySQL	us-east-1a	db.t3.medium	<span>Green</span>
testsAurora-us-east-1b	Reader	Aurora MySQL	us-east-1b	db.t3.medium	<span>Green</span>

4. Choose the name of an Aurora Serverless v1 DB cluster to display its details.

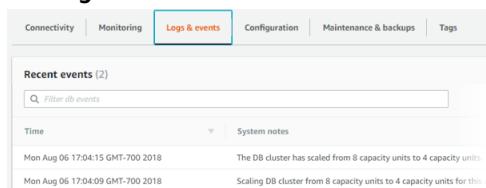
On the **Connectivity & security** tab, note the database endpoint. Use this endpoint to connect to your Aurora Serverless v1 DB cluster.



Choose the **Configuration** tab to view the capacity settings.



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.



## 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. 1554) 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 \(p. 1557\)](#). To learn more about how to monitor Aurora DB clusters through CloudWatch, see [Monitoring log events in Amazon CloudWatch \(p. 927\)](#).

To connect to an Aurora Serverless v1 DB cluster, use the database endpoint. For more information, see [Connecting to an Amazon Aurora DB cluster \(p. 207\)](#).

**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 \(p. 1579\)](#).

### To disable deletion protection using the AWS Management Console

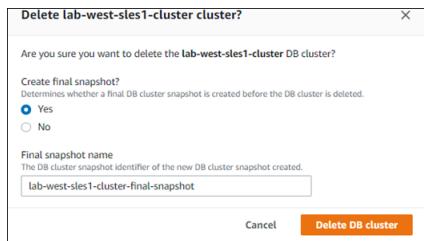
1. Sign in to the AWS Management Console and open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. In the navigation pane, choose **DB clusters**.
3. Choose your Aurora Serverless v1 DB cluster from the list.
4. Choose **Modify** to open your DB cluster's configuration. The Modify 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. Clear the **Enable deletion protection** check box 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 is. 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 \(p. 32\)](#).

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 and 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 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 Serverless clusters](#) in the *Release Notes for Aurora MySQL*.

## Aurora PostgreSQL Serverless

If you want to use Aurora PostgreSQL for your Aurora Serverless v1 DB cluster, you can choose among Aurora PostgreSQL 10-compatible and 11-compatible versions. 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 AWS Region.

For example, the minor version Aurora PostgreSQL 10.18 release was transparently applied to all Aurora Serverless v1 DB clusters running the previous Aurora PostgreSQL version. For more information about the Aurora PostgreSQL version 10.18 update, see [PostgreSQL 10.18](#) in the *Release Notes for Aurora PostgreSQL*.

# Using the Data API for Aurora Serverless v1

By using the Data API for Aurora Serverless v1, you can work with a web-services interface to your Aurora Serverless v1 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 \(p. 1600\)](#).

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 \(p. 1582\)](#).

You can also use Data API to integrate Aurora Serverless v1 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 v1 cluster. You can also modify the configuration later. For more information, see [Enabling the Data API \(p. 1584\)](#).

After you enable the Data API, you can also use the query editor for Aurora Serverless v1. For more information, see [Using the query editor for Aurora Serverless v1 \(p. 1613\)](#).

**Note**

The Data API and query editor aren't supported for Aurora Serverless v2.

**Topics**

- [Data API availability \(p. 1581\)](#)
- [Authorizing access to the Data API \(p. 1582\)](#)
- [Enabling the Data API \(p. 1584\)](#)
- [Creating an Amazon VPC endpoint for the Data API \(AWS PrivateLink\) \(p. 1586\)](#)
- [Calling the Data API \(p. 1588\)](#)
- [Using the Java client library for Data API \(p. 1602\)](#)
- [Processing query results in JSON format \(p. 1603\)](#)
- [Troubleshooting Data API issues \(p. 1608\)](#)
- [Logging Data API calls with AWS CloudTrail \(p. 1610\)](#)

## Data API availability

The Data API is available only for Aurora Serverless v1 DB clusters using specific Aurora MySQL and Aurora PostgreSQL versions, and in specific AWS Regions. For more information, see [Data API for Aurora Serverless v1 \(p. 33\)](#).

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 \(p. 1584\)](#).

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:)"
            ],
            "Resource": "arn:aws:rds:us-east-2:111122223333:cluster:*",
            "Condition": {
                "StringEquals": {
                    "aws:ResourceTag/environment": [
                        "production"
                    ]
                }
            }
        }
    ]
}
```

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 v1 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 [Create a database 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 v1 DB cluster. You can enable the Data API when you create or modify the DB cluster.

**Note**

Currently, you can't use the Data API with Aurora Serverless v2 DB instances.

### Console

You can enable the Data API by using the RDS console when you create or modify an Aurora Serverless v1 DB cluster. When you create an Aurora Serverless v1 DB cluster, you do so by enabling the Data API in the RDS console's **Connectivity** section. When you modify an Aurora Serverless v1 DB cluster, you do so by enabling the Data API in the RDS console's **Network & Security** section.

The following screenshot shows the enabled **Data API** when modifying an Aurora Serverless v1 DB cluster.

## Connectivity

**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 VPC security groups* ▾

**default X**

**Web Service Data API**

**Data API** [Info](#)

Enable the SQL HTTP endpoint, a connectionless Web Service API for running SQL queries against this database. When the SQL HTTP endpoint is enabled, you can also query your database from inside the RDS console (these features are free to use).

For instructions, see [Creating an Aurora Serverless v1 DB cluster \(p. 1559\)](#) and [Modifying an Aurora Serverless v1 DB cluster \(p. 1570\)](#).

## AWS CLI

When you create or modify an Aurora Serverless v1 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 v1 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)

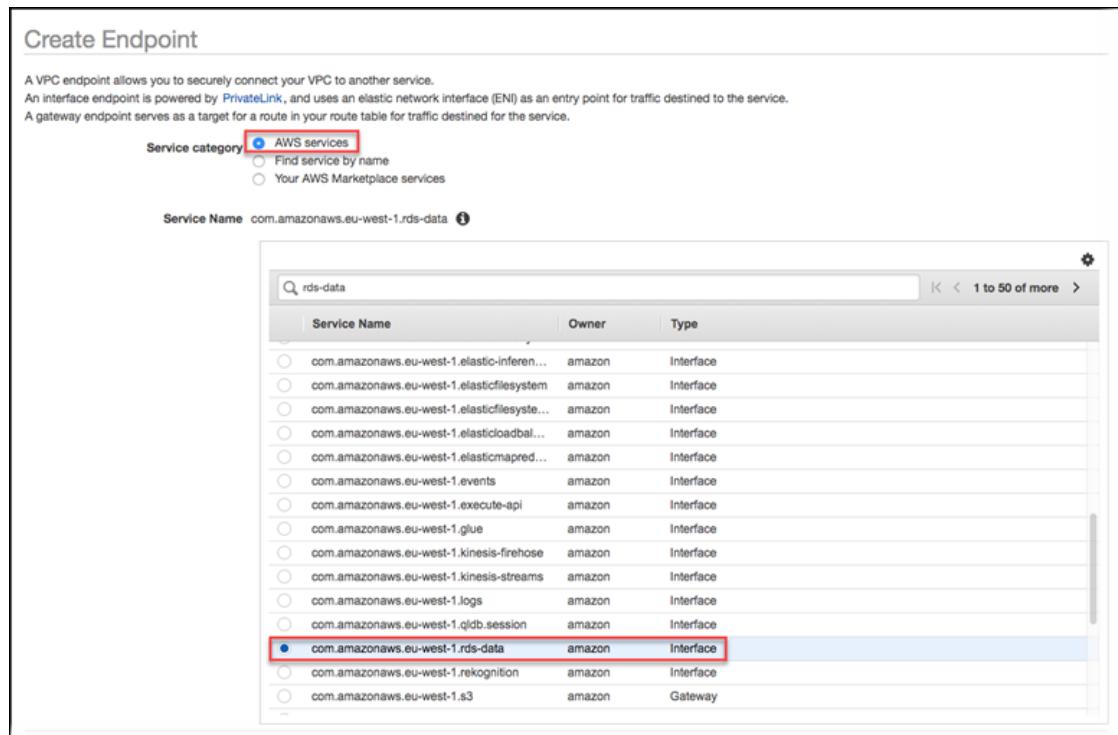
Amazon VPC enables you to launch AWS resources, such as Aurora DB clusters and applications, into a virtual private cloud (VPC). AWS PrivateLink provides private connectivity between VPCs and AWS services with high security on the Amazon network. Using AWS PrivateLink, you can create Amazon VPC endpoints, 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**.



4. For **VPC**, choose the VPC to create the endpoint in.

Choose the VPC that contains the application that makes Data API calls.

- For **Subnets**, choose the subnet for each Availability Zone (AZ) used by the AWS service that is running your application.

Availability Zone	Subnet ID
<input checked="" type="checkbox"/> eu-west-1a (euw1-az2)	subnet-2c8ee364
<input checked="" type="checkbox"/> eu-west-1b (euw1-az3)	subnet-cefa2594
<input checked="" type="checkbox"/> 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.

- For **Enable DNS name**, select **Enable for this endpoint**.

Enable DNS name  Enable for this endpoint

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.

- 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.

- 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.

- Choose **Create endpoint**.

After the endpoint is created, choose the link in the AWS Management Console to view the endpoint details.

Endpoints > Create Endpoint

## Create Endpoint

The following VPC Endpoint was created:

VPC Endpoint ID vpce-0...

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	vpc-c20d2da4				
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-[REDACTED].rds-data.eu-west-1.vpce.amazonaws.com ([REDACTED]) vpce-[REDACTED]-eu-west-1b.rds-data.eu-west-1.vpce.amazonaws.com ([REDACTED]) vpce-[REDACTED]-eu-west-1c.rds-data.eu-west-1.vpce.amazonaws.com ([REDACTED]) vpce-[REDACTED]-eu-west-1a.rds-data.eu-west-1.vpce.amazonaws.com ([REDACTED]) rds-data.eu-west-1.amazonaws.com ([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 \(p. 1588\)](#).

## Calling the Data API

With the Data API enabled on your Aurora Serverless v1 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
BeginTransaction	<code>rds-data begin-transaction</code>	Starts a SQL transaction.
CommitTransaction	<code>rds-data commit-transaction</code>	Ends a SQL transaction and commits the changes.
RollbackTransaction	<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 v1 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
<code>INTEGER, TINYINT, SMALLINT, BIGINT</code>	<code>LONG (or STRING)</code>
<code>FLOAT, REAL, DOUBLE</code>	<code>DOUBLE</code>
<code>DECIMAL</code>	<code>STRING</code>
<code>BOOLEAN, BIT</code>	<code>BOOLEAN</code>
<code>BLOB, BINARY, LONGVARBINARY, VARBINARY</code>	<code>BLOB</code>
<code>CLOB</code>	<code>STRING</code>
Other types (including types related to date and time)	<code>STRING</code>

#### 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 v1 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 \(p. 1590\)](#)
- [Running a SQL statement \(p. 1591\)](#)
- [Running a batch SQL statement over an array of data \(p. 1594\)](#)
- [Committing a SQL transaction \(p. 1595\)](#)
- [Rolling back a SQL transaction \(p. 1596\)](#)

## 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.

- `--format-records-as "JSON" | "NONE"` – An optional value that specifies whether to format the result set as a JSON string. The default is "NONE". For usage information about processing JSON result sets, see [Processing query results in JSON format \(p. 1603\)](#).

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\\": \\"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\\": \\"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 v1 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 \(p. 1597\)](#)
- [Running a DML SQL statement \(p. 1598\)](#)
- [Running a SQL transaction \(p. 1598\)](#)

## 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': 'FERNANDEZ'
        },
        {
            'stringValue': 'SARAH'
        },
        {
            'stringValue': '2016-02-15 04:34:33.0'
        }
    ]
]
```

```
{  
    '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 v1 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 \(p. 1599\)](#)
- [Running a SQL transaction \(p. 1600\)](#)
- [Running a batch SQL operation \(p. 1601\)](#)

## 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.beginTransactionbeginTransactionRequest();
        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);

```

#### Note

The `mapToList` function converts a SQL result set into a user-defined object list. We don't support using the `.withFormatRecordsAs(RecordsFormatType.JSON)` statement in an `ExecuteStatement` call for the Java client library, because it serves the same purpose. For more information, see [Processing query results in JSON format \(p. 1603\)](#).

## Processing query results in JSON format

When you call the `ExecuteStatement` operation, you can choose to have the query results returned as a string in JSON format. That way, you can use your programming language's JSON parsing capabilities

to interpret and reformat the result set. Doing so can help to avoid writing extra code to loop through the result set and interpret each column value.

To request the result set in JSON format, you pass the optional `formatRecordsAs` parameter with a value of `JSON`. The JSON-formatted result set is returned in the `formattedRecords` field of the `ExecuteStatementResponse` structure.

The `BatchExecuteStatement` action doesn't return a result set. Thus, the JSON option doesn't apply to that action.

To customize the keys in the JSON hash structure, define column aliases in the result set. You can do so by using the `AS` clause in the column list of your SQL query.

You might use the JSON capability to make the result set easier to read and map its contents to language-specific frameworks. Because the volume of the ASCII-encoded result set is larger than the default representation, you might choose the default representation for queries that return large numbers of rows or large column values that consume more memory than is available to your application.

#### Topics

- [Retrieving query results in JSON format \(p. 1604\)](#)
- [Data Type Mapping \(p. 1604\)](#)
- [Troubleshooting \(p. 1605\)](#)
- [Examples \(p. 1605\)](#)

## Retrieving query results in JSON format

To receive the the result set as a JSON string, include

`.withFormatRecordsAs(RecordsFormatType.JSON)` in the `ExecuteStatement` call. The return value comes back as a JSON string in the `formattedRecords` field. In this case, the `columnMetadata` is `null`. The column labels are the keys of the object that represents each row. These column names are repeated for each row in the result set. The column values are quoted strings, numeric values, or special values representing `true`, `false`, or `null`. Column metadata such as length constraints and the precise type for numbers and strings isn't preserved in the JSON response.

If you omit the `.withFormatRecordsAs()` call or specify a parameter of `NONE`, the result set is returned in binary format using the `Records` and `columnMetadata` fields.

## Data Type Mapping

The SQL values in the result set are mapped to a smaller set of JSON types. The values are represented in JSON as strings, numbers, and some special constants such as `true`, `false`, and `null`. You can convert these values into variables in your application, using strong or weak typing as appropriate for your programming language.

JDBC data type	JSON data type
<code>INTEGER</code> , <code>TINYINT</code> , <code>SMALLINT</code> , <code>BIGINT</code>	Number by default. String if the <code>LongReturnType</code> option is set to <code>STRING</code> .
<code>FLOAT</code> , <code>REAL</code> , <code>DOUBLE</code>	Number
<code>DECIMAL</code>	String by default. Number if the <code>DecimalReturnType</code> option is set to <code>DOUBLE_OR_LONG</code> .

JDBC data type	JSON data type
STRING	String
BOOLEAN, BIT	Boolean
BLOB, BINARY, VARBINARY, LONGVARBINARY	String in base64 encoding.
CLOB	String
ARRAY	Array
NULL	null
Other types (including types related to date and time)	String

## Troubleshooting

The JSON response is limited to 10 megabytes. If the response is larger than this limit, your program receives a `BadRequestException` error. In this case, you can resolve the error using one of the following techniques:

- Reduce the number of rows in the result set. To do so, add a `LIMIT` clause. You might split a large result set into multiple smaller ones by submitting several queries with `LIMIT` and `OFFSET` clauses. If the result set includes rows that are filtered out by application logic, you can remove those rows from the result set by adding more conditions in the `WHERE` clause.
- Reduce the number of columns in the result set. To do so, remove items from the select list of the query.
- Shorten the column labels by using column aliases in the query. Each column name is repeated in the JSON string for each row in the result set. Thus, a query result with long column names and many rows could exceed the size limit. In particular, use column aliases for complicated expressions to avoid having the entire expression repeated in the JSON string.
- Although with SQL you can use column aliases to produce a result set having more than one column with the same name, duplicate key names aren't allowed in JSON. The RDS Data API returns an error if you request the result set in JSON format and more than one column has the same name. Thus, make sure that all the column labels have unique names.

## Examples

The following Java examples show how to call `ExecuteStatement` with the response as a JSON-formatted string, then interpret the result set. Substitute the appropriate values for the `databaseName`, `secretStoreArn`, and `clusterArn` parameters.

The following Java example demonstrates a query that returns a decimal numeric value in the result set. The `assertThat` calls test that the fields of the response have the expected properties based on the rules for JSON result sets.

This example works with the following schema and sample data:

```
create table test_simplified_json (a float);
insert into test_simplified_json values(10.0);

public void JSON_result_set_demo() {
```

```

var sql = "select * from test_simplified_json";
var request = new ExecuteStatementRequest()
    .withDatabase(databaseName)
    .withSecretArn(secretStoreArn)
    .withResourceArn(clusterArn)
    .withSql(sql)
    .withFormatRecordsAs(RecordsFormatType.JSON);
var result = rdsdataClient.executeStatement(request);
}

```

The value of the `formattedRecords` field from the preceding program is:

```
[{"a":10.0}]
```

The `Records` and `ColumnMetadata` fields in the response are both null, due to the presence of the JSON result set.

The following Java example demonstrates a query that returns an integer numeric value in the result set. The example calls `getFormattedRecords` to return only the JSON-formatted string and ignore the other response fields that are blank or null. The example deserializes the result into a structure representing a list of records. Each record has fields whose names correspond to the column aliases from the result set. This technique simplifies the code that parses the result set. Your application doesn't have to loop through the rows and columns of the result set and convert each value to the appropriate type.

This example works with the following schema and sample data:

```

create table test_simplified_json (a int);
insert into test_simplified_json values(17);

```

```

public void JSON_deserialization_demo() {
    var sql = "select * from test_simplified_json";
    var request = new ExecuteStatementRequest()
        .withDatabase(databaseName)
        .withSecretArn(secretStoreArn)
        .withResourceArn(clusterArn)
        .withSql(sql)
        .withFormatRecordsAs(RecordsFormatType.JSON);
    var result = rdsdataClient.executeStatement(request)
        .getFormattedRecords();

    /* Turn the result set into a Java object, a list of records.
     * Each record has a field 'a' corresponding to the column
     * labelled 'a' in the result set. */
    private static class Record { public int a; }
    var recordsList = new ObjectMapper().readValue(
        response, new TypeReference<List<Record>>() {
    });
}

```

The value of the `formattedRecords` field from the preceding program is:

```
[{"a":17}]
```

To retrieve the `a` column of result row 0, the application would refer to `recordsList.get(0).a`.

In contrast, the following Java example shows the kind of code that's required to construct a data structure holding the result set when you don't use the JSON format. In this case, each row of the result set contains fields with information about a single user. Building a data structure to represent the result set requires looping through the rows. For each row, the code retrieves the value of each field, performs an appropriate type conversion, and assigns the result to the corresponding field in the

object representing the row. Then the code adds the object representing each user to the data structure representing the entire result set. If the query was changed to reorder, add, or remove fields in the result set, the application code would have to change also.

```
/* Verbose result-parsing code that doesn't use the JSON result set format */
for (var row: response.getRecords()) {
    var user = User.builder()
        .userId(row.get(0).getLongValue())
        .firstName(row.get(1).getStringValue())
        .lastName(row.get(2).getStringValue())
        .dob(Instant.parse(row.get(3).getStringValue()))
        .build();
    result.add(user);
}
```

The following sample values show the values of the `formattedRecords` field for result sets with different numbers of columns, column aliases, and column data types.

If the result set includes multiple rows, each row is represented as an object that is an array element. Each column in the result set becomes a key in the object. The keys are repeated for each row in the result set. Thus, for result sets consisting of many rows and columns, you might need to define short column aliases to avoid exceeding the length limit for the entire response.

This example works with the following schema and sample data:

```
create table sample_names (id int, name varchar(128));
insert into sample_names values (0, "Jane"), (1, "Mohan"), (2, "Maria"), (3, "Bruce"), (4, "Jasmine");
```

```
[{"id":0,"name":"Jane"}, {"id":1,"name":"Mohan"}, {"id":2,"name":"Maria"}, {"id":3,"name":"Bruce"}, {"id":4,"name":"Jasmine"}]
```

If a column in the result set is defined as an expression, the text of the expression becomes the JSON key. Thus, it's typically convenient to define a descriptive column alias for each expression in the select list of the query. For example, the following query includes expressions such as function calls and arithmetic operations in its select list.

```
select count(*), max(id), 4+7 from sample_names;
```

Those expressions are passed through to the JSON result set as keys.

```
[{"count(*)":5, "max(id)":4, "4+7":11}]
```

Adding `AS` columns with descriptive labels makes the keys simpler to interpret in the JSON result set.

```
select count(*) as rows, max(id) as largest_id, 4+7 as addition_result from sample_names;
```

With the revised SQL query, the column labels defined by the `AS` clauses are used as the key names.

```
[{"rows":5, "largest_id":4, "addition_result":11}]
```

The value for each key-value pair in the JSON string can be a quoted string. The string might contain unicode characters. If the string contains escape sequences or the `"` or `\` characters, those characters are preceded by backslash escape characters. The following examples of JSON strings demonstrate these possibilities. For example, the `string_with_escape_sequences` result contains the special characters backspace, newline, carriage return, tab, form feed, and `\``.

```
[{"quoted_string":"hello"}]  
[{"unicode_string":="#">  
[{"string_with_escape_sequences":"\b \n \r \t \f \\ ''"}]
```

The value for each key-value pair in the JSON string can also represent a number. The number might be an integer, a floating-point value, a negative value, or a value represented as exponential notation. The following examples of JSON strings demonstrate these possibilities.

```
[{"integer_value":17}][{"float_value":10.0}][{"negative_value":-9223372036854775808,"positive_value":9223372036854775807}][{"very_small_floating_point_value":4.9E-324,"very_large_floating_point_value":1.7976931348623157E308}]
```

Boolean and null values are represented with the unquoted special keywords `true`, `false`, and `null`. The following examples of JSON strings demonstrate these possibilities.

```
[{"boolean_value_1":true,"boolean_value_2":false}][{"unknown_value":null}]
```

If you select a value of a BLOB type, the result is represented in the JSON string as a base64-encoded value. To convert the value back to its original representation, you can use the appropriate decoding function in your application's language. For example, in Java you call the function `Base64.getDecoder().decode()`. The following sample output shows the result of selecting a BLOB value of `hello world` and returning the result set as a JSON string.

```
[{"blob_column":"aGVsbG8gd29ybGQ="}]
```

The following Python example shows how to access the values from the result of a call to the Python `execute_statement` function. The result set is a string value in the field `response['formattedRecords']`. The code turns the JSON string into a data structure by calling the `json.loads` function. Then each row of the result set is a list element within the data structure, and within each row you can refer to each field of the result set by name.

```
import json

result = json.loads(response['formattedRecords'])
print (result[0]["id"])
```

The following Javascript example shows how to access the values from the result of a call to the Javascript `executeStatement` function. The result set is a string value in the field `response.formattedRecords`. The code turns the JSON string into a data structure by calling the `JSON.parse` function. Then each row of the result set is an array element within the data structure, and within each row you can refer to each field of the result set by name.

```
<script>
  const result = JSON.parse(response.formattedRecords);
  document.getElementById("display").innerHTML = result[0].id;
</script>
```

## 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 \(p. 1609\)](#)
- [Packet for query is too large \(p. 1609\)](#)
- [Database response exceeded size limit \(p. 1609\)](#)
- [HttpEndpoint is not enabled for cluster <cluster\\_ID> \(p. 1609\)](#)

## 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 appended to the error message, and is one of the following:

- Transaction may be expired.

Make sure that each transactional call runs within three minutes of the previous one.

It's also possible that the specified transaction ID wasn't created by a [BeginTransaction](#) call. Make sure that your call has a valid transaction ID.

- One previous call resulted in a termination of your transaction.

The transaction was already ended by your [CommitTransaction](#) or [RollbackTransaction](#) call.

- Transaction has been aborted due to an error from a previous call.

Check whether your previous calls have thrown any exceptions.

For information about running transactions, see [Calling the Data API \(p. 1588\)](#).

## 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>

Check the following potential causes for this issue:

- The Data API isn't enabled for the Aurora Serverless v1 DB cluster. To use the Data API with an Aurora Serverless v1 DB cluster, the Data API must be enabled for the DB cluster. For information about enabling the Data API, see [Enabling the Data API \(p. 1584\)](#).
- The DB cluster was renamed after the Data API was enabled for it. In that case, turn off the Data API for that cluster and then enable it again.

- The ARN you specified doesn't precisely match the ARN of the cluster. Check that the ARN returned from another source or constructed by program logic matches the ARN of the cluster exactly. For example, make sure that the ARN you use has the correct letter case for all alphabetic characters.

If the Data API has not been enabled for the DB cluster, enable it. Make sure that it's an Aurora Serverless v1 cluster. Currently, you can't use the Data API with Aurora Serverless v2.

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 \(p. 1584\)](#).

## 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": "AKIAI44QH8DHBEXAMPLE",  
        "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 v1

With the query editor for Aurora Serverless v1, you can run SQL queries in the RDS console. You can run any valid SQL statement on the Aurora Serverless v1 DB cluster, including data manipulation and data definition statements.

The query editor requires an Aurora Serverless v1 DB cluster with the Data API enabled. For information about creating an Aurora Serverless v1 DB cluster with the Data API enabled, see [Using the Data API for Aurora Serverless v1 \(p. 1581\)](#).

**Note**

The Data API and query editor aren't supported for Aurora Serverless v2.

## Availability of the query editor

The query editor is available only for Aurora Serverless v1 DB clusters using specific Aurora MySQL and Aurora PostgreSQL versions that support the Data API. The query editor is available only in AWS Regions where the Data API is supported. For more information, see [Data API for Aurora Serverless v1 \(p. 33\)](#).

## 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.

**Note**

Make sure to use the same user name and password when you create the IAM user as you did for the database user, such as the master user name and password. For more information, see [Creating an IAM user in your AWS account](#) in the *AWS Identity and Access Management User Guide*.

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: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 v1 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 v1 DB clusters that you want to query.
3. In the navigation pane, choose **Databases**.
4. Choose the Aurora Serverless v1 DB cluster that you want to run SQL queries on.
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

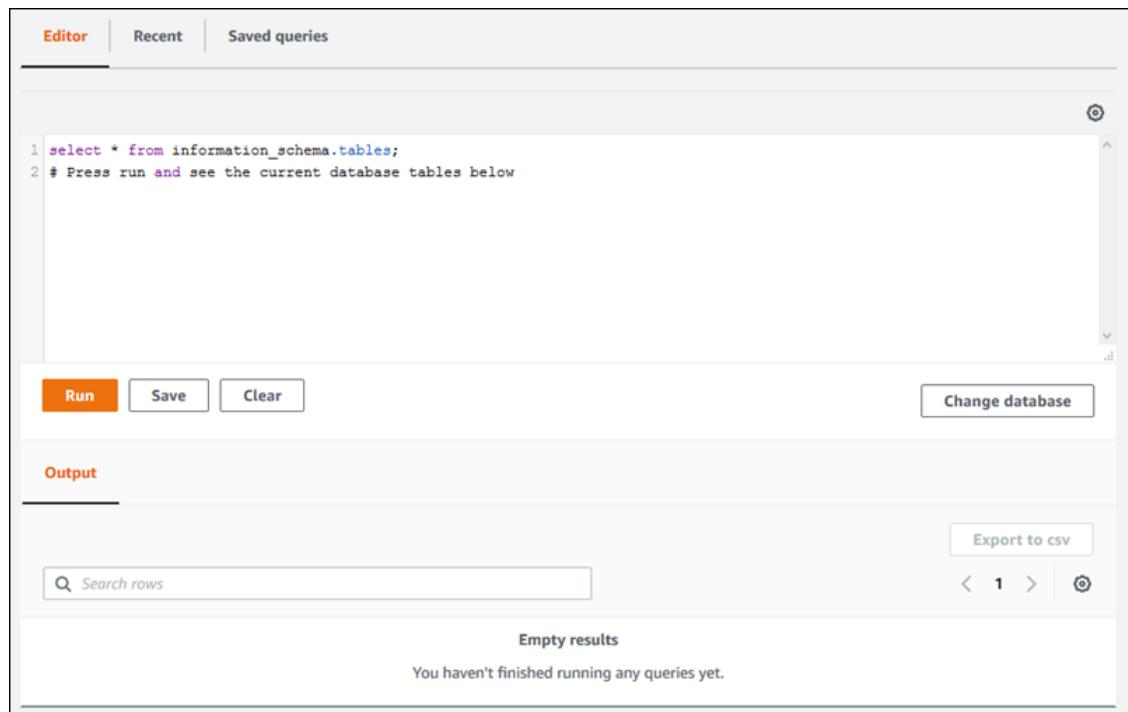
**Cancel** **Connect to database**

6. Enter the following information:
  - a. For **Database instance or cluster**, choose the Aurora Serverless v1 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.



```
1 select * from information_schema.tables;
2 # Press run and see the current database tables below
```

Run Save Clear Change database

Output

Search rows Export to csv

Empty results

You haven't finished running any queries yet.

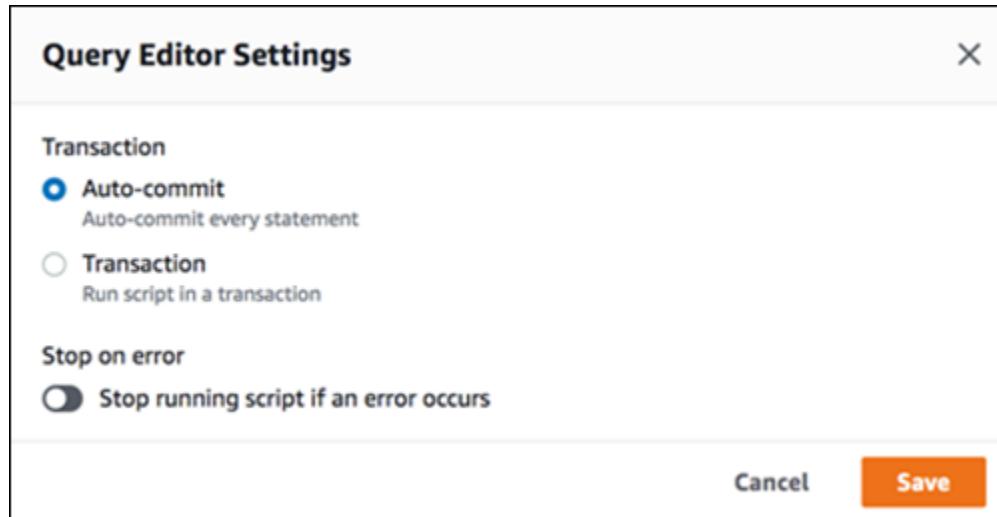
Each SQL statement can commit automatically, or you can run SQL statements in a script as part of a transaction. To control this behavior, choose the gear icon above the query window.



```
1 select * from information_schema.tables;
2 # Press run and see the current database tables below
```

Save Clear Change database

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 \(p. 1618\)](#)
- [CreateQueryHistory \(p. 1618\)](#)
- [CreateTab \(p. 1618\)](#)

- [DeleteFavoriteQueries \(p. 1618\)](#)
- [DeleteQueryHistory \(p. 1618\)](#)
- [DeleteTab \(p. 1618\)](#)
- [DescribeFavoriteQueries \(p. 1618\)](#)
- [DescribeQueryHistory \(p. 1618\)](#)
- [DescribeTabs \(p. 1618\)](#)
- [GetQueryString \(p. 1619\)](#)
- [UpdateFavoriteQuery \(p. 1619\)](#)
- [UpdateQueryHistory \(p. 1619\)](#)
- [UpdateTab \(p. 1619\)](#)

## 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.

# 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 <a href="#">Best practices with Amazon Aurora MySQL (p. 940)</a>
Amazon Aurora PostgreSQL	See <a href="#">Best practices with Amazon Aurora PostgreSQL (p. 1200)</a>

## Note

For common recommendations for Aurora, see [Viewing Amazon Aurora recommendations \(p. 444\)](#).

## Topics

- [Basic operational guidelines for Amazon Aurora \(p. 1620\)](#)
- [DB instance RAM recommendations \(p. 1620\)](#)
- [Monitoring Amazon Aurora \(p. 1621\)](#)
- [Working with DB parameter groups and DB cluster parameter groups \(p. 1621\)](#)
- [Amazon Aurora best practices presentation video \(p. 1621\)](#)

# 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, db.t3, or db.t4g instance classes.

For more information, see [Amazon CloudWatch metrics for Amazon Aurora \(p. 525\)](#).

## 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 \(p. 427\)](#).

## 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 \(p. 368\)](#).

## 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 \(p. 274\)](#), [Amazon Aurora MySQL performance enhancements \(p. 651\)](#), [Managing Amazon Aurora PostgreSQL \(p. 1143\)](#), and [Monitoring DB load with Performance Insights on Amazon Aurora \(p. 461\)](#).

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 \(p. 305\)](#) 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 \(p. 73\)](#), 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 \(p. 73\)](#). 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 \(p. 127\)](#).

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 \(p. 207\)](#).

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 \(p. 690\)](#) or [Migrating data to Amazon Aurora with PostgreSQL compatibility \(p. 1048\)](#) 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 \(p. 1628\)](#).

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 \(p. 217\)](#). For the configuration settings that are or aren't applicable to Aurora clusters, see [Aurora MySQL configuration parameters \(p. 949\)](#) or [Amazon Aurora PostgreSQL parameters \(p. 1369\)](#) 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 \(p. 207\)](#). 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 \(p. 1627\)](#).

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 \(p. 373\)](#) and [Restoring from a DB cluster snapshot \(p. 375\)](#).

## 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.

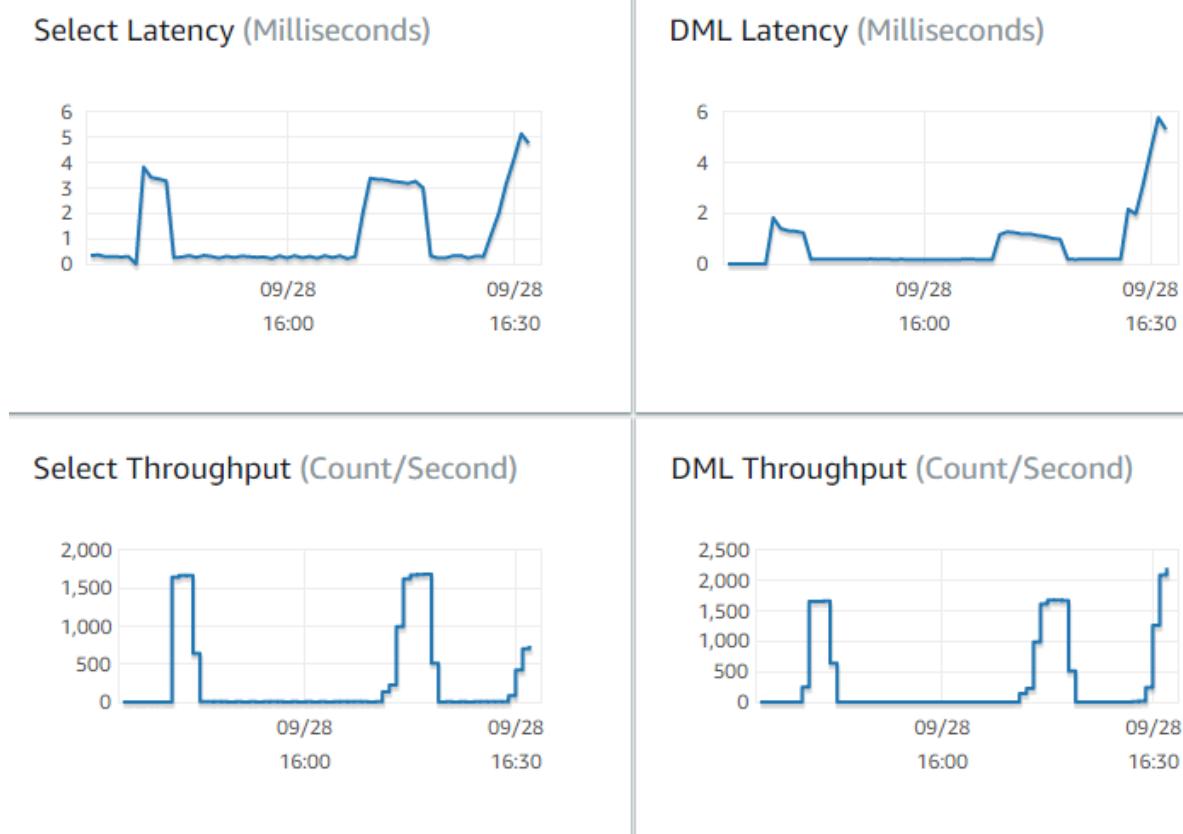
To have more choices on the **Monitoring** tab, enable Enhanced Monitoring and Performance Insights in the cluster settings. You can also enable those choices later if you didn't choose them when setting up the cluster.

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 \(p. 738\)](#).

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 \(p. 274\)](#), [Amazon Aurora MySQL performance enhancements \(p. 651\)](#), [Managing Amazon Aurora PostgreSQL \(p. 1143\)](#), and [Monitoring DB load with Performance Insights on Amazon Aurora \(p. 461\)](#). 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 VPC VPCs and Amazon Aurora \(p. 1729\)](#).
- 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 \(p. 87\)](#).

- 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 \(p. 1642\)](#).
- 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 \(p. 1638\)](#).
- 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 \(p. 681\)](#).

For information about security with Aurora PostgreSQL, see [Security with Amazon Aurora PostgreSQL \(p. 1018\)](#).

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 have to configure security only for your use cases. You don't have to configure security access for processes that Amazon Aurora manages. These include creating backups, automatic failover, 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 \(p. 1635\)](#)
- [Data protection in Amazon RDS \(p. 1637\)](#)
- [Identity and access management for Amazon Aurora \(p. 1653\)](#)
- [Logging and monitoring in Amazon Aurora \(p. 1711\)](#)
- [Compliance validation for Amazon Aurora \(p. 1714\)](#)
- [Resilience in Amazon Aurora \(p. 1715\)](#)
- [Infrastructure security in Amazon Aurora \(p. 1717\)](#)
- [Amazon RDS API and interface VPC endpoints \(AWS PrivateLink\) \(p. 1718\)](#)
- [Security best practices for Amazon Aurora \(p. 1720\)](#)
- [Controlling access with security groups \(p. 1721\)](#)
- [Master user account privileges \(p. 1723\)](#)
- [Using service-linked roles for Amazon Aurora \(p. 1724\)](#)
- [Amazon VPC VPCs and Amazon Aurora \(p. 1729\)](#)

## 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 \(p. 1636\)](#)
- [IAM database authentication \(p. 1636\)](#)
- [Kerberos authentication \(p. 1636\)](#)

## Password authentication

With *password authentication*, your database 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 \(p. 1683\)](#).

## 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 clusters.

You can make it possible for your database users to authenticate against DB clusters 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 \(p. 1035\)](#).

## 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 \(p. 1637\)](#)
- [Internetwork traffic privacy \(p. 1652\)](#)

## Protecting data using encryption

You can enable encryption for database resources. You can also encrypt connections to DB clusters.

### Topics

- [Encrypting Amazon Aurora resources \(p. 1638\)](#)

- [AWS KMS key management \(p. 1641\)](#)
- [Using SSL/TLS to encrypt a connection to a DB cluster \(p. 1642\)](#)
- [Rotating your SSL/TLS certificate \(p. 1644\)](#)

## 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 \(p. 1638\)](#)
- [Encrypting an Amazon Aurora DB cluster \(p. 1638\)](#)
- [Determining whether encryption is turned on for a DB cluster \(p. 1639\)](#)
- [Availability of Amazon Aurora encryption \(p. 1640\)](#)
- [Limitations of Amazon Aurora encrypted DB clusters \(p. 1640\)](#)

## 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](#).

## Encrypting an Amazon Aurora DB cluster

To encrypt a new DB cluster, choose **Enable encryption** on the console. For information on creating a DB cluster, see [Creating an Amazon Aurora DB cluster \(p. 127\)](#).

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, Aurora loses access when the KMS key isn't enabled, or when Aurora 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 DB cluster if it is. Restart the DB cluster using the AWS CLI command [start-db-cluster](#). Currently, you can't start a DB cluster in this state using the AWS Management Console.

If the DB cluster isn't recovered, then it 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 turn on 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

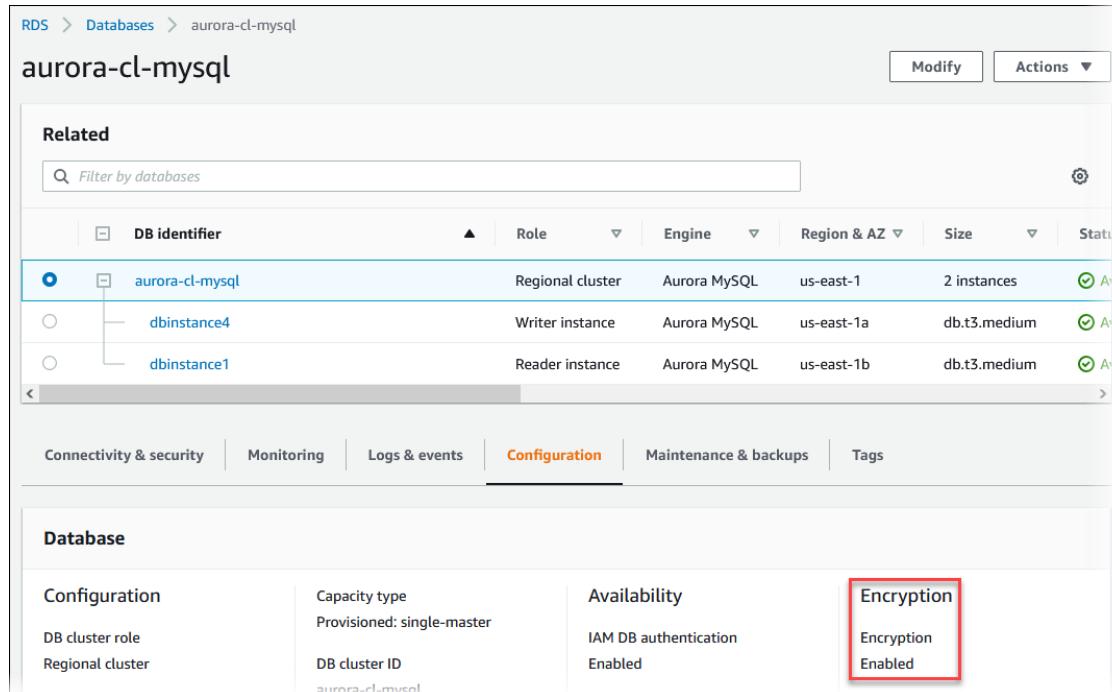
You can use the AWS Management Console, AWS CLI, or RDS API to determine whether encryption at rest is turned on for a DB cluster.

### 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 'Databases' page for the 'aurora-cl-mysql' cluster. The 'Configuration' tab is selected. In the 'Database' section, under the 'Encryption' heading, it says 'Encryption Enabled'. This section is highlighted with a red box.

DB identifier	Role	Engine	Region & AZ	Size	Status
aurora-cl-mysql	Regional cluster	Aurora MySQL	us-east-1	2 instances	<span>Enabled</span>
dbinstance4	Writer instance	Aurora MySQL	us-east-1a	db.t3.medium	<span>Enabled</span>
dbinstance1	Reader instance	Aurora MySQL	us-east-1b	db.t3.medium	<span>Enabled</span>

## 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 turn off 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 turn off 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 create an Aurora resource using a customer managed key, a user must have permissions to call the following operations on the customer managed key:

- kms>CreateGrant
- kms>DescribeKey

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 \(p. 681\)](#)
- [Security with Amazon Aurora PostgreSQL \(p. 1018\)](#)

### 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 \(p. 1435\)](#).

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 \(p. 1546\)](#).

## 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
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)	<a href="#">us-gov-east-1-bundle.pem</a>	<a href="#">us-gov-east-1-bundle.p7b</a>
AWS GovCloud (US-West)	<a href="#">us-gov-west-1-bundle.pem</a>	<a href="#">us-gov-west-1-bundle.p7b</a>

## 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 \(p. 1642\)](#).

### 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 \(p. 1435\)](#).

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 \(p. 1546\)](#).

### 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 \(p. 1644\)](#)
- [Updating your CA certificate by applying DB instance maintenance \(p. 1647\)](#)
- [Sample script for importing certificates into your trust store \(p. 1651\)](#)

## 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 \(p. 1642\)](#).

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 \(p. 687\)](#)
- [Updating applications to connect to Aurora PostgreSQL DB clusters using new SSL/TLS certificates \(p. 1032\)](#)

For a sample script that updates a trust store for a Linux operating system, see [Sample script for importing certificates into your trust store \(p. 1651\)](#).

**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 screenshot shows the AWS RDS console with the path: RDS > Databases > aurora-test > dbinstance1. The main title is "dbinstance1". On the right, there is a red box highlighting the "More" button. Below the title, the "Related" section displays a table of database instances:

DB identifier	Role	Engine	Region & AZ	Size
aurora-test	Regional	Aurora MySQL	us-east-1	3 instances
dbinstance4	Writer	Aurora MySQL	us-east-1a	db.r5.large
dbinstance1	Reader	Aurora MySQL	us-east-1b	db.r5.large
dbinstance2	Reader	Aurora MySQL	us-east-1b	db.r5.large

Below the table, there are tabs: Connectivity & security (selected), Monitoring, Logs & events, Configuration, Maintenance, and Tags.

The **Modify DB Instance** page appears.

- In the **Connectivity** section, choose **rds-ca-2019**.

**Certificate authority**  
Certificate authority for this DB instance

rds-ca-2019

rds-ca-2015

rds-ca-2019

EC2 instances and devices outside of the VPC hosting the DB instance will connect to the DB instances. You can add more VPC security groups that specify which EC2 instances and devices can connect to the DB instance.

- Choose **Continue** and check the summary of modifications.
- To apply the changes immediately, choose **Apply immediately**.

**Important**

Choosing this option restarts your database immediately.

- 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 \(p. 1642\)](#).
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 \(p. 687\)](#)
- [Updating applications to connect to Aurora PostgreSQL DB clusters using new SSL/TLS certificates \(p. 1032\)](#)

For a sample script that updates a trust store for a Linux operating system, see [Sample script for importing certificates into your trust store \(p. 1651\)](#).

**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.

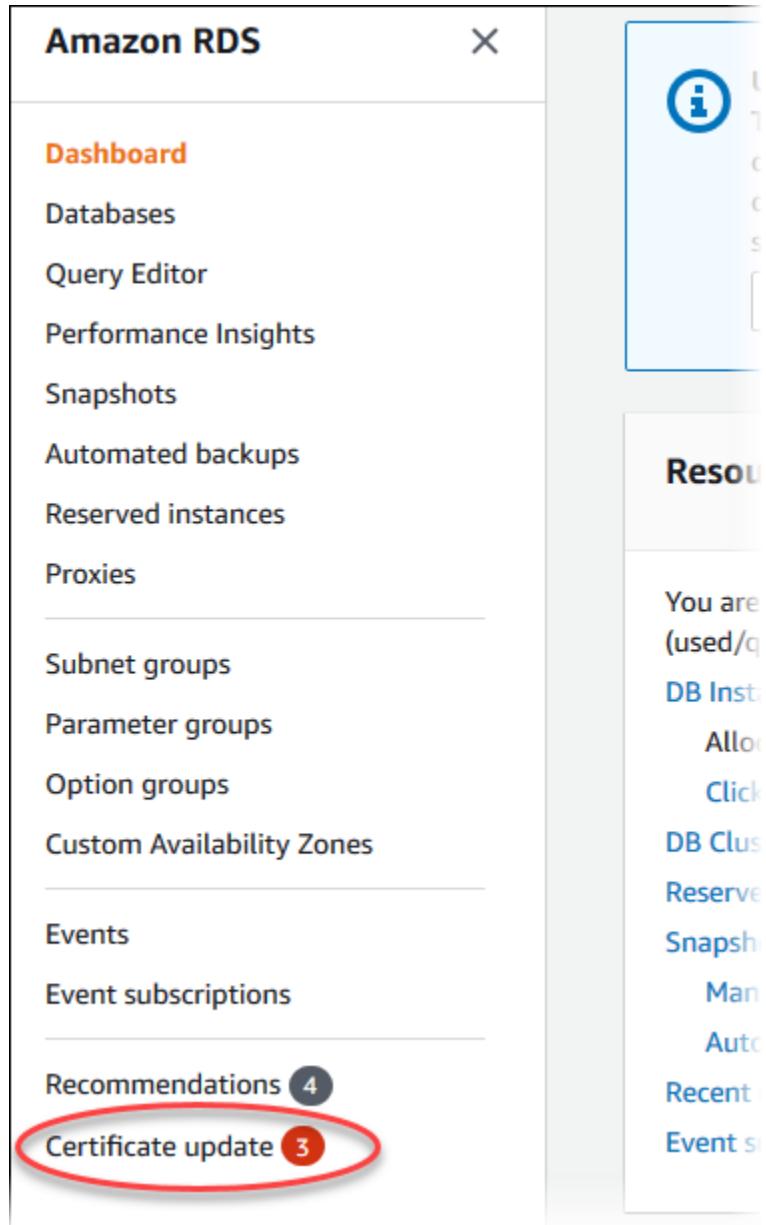
[Updating your CA certificate by applying maintenance to multiple DB instances](#)

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 \(p. 1647\)](#). 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 \(p. 1651\)](#)
- [Sample script for importing certificates on macOS \(p. 1651\)](#)

### 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://truststore.pki.rds.amazonaws.com/global/global-bundle.pem" > ${mydir}/
global-bundle.pem
awk 'split_after == 1 {n++;split_after=0} /-----END CERTIFICATE-----/ {split_after=1}{print
> "rds-ca-" n ".pem"}' < ${mydir}/global-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}/global-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://truststore.pki.rds.amazonaws.com/global/global-bundle.pem" > ${mydir}/
global-bundle.pem
split -p "-----BEGIN CERTIFICATE-----" ${mydir}/global-bundle.pem rds-ca-
for CERT in rds-ca-*; do
    alias=$(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}/global-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 for 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 Amazon RDS resources. IAM is an AWS service that you can use with no additional charge.

## Topics

- [Audience \(p. 1653\)](#)
- [Authenticating with identities \(p. 1653\)](#)
- [Managing access using policies \(p. 1655\)](#)
- [How Amazon Aurora works with IAM \(p. 1657\)](#)
- [Identity-based policy examples for Amazon Aurora \(p. 1662\)](#)
- [AWS managed policies for Amazon RDS \(p. 1673\)](#)
- [Amazon RDS updates to AWS managed policies \(p. 1679\)](#)
- [Preventing cross-service confused deputy problems \(p. 1681\)](#)
- [IAM database authentication \(p. 1683\)](#)
- [Troubleshooting Amazon Aurora identity and access \(p. 1710\)](#)

## Audience

How you use AWS Identity and Access Management (IAM) differs, depending on the work you do in Amazon 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 \(p. 1710\)](#).

**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 \(p. 1657\)](#).

**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 [Identity-based policy examples for Amazon Aurora \(p. 1662\)](#).

## 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 create an AWS account, you begin with one 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. Safeguard your root user credentials and use them to perform the tasks that only the root user can perform. For the complete list of tasks that require you to sign in as the root user, see [Tasks that require root user credentials](#) in the [AWS General Reference](#).

## 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 \(p. 1683\)](#).

## 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, a web identity provider, or the IAM Identity Center identity store. These identities are known as *federated identities*. To assign permissions to federated identities, you can create a role and define permissions for the role. When an external identity authenticates, the identity is associated with the role and is granted the permissions that are defined by it. If you use IAM Identity Center, you configure a permission set. IAM Identity Center correlates the permission set to a role in IAM to control what your identities can access after they authenticate. For more information about identity federation, see [Creating a role for a third-party Identity Provider](#) in the *IAM User Guide*. For more information about IAM Identity Center, see [What is IAM Identity Center?](#) in the *AWS IAM Identity Center (successor to AWS Single Sign-On) 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*.
- **Cross-service access** – Some AWS services use features in other AWS services. For example, when you make a call in a service, it's common for that service to run applications in Amazon EC2 or store objects in Amazon S3. A service might do this using the calling principal's permissions, using a service role, or using a service-linked role.
- **Principal permissions** – When you use an IAM user or role to perform actions in AWS, you are considered a principal. Policies grant permissions to a principal. When you use some services, you might perform an action that then triggers another action in a different service. In this case, you must have permissions to perform both actions. To see whether an action requires additional dependent actions in a policy, see [Actions, Resources, and Condition Keys for Amazon RDS](#) in the *Service Authorization Reference*.
- **Service role** – 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*.
- **Service-linked role** – A service-linked role is a type of service role that is linked to an AWS service. The service can assume the role to perform an action on your behalf. Service-linked roles appear in your IAM account and are owned by the service. An IAM administrator can view, but not edit the permissions for service-linked roles.
- **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*.

For information about AWS managed policies that are specific to Amazon Aurora, see [AWS managed policies for Amazon RDS \(p. 1673\)](#).

## 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*.

## How Amazon Aurora works with IAM

Before you use IAM to manage access to Amazon Aurora, you should understand what IAM features are available to use with Aurora.

### IAM features you can use with Amazon Aurora

IAM feature	Amazon Aurora support
Identity-based policies ( <a href="#">p. 1658</a> )	Yes
Resource-based policies ( <a href="#">p. 1658</a> )	No
Policy actions ( <a href="#">p. 1658</a> )	Yes
Policy resources ( <a href="#">p. 1659</a> )	Yes
Policy condition keys (service-specific) ( <a href="#">p. 1660</a> )	Yes
ACLs ( <a href="#">p. 1660</a> )	No
Attribute-based access control (ABAC) (tags in policies) ( <a href="#">p. 1661</a> )	Yes
Temporary credentials ( <a href="#">p. 1661</a> )	Yes
Principal permissions ( <a href="#">p. 1661</a> )	Yes
Service roles ( <a href="#">p. 1662</a> )	Yes
Service-linked roles ( <a href="#">p. 1662</a> )	Yes

To get a high-level view of how Amazon 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 \(\[p. 1658\]\(#\)\)](#)
- [Resource-based policies within Aurora \(\[p. 1658\]\(#\)\)](#)
- [Policy actions for Aurora \(\[p. 1658\]\(#\)\)](#)
- [Policy resources for Aurora \(\[p. 1659\]\(#\)\)](#)
- [Policy condition keys for Aurora \(\[p. 1660\]\(#\)\)](#)
- [Access control lists \(ACLs\) in Aurora \(\[p. 1660\]\(#\)\)](#)
- [Attribute-based access control \(ABAC\) in policies with Aurora tags \(\[p. 1661\]\(#\)\)](#)
- [Using temporary credentials with Aurora \(\[p. 1661\]\(#\)\)](#)
- [Cross-service principal permissions for Aurora \(\[p. 1661\]\(#\)\)](#)
- [Service roles for Aurora \(\[p. 1662\]\(#\)\)](#)
- [Service-linked roles for Aurora \(\[p. 1662\]\(#\)\)](#)

## Aurora identity-based policies

Supports identity-based policies	Yes
----------------------------------	-----

Identity-based policies are JSON permissions policy documents that you can attach to an identity, such as an IAM user, group of users, or role. These policies control what actions users and roles 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*.

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. You can't specify the principal in an identity-based policy because it applies to the user or role to which it is attached. To learn about all of the elements that you can use in a JSON policy, see [IAM JSON policy elements reference](#) in the *IAM User Guide*.

### Identity-based policy examples for Aurora

To view examples of Aurora identity-based policies, see [Identity-based policy examples for Amazon Aurora](#) (p. 1662).

## Resource-based policies within Aurora

Supports resource-based policies	No
----------------------------------	----

Resource-based policies are JSON policy documents that you attach to a resource. Examples of resource-based policies are IAM *role trust policies* and Amazon S3 *bucket policies*. In services that support resource-based policies, service administrators can use them to control access to a specific resource. For the resource where the policy is attached, the policy defines what actions a specified principal can perform on that resource and under what conditions. You must [specify a principal](#) in a resource-based policy. Principals can include accounts, users, roles, federated users, or AWS services.

To enable cross-account access, you can specify an entire account or IAM entities in another account as the principal in a resource-based policy. Adding a cross-account principal to a resource-based policy is only half of establishing the trust relationship. When the principal and the resource are in different AWS accounts, an IAM administrator in the trusted account must also grant the principal entity (user or role) permission to access the resource. They grant permission by attaching an identity-based policy to the entity. However, if a resource-based policy grants access to a principal in the same account, no additional identity-based policy is required. For more information, see [How IAM roles differ from resource-based policies](#) in the *IAM User Guide*.

## Policy actions for Aurora

Supports policy actions	Yes
-------------------------	-----

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The **Action** element of a JSON policy describes the actions that you can use to allow or deny access in a policy. Policy actions usually have the same name as the associated AWS API operation. There are some exceptions, such as *permission-only actions* that don't have a matching API operation. There are also some operations that require multiple actions in a policy. These additional actions are called *dependent actions*.

Include actions 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*.

## Policy resources for Aurora

Supports policy resources	Yes
---------------------------	-----

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The Resource JSON policy element specifies the object or objects to which the action applies. Statements must include either a `Resource` or a `NotResource` element. As a best practice, specify a resource using its [Amazon Resource Name \(ARN\)](#). You can do this for actions that support a specific resource type, known as *resource-level permissions*.

For actions that don't support resource-level permissions, such as listing operations, use a wildcard (\*) to indicate that the statement applies to all resources.

```
"Resource": "*"
```

The DB instance resource has the following Amazon Resource Name (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, can't be performed on a specific resource. In those cases, 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](#).

## Policy condition keys for Aurora

Supports service-specific policy condition keys	Yes
---	-----

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The `Condition` element (or *Condition block*) lets you specify conditions in which a statement is in effect. The `Condition` element is optional. You can create 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*.

AWS supports global condition keys and service-specific condition keys. To see all AWS global condition keys, see [AWS global condition context keys](#) 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](#).

## Access control lists (ACLs) in Aurora

Supports access control lists (ACLs)	No
--------------------------------------	----

Access control lists (ACLs) control which principals (account members, users, or roles) have permissions to access a resource. ACLs are similar to resource-based policies, although they do not use the JSON policy document format.

## Attribute-based access control (ABAC) in policies with Aurora tags

Supports attribute-based access control (ABAC) tags in policies	Yes
---	-----

Attribute-based access control (ABAC) is an authorization strategy that defines permissions based on attributes. In AWS, these attributes are called *tags*. You can attach tags to IAM entities (users or roles) and to many AWS resources. Tagging entities and resources is the first step of ABAC. Then you design ABAC policies to allow operations when the principal's tag matches the tag on the resource that they are trying to access.

ABAC is helpful in environments that are growing rapidly and helps with situations where policy management becomes cumbersome.

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 about ABAC, see [What is ABAC?](#) in the *IAM User Guide*. To view a tutorial with steps for setting up ABAC, see [Use attribute-based access control \(ABAC\)](#) in the *IAM User Guide*.

For more information about tagging Aurora resources, see [Specifying conditions: Using custom tags \(p. 1669\)](#). 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 \(p. 1666\)](#).

## Using temporary credentials with Aurora

Supports temporary credentials	Yes
--------------------------------	-----

Some AWS services don't work when you sign in using temporary credentials. For additional information, including which AWS services work with temporary credentials, see [AWS services that work with IAM](#) in the *IAM User Guide*.

You are using temporary credentials if you sign in to the AWS Management Console using any method except a user name and password. For example, when you access AWS using your company's single sign-on (SSO) link, that process automatically creates temporary credentials. You also automatically create temporary credentials when you sign in to the console as a user and then switch roles. For more information about switching roles, see [Switching to a role \(console\)](#) in the *IAM User Guide*.

You can manually create temporary credentials using the AWS CLI or AWS API. You can then use those temporary credentials to access AWS. AWS recommends that you dynamically generate temporary credentials instead of using long-term access keys. For more information, see [Temporary security credentials in IAM](#).

## Cross-service principal permissions for Aurora

Supports principal permissions	Yes
--------------------------------	-----

When you use an IAM user or role to perform actions in AWS, you are considered a principal. Policies grant permissions to a principal. When you use some services, you might perform an action that then triggers another action in a different service. In this case, you must have permissions to perform both actions. To see whether an action requires additional dependent actions in a policy, see [Actions, Resources, and Condition Keys for Amazon RDS](#) in the *Service Authorization Reference*.

## Service roles for Aurora

Supports service roles	Yes
------------------------	-----

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*.

### Warning

Changing the permissions for a service role might break Aurora functionality. Edit service roles only when Aurora provides guidance to do so.

## Service-linked roles for Aurora

Supports service-linked roles	Yes
-------------------------------	-----

A service-linked role is a type of service role that is linked to an AWS service. The service can assume the role to perform an action on your behalf. Service-linked roles appear in your IAM account and are owned by the service. An IAM administrator can view, but not edit the permissions for service-linked roles.

For details about using Aurora service-linked roles, see [Using service-linked roles for Amazon Aurora](#) (p. 1724).

## Identity-based policy examples for Amazon Aurora

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. 1663)
- [Using the Aurora console](#) (p. 1663)
- [Allow users to view their own permissions](#) (p. 1664)
- [Allow a user to create DB instances in an AWS account](#) (p. 1664)
- [Permissions required to use the console](#) (p. 1665)
- [Allow a user to perform any describe action on any RDS resource](#) (p. 1666)
- [Allow a user to create a DB instance that uses the specified DB parameter group and subnet group](#) (p. 1666)
- [Grant permission for actions on a resource with a specific tag with two different values](#) (p. 1666)
- [Prevent a user from deleting a DB instance](#) (p. 1667)

- Deny all access to a resource (p. 1667)
- Example policies: Using condition keys (p. 1667)
- Specifying conditions: Using custom tags (p. 1669)

## Policy best practices

Identity-based policies determine whether someone can create, access, or delete Amazon RDS 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 with AWS managed policies and move toward least-privilege permissions** – To get started granting permissions to your users and workloads, use the *AWS managed policies* that grant permissions for many common use cases. They are available in your AWS account. We recommend that you reduce permissions further by defining AWS customer managed policies that are specific to your use cases. For more information, see [AWS managed policies](#) or [AWS managed policies for job functions](#) in the *IAM User Guide*.
- **Apply least-privilege permissions** – When you set permissions with IAM policies, grant only the permissions required to perform a task. You do this by defining the actions that can be taken on specific resources under specific conditions, also known as *least-privilege permissions*. For more information about using IAM to apply permissions, see [Policies and permissions in IAM](#) in the *IAM User Guide*.
- **Use conditions in IAM policies to further restrict access** – You can add a condition to your policies to limit access to actions and resources. For example, you can write a policy condition to specify that all requests must be sent using SSL. You can also use conditions to grant access to service actions if they are used through a specific AWS service, such as AWS CloudFormation. For more information, see [IAM JSON policy elements: Condition](#) in the *IAM User Guide*.
- **Use IAM Access Analyzer to validate your IAM policies to ensure secure and functional permissions** – IAM Access Analyzer validates new and existing policies so that the policies adhere to the IAM policy language (JSON) and IAM best practices. IAM Access Analyzer provides more than 100 policy checks and actionable recommendations to help you author secure and functional policies. For more information, see [IAM Access Analyzer policy validation](#) in the *IAM User Guide*.
- **Require multi-factor authentication (MFA)** – If you have a scenario that requires IAM users or root users in your account, turn on MFA for additional security. To require MFA when API operations are called, add MFA conditions to your policies. For more information, see [Configuring MFA-protected API access](#) in the *IAM User Guide*.

For more information about best practices in IAM, see [Security best practices in IAM](#) 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 allow you to list and view details about the Amazon Aurora resources in your AWS account. If you create an identity-based policy that is more restrictive than the minimum required permissions, the console won't function as intended for entities (IAM users or roles) with that policy.

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.

To ensure that those entities can still use the Aurora console, also attach the following AWS managed policy to the entities.

AmazonRDSReadOnlyAccess
-------------------------

For more information, see [Adding permissions to a user](#) in the *IAM User Guide*.

## 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:subnetgroup: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 \(p. 360\)](#).

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 [Policy condition keys for Aurora \(p. 1660\)](#). 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 \(p. 57\)](#).

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 \(p. 1655\)](#).

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` API operations 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"  
                }  
            }  
        }  
    ]  
}
```

```
        "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

**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
db-tag	DB instances, including read replicas
snapshot-tag	DB snapshots
ri-tag	Reserved DB instances
og-tag	DB option groups
pg-tag	DB parameter groups

RDS tag identifier	Applies to
subgrp-tag	DB subnet groups
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 \(p. 352\)](#).

#### 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 \(p. 360\)](#).

#### 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` API operations 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 \(p. 360\)](#).

```
{  
    "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"  
                }  
            }  
        }  
    ]  
}
```

## AWS managed policies for Amazon RDS

To add permissions to users, groups, and roles, it's easier to use AWS managed policies than to write policies yourself. It takes time and expertise to [create IAM customer managed policies](#) that provide your team with only the permissions they need. To get started quickly, you can use our AWS managed policies. These policies cover common use cases and are available in your AWS account. For more information about AWS managed policies, see [AWS managed policies](#) in the *IAM User Guide*.

AWS services maintain and update AWS managed policies. You can't change the permissions in AWS managed policies. Services occasionally add additional permissions to an AWS managed policy to support new features. This type of update affects all identities (users, groups, and roles) where the policy is attached. Services are most likely to update an AWS managed policy when a new feature is launched or when new operations become available. Services don't remove permissions from an AWS managed policy, so policy updates don't break your existing permissions.

Additionally, AWS supports managed policies for job functions that span multiple services. For example, the `ReadOnlyAccess` AWS managed policy provides read-only access to all AWS services and resources. When a service launches a new feature, AWS adds read-only permissions for new operations and resources. For a list and descriptions of job function policies, see [AWS managed policies for job functions](#) in the *IAM User Guide*.

### Topics

- [AWS managed policy: AmazonRDSReadOnlyAccess \(p. 1673\)](#)
- [AWS managed policy: AmazonRDSFullAccess \(p. 1674\)](#)
- [AWS managed policy: AmazonRDSDataFullAccess \(p. 1675\)](#)
- [AWS managed policy: AmazonRDSEnhancedMonitoringRole \(p. 1676\)](#)
- [AWS managed policy: AmazonRDSPerformanceInsightsReadOnly \(p. 1677\)](#)
- [AWS managed policy: AmazonRDSDirectoryServiceAccess \(p. 1678\)](#)
- [AWS managed policy: AmazonRDSServiceRolePolicy \(p. 1678\)](#)

## AWS managed policy: AmazonRDSReadOnlyAccess

This policy allows read-only access to Amazon RDS through the AWS Management Console.

### Permissions details

This policy includes the following permissions:

- `rds` – Allows principals to describe Amazon RDS resources and list the tags for Amazon RDS resources.
- `cloudwatch` – Allows principals to get Amazon CloudWatch metric statistics.
- `ec2` – Allows principals to describe Availability Zones and networking resources.
- `logs` – Allows principals to describe CloudWatch Logs log streams of log groups, and get CloudWatch Logs log events.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Action": [  
                "rds:Describe*",  
                "rds>ListTagsForResource",  
                "ec2:DescribeAccountAttributes",  
                "ec2:DescribeAvailabilityZones",  
                "ec2:DescribeInternetGateways",  
                "logs:GetLogEvents",  
                "logs:DescribeLogStreams",  
                "logs:CreateLogStream",  
                "logs:PutLogEvents"  
            ],  
            "Effect": "Allow",  
            "Resource": "*"  
        }  
    ]  
}
```

```
        "ec2:DescribeSecurityGroups",
        "ec2:DescribeSubnets",
        "ec2:DescribeVpcAttribute",
        "ec2:DescribeVpcs"
    ],
    "Effect": "Allow",
    "Resource": "*"
},
{
    "Action": [
        "cloudwatch:GetMetricStatistics",
        "logs:DescribeLogStreams",
        "logs:GetLogEvents"
    ],
    "Effect": "Allow",
    "Resource": "*"
}
]
```

## AWS managed policy: AmazonRDSFullAccess

This policy provides full access to Amazon RDS through the AWS Management Console.

### Permissions details

This policy includes the following permissions:

- **rds** – Allows principals full access to Amazon RDS.
- **application-autoscaling** – Allows principals describe and manage Application Auto Scaling scaling targets and policies.
- **cloudwatch** – Allows principals get CloudWatch metric statics and manage CloudWatch alarms.
- **ec2** – Allows principals to describe Availability Zones and networking resources.
- **logs** – Allows principals to describe CloudWatch Logs log streams of log groups, and get CloudWatch Logs log events.
- **outposts** – Allows principals to get AWS Outposts instance types.
- **pi** – Allows principals to get Performance Insights metrics.
- **sns** – Allows principals to Amazon Simple Notification Service (Amazon SNS) subscriptions and topics, and to publish Amazon SNS messages.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "rds:*",
                "application-autoscaling>DeleteScalingPolicy",
                "application-autoscaling>DeregisterScalableTarget",
                "application-autoscaling>DescribeScalableTargets",
                "application-autoscaling>DescribeScalingActivities",
                "application-autoscaling>DescribeScalingPolicies",
                "application-autoscaling>PutScalingPolicy",
                "application-autoscaling>RegisterScalableTarget",
                "cloudwatch:DescribeAlarms",
                "cloudwatch:GetMetricStatistics",
                "cloudwatch:PutMetricAlarm",
                "cloudwatch>DeleteAlarms",
                "ec2:DescribeAccountAttributes",
                "ec2:DescribeAvailabilityZones",
                "ec2:DescribeRegions",
                "ec2:DescribeSubnets",
                "ec2:DescribeVpcAttribute",
                "ec2:DescribeVpcs"
            ],
            "Effect": "Allow",
            "Resource": "*"
        }
    ]
}
```

```
"ec2:DescribeCoipPools",
"ec2:DescribeInternetGateways",
"ec2:DescribeLocalGatewayRouteTablePermissions",
"ec2:DescribeLocalGatewayRouteTables",
"ec2:DescribeLocalGatewayRouteTableVpcAssociations",
"ec2:DescribeLocalGateways",
"ec2:DescribeSecurityGroups",
"ec2:DescribeSubnets",
"ec2:DescribeVpcAttribute",
"ec2:DescribeVpcs",
"ec2:GetCoipPoolUsage",
"sns>ListSubscriptions",
"sns>ListTopics",
"sns>Publish",
"logs>DescribeLogStreams",
"logs>GetLogEvents",
"outposts>GetOutpostInstanceTypes"
],
"Effect": "Allow",
"Resource": "*"
},
{
"Action": "pi:*",
"Effect": "Allow",
"Resource": "arn:aws:pi:*::metrics/rds/*"
},
{
"Action": "iam>CreateServiceLinkedRole",
"Effect": "Allow",
"Resource": "*",
"Condition": {
"StringLike": {
"iam:AWSPropertyName": [
"rds.amazonaws.com",
"rds.application-autoscaling.amazonaws.com"
]
}
}
}
]
```

## AWS managed policy: AmazonRDSDDataFullAccess

This policy allows full access to use the Data API and the query editor on Aurora Serverless clusters in a specific AWS account. This policy allows the AWS account to get the value of a secret from AWS Secrets Manager.

You can attach the [AmazonRDSDDataFullAccess](#) policy to your IAM identities.

### Permissions details

This policy includes the following permissions:

- **dbqms** – Allows principals to access, create, delete, describe, and update queries. 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.
- **rds-data** – Allows principals to run SQL statements on Aurora Serverless databases.
- **secretsmanager** – Allows principals to get the value of a secret from AWS Secrets Manager.

{

```

    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "SecretsManagerDbCredentialsAccess",
            "Effect": "Allow",
            "Action": [
                "secretsmanager:GetSecretValue",
                "secretsmanager:PutResourcePolicy",
                "secretsmanager:PutSecretValue",
                "secretsmanager>DeleteSecret",
                "secretsmanager:DescribeSecret",
                "secretsmanager:TagResource"
            ],
            "Resource": "arn:aws:secretsmanager:*:secret:rds-db-credentials/*"
        },
        {
            "Sid": "RDSDDataServiceAccess",
            "Effect": "Allow",
            "Action": [
                "dbqms>CreateFavoriteQuery",
                "dbqms:DescribeFavoriteQueries",
                "dbqms:UpdateFavoriteQuery",
                "dbqms>DeleteFavoriteQueries",
                "dbqms:GetQueryString",
                "dbqms>CreateQueryHistory",
                "dbqms:DescribeQueryHistory",
                "dbqms:UpdateQueryHistory",
                "dbqms>DeleteQueryHistory",
                "rds-data:ExecuteSql",
                "rds-data:ExecuteStatement",
                "rds-data:BatchExecuteStatement",
                "rds-data:BeginTransaction",
                "rds-data:CommitTransaction",
                "rds-data:RollbackTransaction",
                "secretsmanager>CreateSecret",
                "secretsmanager>ListSecrets",
                "secretsmanager:GetRandomPassword",
                "tag:GetResources"
            ],
            "Resource": "*"
        }
    ]
}

```

## AWS managed policy: AmazonRDSEnhancedMonitoringRole

This policy provides access to Amazon CloudWatch Logs for Amazon RDS Enhanced Monitoring.

### Permissions details

This policy includes the following permissions:

- logs – Allows principals to create CloudWatch Logs log groups and retention policies, and to create and describe CloudWatch Logs log streams of log groups. It also allows principals to put and get CloudWatch Logs log events.

```

{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "EnableCreationAndManagementOfRDSCloudwatchLogGroups",
            "Effect": "Allow",

```

```

        "Action": [
            "logs:CreateLogGroup",
            "logs:PutRetentionPolicy"
        ],
        "Resource": [
            "arn:aws:logs:*:log-group:RDS*"
        ]
    },
    {
        "Sid": "EnableCreationAndManagementOfRDSCloudwatchLogStreams",
        "Effect": "Allow",
        "Action": [
            "logs:CreateLogStream",
            "logs:PutLogEvents",
            "logs:DescribeLogStreams",
            "logs:GetLogEvents"
        ],
        "Resource": [
            "arn:aws:logs:*:log-group:RDS*:log-stream:*
        ]
    }
]
}

```

## AWS managed policy: [AmazonRDSPerformanceInsightsReadOnly](#)

This policy provides read-only access to Amazon RDS Performance Insights for Amazon RDS DB instances and Amazon Aurora DB clusters.

### Permissions details

This policy includes the following permissions:

- **rds** – Allows principals to describe Amazon RDS DB instances and Amazon Aurora DB clusters.
- **pi** – Allows principals make calls to the Amazon RDS Performance Insights API and access Performance Insights metrics.

```

{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "rds:DescribeDBInstances",
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": "rds:DescribeDBClusters",
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": "pi:DescribeDimensionKeys",
            "Resource": "arn:aws:pi:metrics/rds/*"
        },
        {
            "Effect": "Allow",
            "Action": "pi:GetDimensionKeyDetails",
            "Resource": "arn:aws:pi:metrics/rds/*"
        },
    ]
}

```

```
{  
    "Effect": "Allow",  
    "Action": "pi:GetResourceMetadata",  
    "Resource": "arn:aws:pi:*.*:metrics/rds/*"  
},  
{  
    "Effect": "Allow",  
    "Action": "pi:GetResourceMetrics",  
    "Resource": "arn:aws:pi:*.*:metrics/rds/*"  
},  
{  
    "Effect": "Allow",  
    "Action": "pi>ListAvailableResourceDimensions",  
    "Resource": "arn:aws:pi:*.*:metrics/rds/*"  
},  
{  
    "Effect": "Allow",  
    "Action": "pi>ListAvailableResourceMetrics",  
    "Resource": "arn:aws:pi:*.*:metrics/rds/*"  
}  
]  
}
```

## AWS managed policy: AmazonRDSDirectoryServiceAccess

This policy allows Amazon RDS to make calls to the AWS Directory Service.

### Permissions details

This policy includes the following permissions:

- **ds** – Allows principals to describe AWS Directory Service directories and control authorization to AWS Directory Service directories.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Action": [  
                "ds:DescribeDirectories",  
                "ds:AuthorizeApplication",  
                "ds:UnauthorizeApplication",  
                "ds:GetAuthorizedApplicationDetails"  
            ],  
            "Effect": "Allow",  
            "Resource": "*"  
        }  
    ]  
}
```

## AWS managed policy: AmazonRDSServiceRolePolicy

You can't attach the `AmazonRDSServiceRolePolicy` policy to your IAM entities. This policy is attached to a service-linked role that allows Amazon RDS to perform actions on your behalf. For more information, see [Service-linked role permissions for Amazon Aurora \(p. 1724\)](#).

## Amazon RDS updates to AWS managed policies

View details about updates to AWS managed policies for Amazon RDS since this service began tracking these changes. For automatic alerts about changes to this page, subscribe to the RSS feed on the Amazon RDS [Document history](#) page.

Change	Description	Date
<a href="#">Service-linked role permissions for Amazon Aurora (p. 1724)</a> – Update to an existing policy	<p>Amazon RDS added a new Amazon CloudWatch namespace to <code>AmazonRDSPreviewServiceRolePolicy</code> for <code>PutMetricData</code>.</p> <p>This namespace is required for Amazon RDS to publish resource usage metrics.</p> <p>For more information, see <a href="#">Using condition keys to limit access to CloudWatch namespaces</a> in the <i>Amazon CloudWatch User Guide</i>.</p>	June 7, 2022
<a href="#">Service-linked role permissions for Amazon Aurora (p. 1724)</a> – Update to an existing policy	<p>Amazon RDS added a new Amazon CloudWatch namespace to <code>AmazonRDSBetaServiceRolePolicy</code> for <code>PutMetricData</code>.</p> <p>This namespace is required for Amazon RDS to publish resource usage metrics.</p> <p>For more information, see <a href="#">Using condition keys to limit access to CloudWatch namespaces</a> in the <i>Amazon CloudWatch User Guide</i>.</p>	June 7, 2022
<a href="#">Service-linked role permissions for Amazon Aurora (p. 1724)</a> – Update to an existing policy	<p>Amazon RDS added a new Amazon CloudWatch namespace to <code>AWSServiceRoleForRDS</code> for <code>PutMetricData</code>.</p> <p>This namespace is required for Amazon RDS to publish resource usage metrics.</p> <p>For more information, see <a href="#">Using condition keys to limit access to CloudWatch namespaces</a> in the <i>Amazon CloudWatch User Guide</i>.</p>	April 22, 2022
<a href="#">Configuring access policies for Performance Insights (p. 473)</a> – New policy	Amazon RDS added a new service-linked role named <code>AmazonRDSPerformanceInsightsReadOnly</code> to allow Amazon RDS to call	March 10, 2022

Change	Description	Date
	AWS services on behalf of your DB instances.	
<a href="#">Service-linked role permissions for Amazon Aurora (p. 1724)</a> – Update to an existing policy	<p>Amazon RDS added new Amazon CloudWatch namespaces to <code>AWSServiceRoleForRDS</code> for <code>PutMetricData</code>.</p> <p>These namespaces are required for Amazon DocumentDB (with MongoDB compatibility) and Amazon Neptune to publish CloudWatch metrics.</p> <p>For more information, see <a href="#">Using condition keys to limit access to CloudWatch namespaces</a> in the <i>Amazon CloudWatch User Guide</i>.</p>	March 4, 2022
Amazon RDS started tracking changes	Amazon RDS started tracking changes for its AWS managed policies.	October 26, 2021

## 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 \(p. 360\)](#).

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:

- [Granting permissions to publish notifications to an Amazon SNS topic \(p. 577\)](#)
- [Setting up access to an Amazon S3 bucket \(p. 1233\) \(PostgreSQL import\)](#)
- [Setting up access to an Amazon S3 bucket \(p. 1246\) \(PostgreSQL export\)](#)

## IAM database authentication

You can authenticate to your DB cluster using AWS Identity and Access Management (IAM) database authentication. IAM database authentication works with 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 \(p. 1642\)](#).
- 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.

In general, consider using IAM database authentication when your applications create fewer than 200 connections per second, and you don't want to manage usernames and passwords directly in your application code.

The AWS JDBC Driver for MySQL supports IAM database authentication. For more information, see [AWS IAM Database Authentication](#) in the AWS JDBC Driver for MySQL GitHub repository.

### Topics

- [Region and version availability \(p. 1683\)](#)
- [CLI and SDK support \(p. 1684\)](#)
- [Limitations for IAM database authentication \(p. 1684\)](#)
- [Recommendations for IAM database authentication \(p. 1684\)](#)
- [Enabling and disabling IAM database authentication \(p. 1685\)](#)
- [Creating and using an IAM policy for IAM database access \(p. 1686\)](#)
- [Creating a database account using IAM authentication \(p. 1689\)](#)
- [Connecting to your DB cluster using IAM authentication \(p. 1690\)](#)

## Region and version availability

IAM database authentication is available in all Regions.

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 14, 13, and 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 \(p. 1414\)](#).

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 \(p. 57\)](#).

## CLI and SDK support

IAM database authentication is available for the [AWS CLI](#) and for the following language-specific AWS SDKs:

- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java](#)
- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP](#)
- [AWS SDK for Python \(Boto3\)](#)
- [AWS SDK for Ruby](#)

## 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 a user (including the RDS the master user), IAM authentication takes precedence over password authentication, so the user must log in as an IAM user.

## Recommendations for IAM database authentication

We recommend the following when using IAM database authentication:

- Use IAM database authentication as a mechanism for temporary, personal access to databases.
- 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.

## 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 \(p. 127\)](#).
- To modify a DB cluster to enable IAM database authentication, see [Modifying an Amazon Aurora DB cluster \(p. 248\)](#).
- To restore a DB cluster from a snapshot with IAM database authentication enabled, see [Restoring from a DB cluster snapshot \(p. 375\)](#).
- To restore a DB cluster to a point in time with IAM database authentication enabled, see [Restoring a DB cluster to a specified time \(p. 415\)](#).

### 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 [Region and version availability \(p. 1683\)](#).

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 [Region and version availability \(p. 1683\)](#).

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 [Region and version availability \(p. 1683\)](#).

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 for Amazon Aurora \(p. 1653\)](#).

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 \(p. 87\)](#) 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.

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 \(p. 1447\)](#).

- **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 [Identity-based policy examples for Amazon Aurora \(p. 1662\)](#).

## 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 Aurora MySQL \(p. 1689\)](#)
- [Using IAM authentication with Aurora PostgreSQL \(p. 1690\)](#)

## Using IAM authentication with Aurora MySQL

With Aurora 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 Aurora 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 \(p. 1686\)](#).

**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 [Region and version availability \(p. 1683\)](#) and [Enabling and disabling IAM database authentication \(p. 1685\)](#).

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 Aurora PostgreSQL

To use IAM authentication with Aurora 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 \(p. 1686\)](#).

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 \(p. 1045\)](#).

## 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](#)

## Prerequisites

The following are prerequisites for connecting to your DB cluster using IAM authentication:

- [Enabling and disabling IAM database authentication \(p. 1685\)](#)
- [Creating and using an IAM policy for IAM database access \(p. 1686\)](#)
- [Creating a database account using IAM authentication \(p. 1689\)](#)

#### Topics

- [Connecting to your DB cluster using IAM authentication from the command line: AWS CLI and mysql client \(p. 1691\)](#)
- [Connecting to your DB cluster using IAM authentication from the command line: AWS CLI and psql client \(p. 1693\)](#)
- [Connecting to your DB cluster using IAM authentication and the AWS SDK for .NET \(p. 1694\)](#)
- [Connecting to your DB cluster using IAM authentication and the AWS SDK for Go \(p. 1697\)](#)
- [Connecting to your DB cluster using IAM authentication and the AWS SDK for Java \(p. 1700\)](#)
- [Connecting to your DB cluster using IAM authentication and the AWS SDK for Python \(Boto3\) \(p. 1708\)](#)

## 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.

#### Prerequisites

The following are prerequisites for connecting to your DB cluster using IAM authentication:

- [Enabling and disabling IAM database authentication \(p. 1685\)](#)
- [Creating and using an IAM policy for IAM database access \(p. 1686\)](#)
- [Creating a database account using IAM authentication \(p. 1689\)](#)

#### 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 \(p. 1691\)](#)
- [Connecting to a DB cluster \(p. 1692\)](#)

## 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 \(p. 683\)](#).

To download an SSL certificate, see [Using SSL/TLS to encrypt a connection to a DB cluster \(p. 1642\)](#).

- **--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.

### Prerequisites

The following are prerequisites for connecting to your DB cluster using IAM authentication:

- [Enabling and disabling IAM database authentication \(p. 1685\)](#)
- [Creating and using an IAM policy for IAM database access \(p. 1686\)](#)
- [Creating a database account using IAM authentication \(p. 1689\)](#)

### 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 \(p. 1693\)](#)
- [Connecting to an Aurora PostgreSQL cluster \(p. 1694\)](#)

## 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 PGASSWORD 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 \(p. 1029\)](#).

To download an SSL certificate, see [Using SSL/TLS to encrypt a connection to a DB cluster \(p. 1642\)](#).

- `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.

### Prerequisites

The following are prerequisites for connecting to your DB cluster using IAM authentication:

- [Enabling and disabling IAM database authentication \(p. 1685\)](#)
- [Creating and using an IAM policy for IAM database access \(p. 1686\)](#)

- [Creating a database account using IAM authentication \(p. 1689\)](#)

## Examples

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 cluster, 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
- `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 \(p. 1642\)](#).

```
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.RDSSAuthTokenGenerator.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={pwd};SslMode=Required;SslCa=fu
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
  - **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.

- Root 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 \(p. 1642\)](#).

```
using System;
using Npgsql;
using Amazon.RDS.Util;

namespace ConsoleApp1
{
    class Program
    {
        static void Main(string[] args)
        {
            var pwd =
RDSAAuthGenerator.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={pwd};Database=mydb;SSL
Mode=Require;Root 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.

### Prerequisites

The following are prerequisites for connecting to your DB cluster using IAM authentication:

- [Enabling and disabling IAM database authentication \(p. 1685\)](#)
- [Creating and using an IAM policy for IAM database access \(p. 1686\)](#)
- [Creating a database account using IAM authentication \(p. 1689\)](#)

### Examples

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 \(p. 1697\)](#)
- [Connecting using IAM authentication and the AWS SDK for Go V1. \(p. 1699\)](#)

## 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/go-sql-driver/mysql"
)

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.

### Prerequisites

The following are prerequisites for connecting to your DB cluster using IAM authentication:

- [Enabling and disabling IAM database authentication \(p. 1685\)](#)
- [Creating and using an IAM policy for IAM database access \(p. 1686\)](#)
- [Creating a database account using IAM authentication \(p. 1689\)](#)
- [Set up the AWS SDK for Java](#)

## Topics

- [Generating an IAM authentication token \(p. 1701\)](#)
- [Manually constructing an IAM authentication token \(p. 1702\)](#)
- [Connecting to a DB cluster \(p. 1704\)](#)

### 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,
newSigningKey(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 dateTime, String canonicalRequest,
String accessKey, String date, String region) throws Exception {
    String credentialScope = date + "/" + region + "/" + serviceName + "/aws4_request";
    return algorithm + '\n' + dateTime + '\n' + credentialScope + '\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) throws SdkClientException {
    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 Aurora 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 \(p. 1642\)](#).) 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 \(p. 1642\)](#). 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:
 * btusi123.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=AKIAPFXHGVDI5RNFO4AQ%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.

### Prerequisites

The following are prerequisites for connecting to your DB cluster using IAM authentication:

- [Enabling and disabling IAM database authentication \(p. 1685\)](#)
- [Creating and using an IAM policy for IAM database access \(p. 1686\)](#)
- [Creating a database account using IAM authentication \(p. 1689\)](#)

In addition, make sure the imported libraries in the sample code exist on your system.

### Examples

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

For `ssl_ca`, specify an SSL certificate. To download an SSL certificate, see [Using SSL/TLS to encrypt a connection to a DB cluster \(p. 1642\)](#).

This code connects to an Aurora MySQL DB cluster.

Before running this code, install Connector/Python version 8.0 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 \(p. 1710\)](#)
- [I'm not authorized to perform iam:PassRole \(p. 1710\)](#)
- [I want to view my access keys \(p. 1710\)](#)
- [I'm an administrator and want to allow others to access Aurora \(p. 1711\)](#)
- [I want to allow people outside of my AWS account to access my Aurora resources \(p. 1711\)](#)

### 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 \(p. 1657\)](#).
- 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 \(p. 615\)](#).

## 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 \(p. 518\)](#).

## 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 \(p. 461\)](#).

## 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 \(p. 597\)](#).

## 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 \(p. 444\)](#).

## 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 [Working with Amazon RDS event notification \(p. 572\)](#).

## 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 \(p. 619\)](#).

For more information about monitoring Aurora see [Monitoring metrics in an Amazon Aurora cluster \(p. 427\)](#).

# 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 \(p. 368\)](#).

## 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 \(p. 73\)](#).

## 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 \(p. 71\)](#).

# 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 cluster. By default, network access is turned off to a DB cluster. 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 clusters that are associated with that security group.

For more information, see [Controlling access with security groups \(p. 1721\)](#).

## 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 DB 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 cluster in a VPC from the internet \(p. 1735\)](#).

### 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 \(p. 1652\)](#).

# 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. DB instances in your VPC don't need public IP addresses to communicate with Amazon RDS API endpoints to launch, modify, or terminate DB instances and DB clusters. Your DB 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](#).

You don't need an interface VPC endpoint to connect to a DB cluster. For more information, see [Scenarios for accessing a DB cluster in a VPC \(p. 1739\)](#).

## 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 \(p. 1634\)](#). 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

VPC security groups control the access that traffic has in and out of a DB cluster. By default, network access is turned off for a DB cluster. 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 clusters that are associated with that security group. You can specify up to 20 rules in a security group.

### Overview of VPC security groups

Each VPC security group rule makes it possible for a specific source to access a DB cluster 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. However, the outbound traffic rules typically don't apply to DB clusters. Outbound traffic rules apply only if the DB cluster 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 clusters 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 turn on Secure Shell (SSH) access for instances in the VPC, 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 \(p. 1652\)](#).

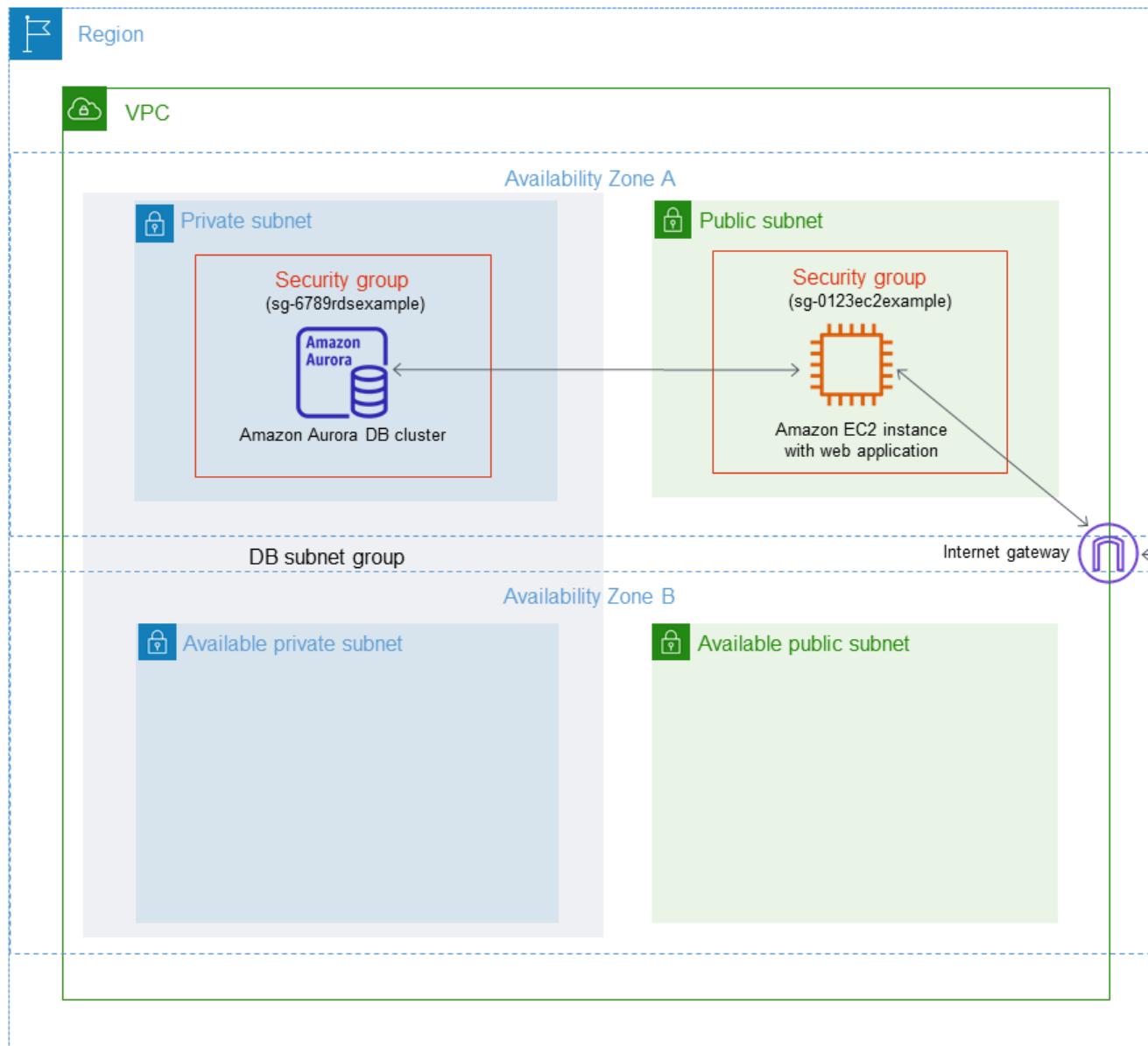
## Security group scenario

A common use of a DB cluster 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 cluster and add the DB cluster to the VPC security group (`sg-6789rdsexample`) that you created in the previous step. When you create the DB cluster, 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 detailed instructions about configuring a VPC for this scenario, see [Tutorial: Create a VPC for use with a DB cluster \(IPv4 only\) \(p. 1744\)](#). For more information about using a VPC, see [Amazon VPC VPCs and Amazon Aurora \(p. 1729\)](#).

## 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 \(p. 90\)](#) and [Security groups](#) in the *Amazon Virtual Private Cloud User Guide*.

## 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 \(p. 248\)](#).

## 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. You can't change the master user name after the DB cluster is created.

### 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 \(p. 248\)](#).

The following table shows the privileges and database roles the master user gets for each of the database engines.

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 For more information about RDS_SUPERUSER, see <a href="#">Understanding PostgreSQL roles and permissions (p. 1019)</a> .

# 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:

```
        "ec2:DescribeCoipPools",
        "ec2:DescribeInternetGateways",
        "ec2:DescribeLocalGatewayRouteTablePermissions",
        "ec2:DescribeLocalGatewayRouteTables",
        "ec2:DescribeLocalGatewayRouteTableVpcAssociations",
        "ec2:DescribeLocalGateways",
        "ec2:DescribeSecurityGroups",
        "ec2:DescribeSubnets",
        "ec2:DescribeVpcAttribute",
        "ec2:DescribeVpcs",
        "ec2:DisassociateAddress",
        "ec2:ModifyNetworkInterfaceAttribute",
        "ec2:ModifyVpcEndpoint",
        "ec2:ReleaseAddress",
        "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:*
```

```
        "Resource": [
            "arn:aws:kinesis:*::stream/aws-rds-das-*"
        ],
    },
{
    "Effect": "Allow",
    "Action": [
        "cloudwatch:PutMetricData"
    ],
    "Resource": "*",
    "Condition": {
        "StringEquals": {
            "cloudwatch:namespace": [
                "AWS/DocDB",
                "AWS/Neptune",
                "AWS/RDS",
                "AWS/Usage"
            ]
        }
    }
},
{
    "Effect": "Allow",
    "Action": [
        "secretsmanager:GetSecretValue",
        "secretsmanager:DescribeSecret",
        "secretsmanager:RestoreSecret",
        "secretsmanager>CreateSecret",
        "secretsmanager>DeleteSecret",
        "secretsmanager:UpdateSecret"
    ],
    "Resource": "arn:aws:secretsmanager:*::secret:rds-sqlserver-ssrs!*"
}
]
```

#### 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 VPC VPCs and Amazon Aurora

Amazon Virtual Private Cloud (Amazon VPC) makes it possible for you to launch AWS resources, such as Aurora DB clusters, into a virtual private cloud (VPC).

When you use a 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 cluster in a VPC.

Accounts have a default VPC. All new DB clusters are created in the default VPC unless you specify otherwise.

## Topics

- [Working with a DB cluster in a VPC \(p. 1729\)](#)
- [Scenarios for accessing a DB cluster in a VPC \(p. 1739\)](#)
- [Tutorial: Create a VPC for use with a DB cluster \(IPv4 only\) \(p. 1744\)](#)
- [Tutorial: Create a VPC for use with a DB cluster \(dual-stack mode\) \(p. 1749\)](#)

Following, you can find a discussion about 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](#).

## Working with a DB cluster in a VPC

Your DB cluster 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 makes it possible for you to launch AWS resources, such as an Amazon Aurora DB cluster 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 that 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 clusters in a VPC , see [Scenarios for accessing a DB cluster in a VPC \(p. 1739\)](#).

## Topics

- [Working with a DB cluster in a VPC \(p. 1729\)](#)
- [Working with DB subnet groups \(p. 1730\)](#)
- [Amazon Aurora IP addressing \(p. 1731\)](#)
- [Hiding a DB cluster in a VPC from the internet \(p. 1735\)](#)
- [Creating a DB cluster in a VPC \(p. 1736\)](#)

In the following tutorials, you can learn to create a VPC that you can use for a common Amazon Aurora scenario:

- [Tutorial: Create a VPC for use with a DB cluster \(IPv4 only\) \(p. 1744\)](#)
- [Tutorial: Create a VPC for use with a DB cluster \(dual-stack mode\) \(p. 1749\)](#)

## Working with a DB cluster in a VPC

Here are some tips on working with a DB cluster 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 cluster. A *subnet* is a segment of a VPC's IP address range that you can specify and that you can use to group DB clusters based on your security and operational needs.
- If you want your DB cluster in the VPC to be publicly accessible, make sure to turn on the VPC attributes *DNS hostnames* and *DNS resolution*.
- Your VPC must have a DB subnet group that you create. 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 the primary DB instance in your DB cluster. The primary DB instance uses the Availability Zone that contains the subnet.
- Your VPC must have a VPC security group that allows access to the DB cluster.

For more information, see [Scenarios for accessing a DB cluster in a VPC \(p. 1739\)](#).

- 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. For example, a range such as 10.0.0.0/24 and 10.0.1.0/24 is typically large enough.
- 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 cluster in a dedicated VPC where the instance tenancy attribute is set to dedicated, the DB instance class of your DB cluster must be one of the approved Amazon EC2 dedicated instance types. For example, the r5.large EC2 dedicated instance corresponds to the db.r5.large 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 a DB cluster, it doesn't guarantee that the DB cluster 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 clusters. By using a DB subnet group, you can specify a particular VPC when creating DB clusters using the AWS CLI or RDS API. If you use the console, you can choose the VPC and subnet groups 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 cluster in a VPC, you choose a DB subnet group for it. From the DB subnet group, Amazon Aurora chooses a subnet and an IP address within that subnet to associate with the primary DB instance in your DB cluster. The DB uses the Availability Zone that contains the subnet.

The subnets in a DB subnet group are either public or private. The subnets are public or private, depending on the configuration that you set for their network access control lists (network ACLs) and routing tables. For a DB cluster to be publicly accessible, all of the subnets in its DB subnet group must be public. If a subnet that's associated with a publicly accessible DB cluster changes from public to private, it can affect DB cluster availability.

To create a DB subnet group that supports dual-stack mode, make sure that each subnet that you add to the DB subnet group has an Internet Protocol version 6 (IPv6) CIDR block associated with it. For more information, see [Amazon Aurora IP addressing \(p. 1731\)](#) and [Migrating to IPv6](#) in the *Amazon VPC User Guide*.

When Amazon Aurora creates a DB cluster in a VPC, it assigns a network interface to your DB cluster by using an IP address from your DB subnet group. However, we strongly recommend that you use the Domain Name System (DNS) name to connect to your DB cluster. We recommend this because the underlying IP address changes during failover.

**Note**

For each DB cluster 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.

## Amazon Aurora IP addressing

IP addresses enable resources in your VPC to communicate with each other, and with resources over the internet. Amazon Aurora support both the Internet Protocol version 4 (IPv4) and IPv6 addressing protocols. By default, Amazon Aurora and Amazon VPC use the IPv4 addressing protocol. You can't turn off this behavior. When you create a VPC, make sure to specify an IPv4 CIDR block (a range of private IPv4 addresses). You can optionally assign an IPv6 CIDR block to your VPC and subnets, and assign IPv6 addresses from that block to DB clusters in your subnet.

Support for the IPv6 protocol expands the number of supported IP addresses. By using the IPv6 protocol, you ensure that you have sufficient available addresses for the future growth of the internet. New and existing RDS resources can use IPv4 and IPv6 addresses within your Amazon VPC. Configuring, securing, and translating network traffic between the two protocols used in different parts of an application can cause operational overhead. You can standardize on the IPv6 protocol for Amazon RDS resources to simplify your network configuration.

### Topics

- [IPv4 addresses \(p. 1731\)](#)
- [IPv6 addresses \(p. 1731\)](#)
- [Dual-stack mode \(p. 1732\)](#)

### IPv4 addresses

When you create a VPC, you must specify a range of IPv4 addresses for the VPC in the form of a CIDR block, such as 10.0.0.0/16. A *DB subnet group* defines the range of IP addresses in this CIDR block that a DB cluster can use. These IP addresses can be private or public.

A private IPv4 address is an IP address that's not reachable over the internet. You can use private IPv4 addresses for communication between your DB cluster and other resources, such as Amazon EC2 instances, in the same VPC. Each DB cluster has a private IP address for communication in the VPC.

A public IP address is an IPv4 address that's reachable from the internet. You can use public addresses for communication between your DB cluster and resources on the internet, such as a SQL client. You control whether your DB cluster receives a public IP address.

For a tutorial that shows you how to create a VPC with only private IPv4 addresses that you can use for a common Amazon Aurora scenario, see [Tutorial: Create a VPC for use with a DB cluster \(IPv4 only\) \(p. 1744\)](#).

### IPv6 addresses

You can optionally associate an IPv6 CIDR block with your VPC and subnets, and assign IPv6 addresses from that block to the resources in your VPC. Each IPv6 addresses is globally unique.

The IPv6 CIDR block for your VPC is automatically assigned from Amazon's pool of IPv6 addresses. You can't choose the range yourself.

When connecting to an IPv6 address, make sure that the following conditions are met:

- The client is configured so that client to database traffic over IPv6 is allowed.
- RDS security groups used by the DB instance are configured correctly so that client to database traffic over IPv6 is allowed.
- The client operating system stack allows traffic on the IPv6 address, and operating system drivers and libraries are configured to choose the correct default DB instance endpoint (either IPv4 or IPv6).

For more information about IPv6, see [IP Addressing in the Amazon VPC User Guide](#).

## Dual-stack mode

When a DB cluster can communicate over both the IPv4 and IPv6 addressing protocols, it's running in dual-stack mode. So, resources can communicate with the DB cluster over IPv4, IPv6, or both. RDS disables Internet Gateway access for IPv6 endpoints of private dual-stack mode DB instances. RDS does this to ensure that your IPv6 endpoints are private and can only be accessed from within your VPC.

### Topics

- [Dual-stack mode and DB subnet groups \(p. 1732\)](#)
- [Working with dual-stack mode DB instances \(p. 1732\)](#)
- [Modifying IPv4-only DB clusters to use dual-stack mode \(p. 1733\)](#)
- [Availability of dual-stack network DB clusters \(p. 1734\)](#)
- [Limitations for dual-stack network DB clusters \(p. 1735\)](#)

For a tutorial that shows you how to create a VPC with both IPv4 and IPv6 addresses that you can use for a common Amazon Aurora scenario, see [Tutorial: Create a VPC for use with a DB cluster \(dual-stack mode\) \(p. 1749\)](#).

### Dual-stack mode and DB subnet groups

To use dual-stack mode, make sure that each subnet in the DB subnet group that you associate with the DB cluster has an IPv6 CIDR block associated with it. You can create a new DB subnet group or modify an existing DB subnet group to meet this requirement. After a DB cluster is in dual-stack mode, clients can connect to it normally. Make sure that client security firewalls and RDS DB instance security groups are accurately configured to allow traffic over IPv6. To connect, clients use the DB cluster primary instance's endpoint. Client applications can specify which protocol is preferred when connecting to a database. In dual-stack mode, the DB cluster detects the client's preferred network protocol, either IPv4 or IPv6, and uses that protocol for the connection.

If a DB subnet group stops supporting dual-stack mode because of subnet deletion or CIDR disassociation, there's a risk of an incompatible network state for DB instances that are associated with the DB subnet group. Also, you can't use the DB subnet group when you create new dual-stack mode DB cluster.

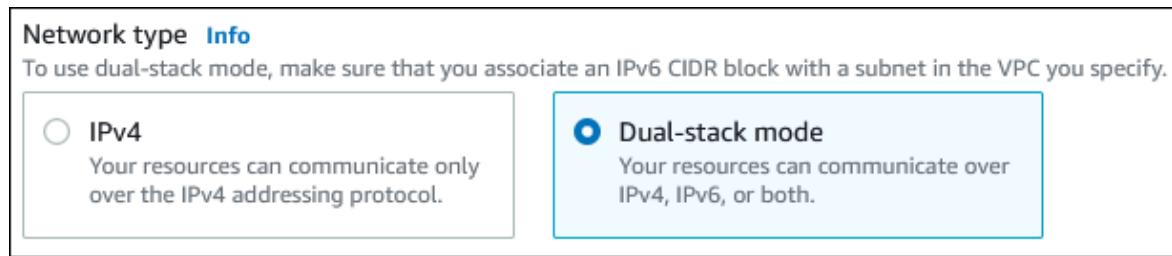
To determine whether a DB subnet group supports dual-stack mode by using the AWS Management Console, view the **Network type** on the details page of the DB subnet group. To determine whether a DB subnet group supports dual-stack mode by using the AWS CLI, call the [describe-db-subnet-groups](#) command and view `SupportedNetworkTypes` in the output.

Read replicas are treated as independent DB instances and can have a network type that's different from the primary DB instance. If you change the network type of a read replica's primary DB instance, the read replica isn't affected. When you are restoring a DB instance, you can restore it to any network type that's supported.

### Working with dual-stack mode DB instances

When you create or modify a DB cluster, you can specify *dual-stack mode* to allow your resources to communicate with your DB cluster over IPv4, IPv6, or both.

When you use the AWS Management Console to create or modify a DB instance, you can specify dual-stack mode in the **Network type** section. The following image shows the **Network type** section in the console.



When you use the AWS CLI to create or modify a DB cluster, set the `--network-type` option to `DUAL` to use dual-stack mode. When you use the RDS API to create or modify a DB cluster, set the `NetworkType` parameter to `DUAL` to use dual-stack mode. When you are modifying the network type of a DB instance, downtime is possible. If dual-stack mode isn't supported by the specified DB engine version or DB subnet group, the `NetworkTypeNotSupported` error is returned.

For more information about creating a DB cluster, see [Creating an Amazon Aurora DB cluster \(p. 127\)](#). For more information about modifying a DB cluster, see [Modifying an Amazon Aurora DB cluster \(p. 248\)](#).

To determine whether a DB cluster is in dual-stack mode by using the console, view the **Network type** on the **Connectivity & security** tab for the DB cluster.

### Modifying IPv4-only DB clusters to use dual-stack mode

You can modify an IPv4-only DB cluster to use dual-stack mode. To do so, change the network type of the DB cluster. The modification might result in downtime.

Before modifying a DB cluster to use dual-stack mode, make sure that its DB subnet group supports dual-stack mode. If the DB subnet group associated with the DB cluster doesn't support dual-stack mode, specify a different DB subnet group that supports it when you modify the DB cluster. If you modify the DB subnet group of a DB cluster before you change the DB cluster to use dual-stack mode, make sure that the DB subnet group is valid for the DB cluster before and after the change.

If you can't connect to the DB cluster after the change, make sure that the client and database security firewalls and route tables are accurately configured to allow cross traffic to the database on the selected network (either IPv4 or IPv6). You might also need to modify operating system parameter, libraries, or drivers to connect using an IPv6 address.

The following limitations apply to modifying a DB cluster to use dual-stack mode:

- Dual-stack mode DB clusters can't be publicly accessible.
- DB clusters can't have an IPv6-only endpoint.

### To modify an IPv4-only DB cluster to use dual-stack mode

1. Modify a DB subnet group to support dual-stack mode, or create a DB subnet group that supports dual-stack mode:

- a. Associate an IPv6 CIDR block with your VPC.

For instructions, see [Associate an IPv6 CIDR block with your VPC](#) in the *Amazon VPC User Guide*.

- b. Attach the IPv6 CIDR block to all of the subnets in your DB subnet group.

For instructions, see [Associate an IPv6 CIDR block with your subnet](#) in the *Amazon VPC User Guide*.

- c. Confirm that the DB subnet group supports dual-stack mode.

If you are using the AWS Management Console, select the DB subnet group, and make sure that the **Supported network types** value is **Dual, IPv4**.

If you are using the AWS CLI, call the [describe-db-subnet-groups](#) command, and make sure that the `SupportedNetworkType` value for the DB instance is `Dual, IPv4`.

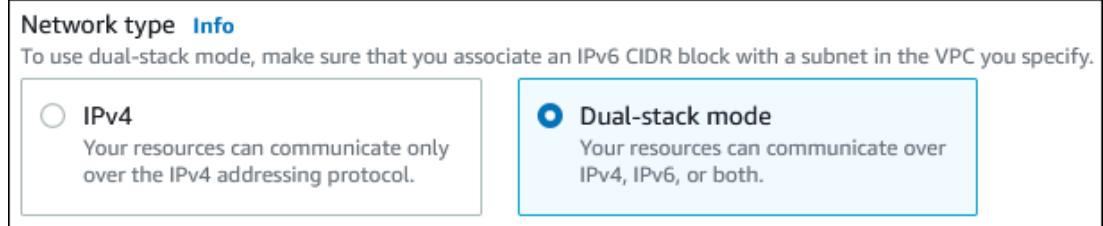
2. Modify the security group associated with the DB cluster to allow IPv6 connections to the database, or create a new security group that allows IPv6 connections.

For instructions, see [Security group rules](#) in the *Amazon VPC User Guide*.

3. Modify the DB cluster to support dual-stack mode. To do so, set the **Network type** to **Dual-stack mode**.

If you are using the console, make sure that the following settings are correct:

- **Network type – Dual-stack mode**



- **DB subnet group** – The DB subnet group that you configured in a previous step
- **Security group** – The security group that you configured in a previous step

If you are using the AWS CLI, make sure that the following settings are correct:

- `--network-type dual`
- `--db-subnet-group-name` – The DB subnet group that you configured in a previous step
- `--vpc-security-group-ids` – The VPC security group that you configured in a previous step

4. Confirm that the DB cluster supports dual-stack mode.

If you are using the console, choose the **Configuration** tab for the DB cluster. On that tab, make sure that the **Network type** value is **Dual-stack mode**.

If you are using the AWS CLI, call the [describe-db-clusters](#) command, and make sure that the `NetworkType` value for the DB cluster is `dual`.

Run the `dig` command on the writer DB instance endpoint to identify the IPv6 address associated with it.

```
dig db-instance-endpoint AAAA
```

Use the writer DB instance endpoint, not the IPv6 address, to connect to the DB cluster.

## Availability of dual-stack network DB clusters

The following DB engine versions support dual-stack network DB clusters:

- Aurora MySQL version 3.02.0 and higher

For more information about Aurora MySQL versions, see the [Release Notes for Aurora MySQL](#).

- Aurora PostgreSQL versions:

- 14.3 and higher 14 versions
- 13.7 and higher 13 versions

For more information about Aurora PostgreSQL versions, see the [Release Notes for Aurora PostgreSQL](#).

### Limitations for dual-stack network DB clusters

The following limitations apply to dual-stack network DB clusters:

- DB clusters can't use the IPv6 protocol exclusively. They can use IPv4 exclusively, or they can use the IPv4 and IPv6 protocol (dual-stack mode).
- Amazon RDS doesn't support native IPv6 subnets.
- DB clusters that use dual-stack mode must be private. They can't be publicly accessible.
- Dual-stack mode doesn't support the db.r3 DB instance classes.
- You can't use RDS Proxy with dual-stack mode DB clusters.

## Hiding a DB cluster 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 cluster with a database that isn't 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. The DB clusters are deployed in the private subnet. In such a deployment, only the web servers have access to the DB clusters. For an illustration of this scenario, see [A DB cluster in a VPC accessed by an EC2 instance in the same VPC \(p. 1739\)](#).

When you launch a DB cluster inside a VPC, the DB cluster 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 cluster also has a public IP address in addition to the private IP address. If the DB cluster is designated as publicly accessible, its DNS endpoint resolves to the private IP address from within the VPC. It resolves to the public IP address from outside of the VPC. Access to the DB cluster is ultimately controlled by the security group it uses. That public access is not permitted if the security group assigned to the DB cluster doesn't include inbound rules that permit it. In addition, for a DB cluster to be publicly accessible, the subnets in its DB subnet group must have an internet gateway. For more information, see [Can't connect to Amazon RDS DB instance \(p. 1761\)](#)

You can modify a DB cluster to turn on or off public accessibility by modifying the **Public access** option. 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

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

Default VPC (vpc-2aed394c) ▾  
Only VPCs with a corresponding DB subnet group are listed.

**After a database is created, you can't change its VPC.**

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

default ▾

**Public access [Info](#)**  
 Yes  
Amazon EC2 instances and devices outside the VPC can connect to your DB cluster. Choose one or more VPC security groups that specify which EC2 instances and devices inside the VPC can connect to the DB cluster.  
 No  
Amazon RDS will not assign a public IP address to the DB cluster. Only Amazon EC2 instances and devices inside the VPC can connect to your DB cluster.

**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 ▾  
default X

▶ Additional configuration

For information about modifying a DB instance to set the **Public access** option, see [Modify a DB instance in a DB cluster \(p. 249\)](#).

## Creating a DB cluster in a VPC

The following procedures help you create a DB cluster in a VPC. To use the default VPC, you can begin with step 2, and use the VPC and DB subnet group have already been created for you. If you want to create an additional VPC, you can create a new VPC.

### Note

If you want your DB cluster 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 \(p. 1737\)](#)
- [Step 2: Create a DB subnet group \(p. 1737\)](#)
- [Step 3: Create a VPC security group \(p. 1739\)](#)
- [Step 4: Create a DB instance in the VPC \(p. 1739\)](#)

## Step 1: Create a VPC

Create a VPC with 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 more information, see [Create a VPC with private and public subnets \(p. 1745\)](#), or see [Create a VPC](#) in the *Amazon VPC User Guide*.

## Step 2: 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 clusters. A DB subnet group allows you to specify a particular VPC when you create DB clusters using the AWS CLI or RDS 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. As a best practice, each DB subnet group should have at least one subnet for every Availability Zone in the AWS Region.

For a DB cluster 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 [Connect to the internet using an internet gateway](#) in the *Amazon VPC User Guide*.

When you create a DB cluster in a VPC, you can choose a DB subnet group. Amazon Aurora chooses a subnet and an IP address within that subnet to associate with your DB cluster. If no DB subnet groups exist, Amazon Aurora creates a default subnet group when you create a DB cluster. Amazon Aurora creates and associates an Elastic Network Interface to your DB cluster with that IP address. The DB cluster uses the Availability Zone that contains the subnet.

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 default VPC or 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 3: Create a VPC security group

Before you create your DB cluster, you can create a VPC security group to associate with your DB cluster. If you don't create a VPC security group, you can use the default security group when you create a DB cluster. For instructions on how to create a security group for your DB cluster, see [Create a VPC security group for a private DB cluster \(p. 1746\)](#), or see [Control traffic to resources using security groups](#) in the [Amazon VPC User Guide](#).

### Step 4: Create a DB instance in the VPC

In this step, you create a DB cluster 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 cluster in the VPC to be publicly accessible, you must enable the VPC attributes *DNS hostnames* and *DNS resolution*. For more information, see [DNS attributes for your VPC](#) in the [Amazon VPC User Guide](#).

For details on how to create a DB cluster, see [Creating an Amazon Aurora DB cluster \(p. 127\)](#).

When prompted in the **Connectivity** section, enter the VPC name, the DB subnet group, and the VPC security group.

**Note**

Updating VPCs isn't currently supported for Aurora DB clusters.

## Scenarios for accessing a DB cluster in a VPC

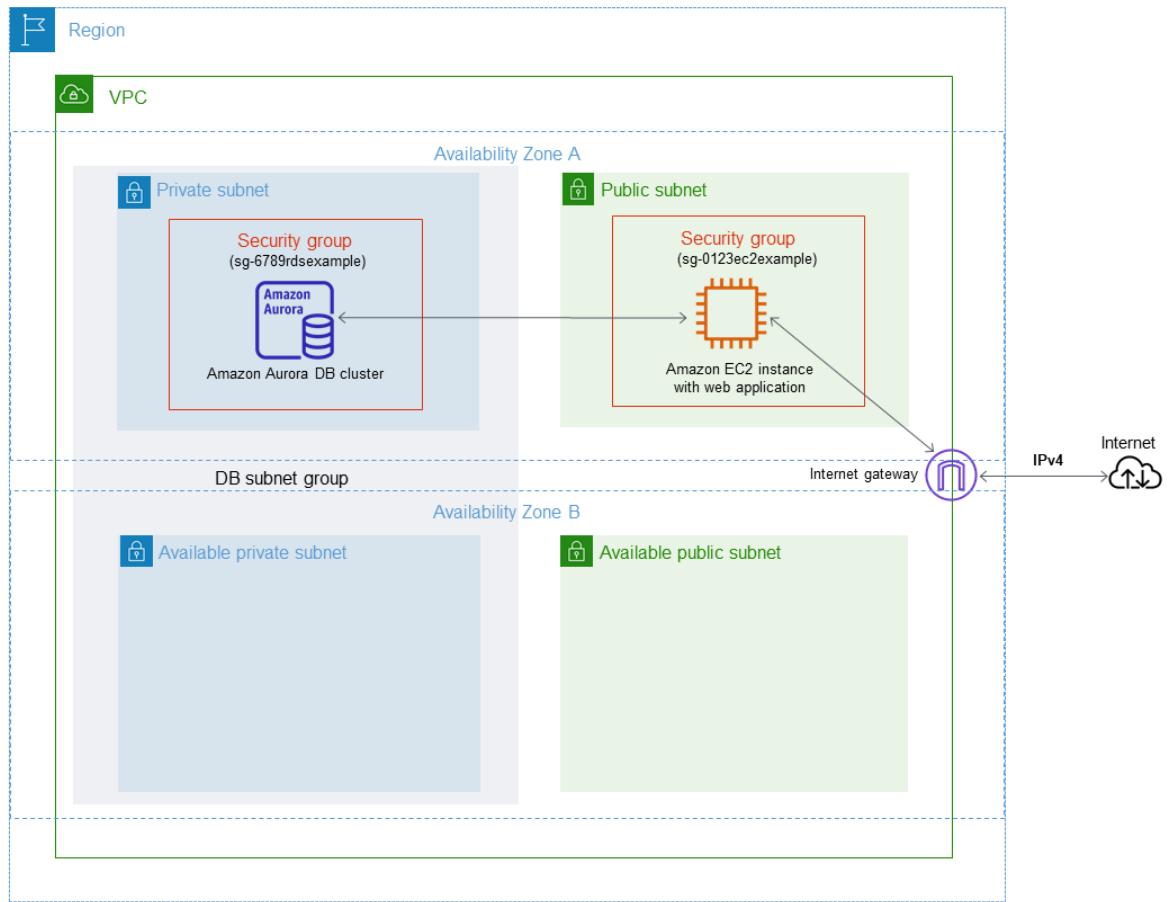
Amazon Aurora supports the following scenarios for accessing a DB cluster in a VPC:

- [An EC2 instance in the same VPC \(p. 1739\)](#)
- [An EC2 instance in a different VPC \(p. 1741\)](#)
- [A client application through the internet \(p. 1742\)](#)
- [A private network \(p. 1743\)](#)

### A DB cluster in a VPC accessed by an EC2 instance in the same VPC

A common use of a DB cluster in a VPC is to share data with an application server that is running in an EC2 instance in the same VPC.

The following diagram shows this scenario.



The simplest way to manage access between EC2 instances and DB clusters in the same VPC is to do the following:

- Create a VPC security group for your DB clusters to be in. This security group can be used to restrict access to the DB clusters. For example, you can create a custom rule for this security group. This might allow TCP access using the port that you assigned to the DB cluster when you created it and an IP address you use to access the DB cluster 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 clusters that allow connections from the security group you created for your EC2 instances. These rules might allow any member of the security group to access the DB clusters.

There is an additional public and private subnet in a separate Availability Zone. An RDS DB subnet group requires a subnet in at least two Availability Zones. The additional subnet makes it easy to switch to a Multi-AZ DB instance deployment in the future.

For a tutorial that shows you how to create a VPC with both public and private subnets for this scenario, see [Tutorial: Create a VPC for use with a DB cluster \(IPv4 only\) \(p. 1744\)](#).

**Tip**

You can set up network connectivity between an Amazon EC2 instance and a DB cluster automatically when you create the DB cluster. For more information, see [Configure automatic network connectivity with an EC2 instance \(p. 127\)](#).

**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 preceding scenario, this is the security group that you use for your DB clusters. Choose the **Inbound rules** tab, and then choose **Edit inbound rules**.
4. On the **Edit inbound rules** page, choose **Add rule**.
5. For **Type**, choose the entry that corresponds to the port you used when you created your DB cluster, 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 **Source**.
8. Choose **Save rules**.

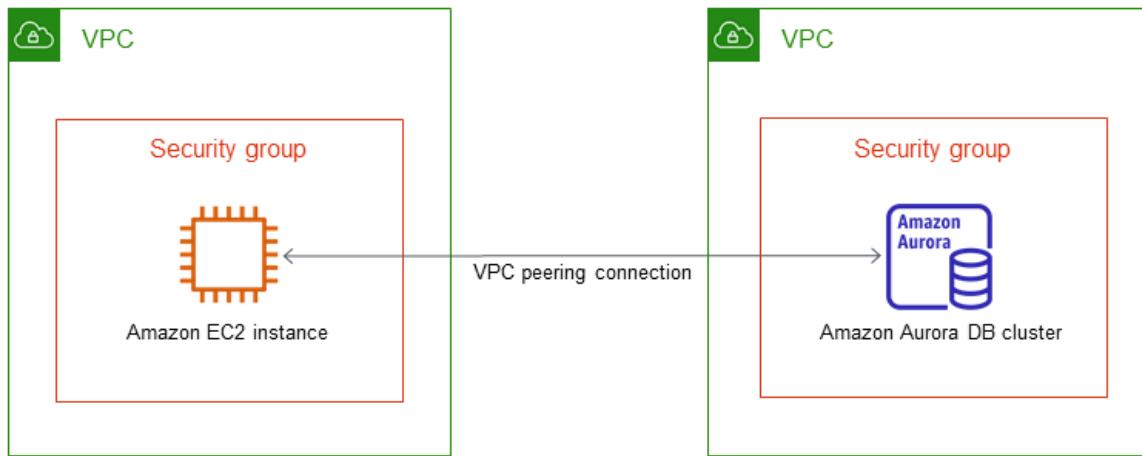
The following screen shows an inbound rule with a security group for its source.

Inbound rules			
Type	Protocol	Port range	Source
MySQL/Aurora	TCP	3306	sg-00bd2328e37926844 (tutorial-securitygroup)

## A DB cluster in a VPC accessed by an EC2 instance in a different VPC

When your DB clusters is in a different VPC from the EC2 instance you are using to access it, you can use VPC peering to access the DB cluster.

The following diagram shows this scenario.

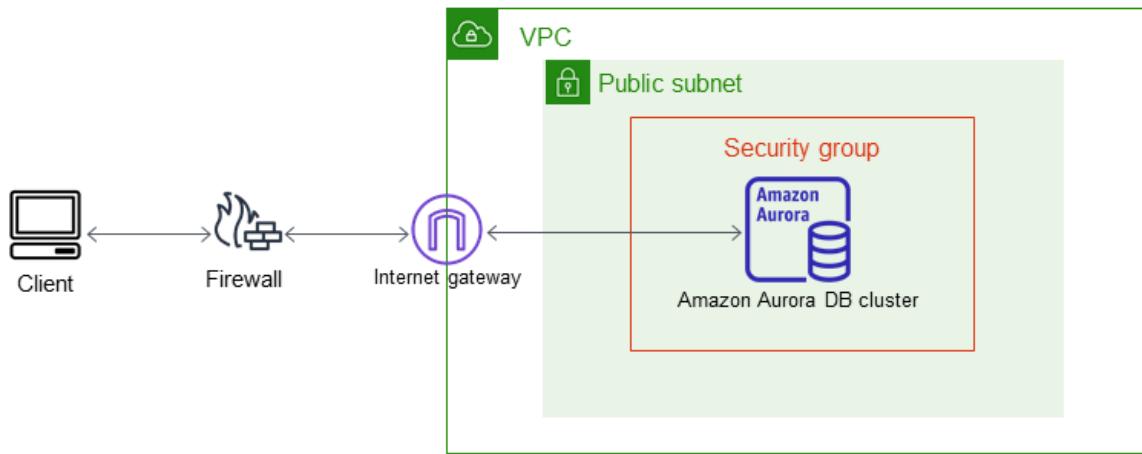


A VPC peering connection is a networking connection between two VPCs that enables you to route traffic between them using private IP addresses. Resources in either VPC can communicate with each other as if they are within the same network. You can create a VPC peering connection between your own VPCs, 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 cluster in a VPC accessed by a client application through the internet

To access a DB clusters 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 cluster that is associated with the VPC and the subnet. Amazon RDS assigns an IP address within the subnet to your DB cluster.
- An internet gateway which connects the VPC to the internet and to other AWS products.

- A security group associated with the DB cluster. The security group's inbound rules allow your client application to access to your DB cluster.

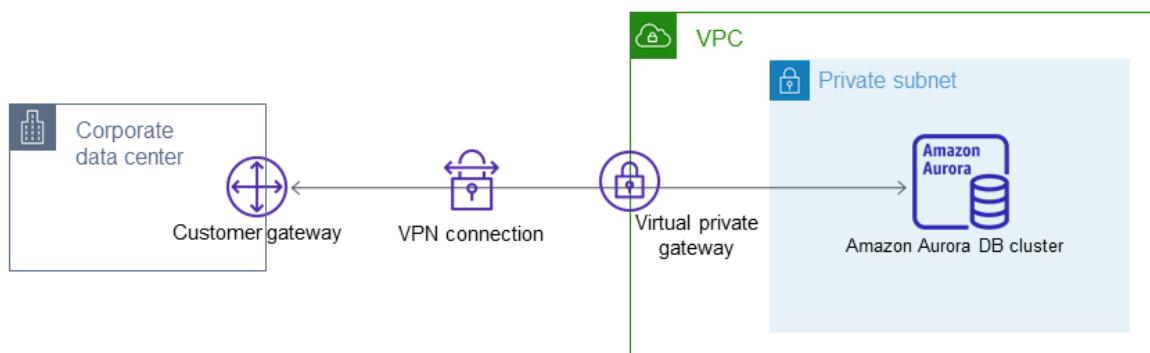
For information about creating a DB clusters in a VPC, see [Creating a DB cluster in a VPC \(p. 1736\)](#).

## A DB cluster in a VPC accessed by a private network

If your DB cluster 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.

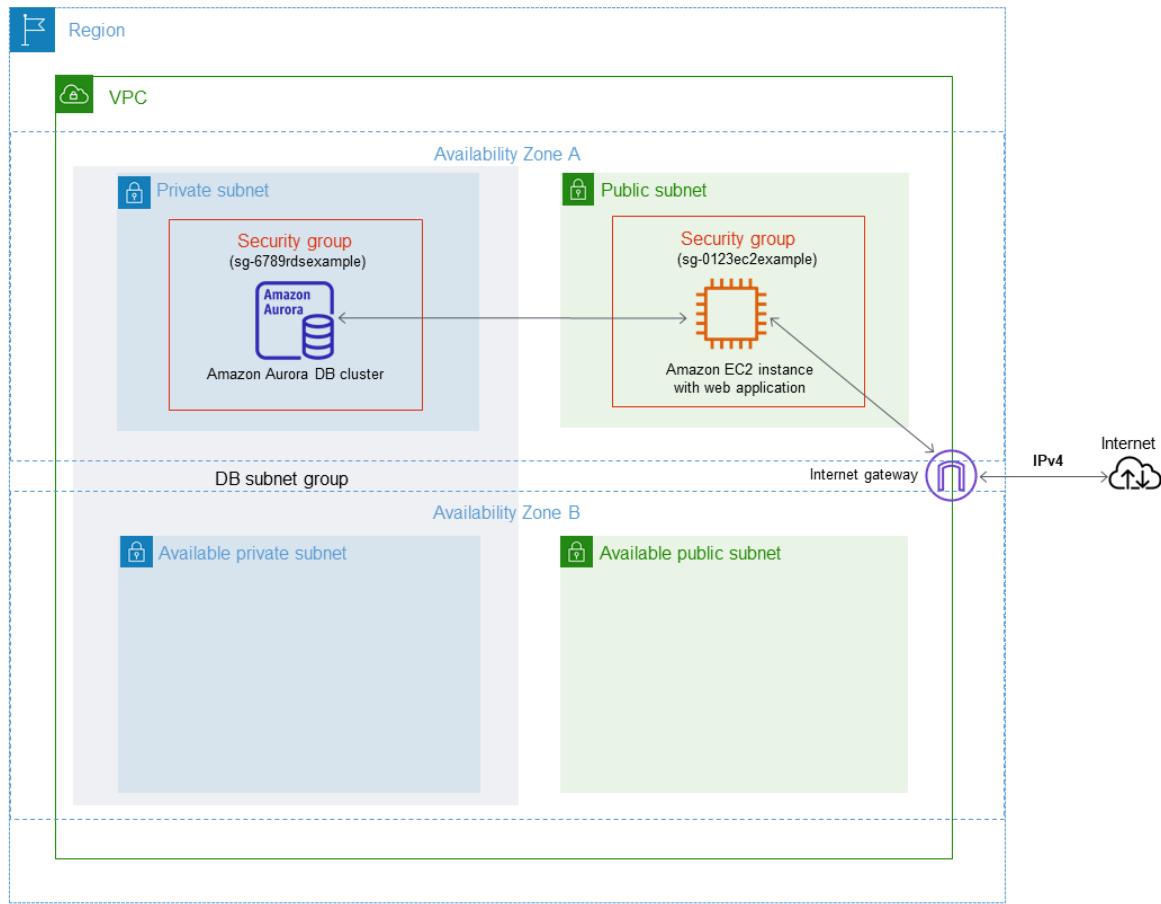


For more information, see [Internetwork traffic privacy \(p. 1652\)](#).

# Tutorial: Create a VPC for use with a DB cluster (IPv4 only)

A common scenario includes a DB cluster in a virtual private cloud (VPC) based on the Amazon VPC service. This VPC 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 cluster in a VPC \(p. 1739\)](#).



Because your DB cluster needs to be available only 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 cluster is hosted in a private subnet. The web server can connect to the DB cluster because it is hosted within the same VPC. But the DB cluster isn't available to the public internet, providing greater security.

This tutorial configures an additional public and private subnet in a separate Availability Zone. These subnets aren't used by the tutorial. An RDS DB subnet group requires a subnet in at least two Availability Zones. The additional subnet makes it easier to configure more than one Aurora DB instance.

This tutorial describes configuring a VPC for Amazon Aurora DB clusters. 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 \(p. 107\)](#). For more information about Amazon VPC, see [Amazon VPC Getting Started Guide](#) and [Amazon VPC User Guide](#).

**Tip**

You can set up network connectivity between an Amazon EC2 instance and a DB cluster automatically when you create the DB cluster. The network configuration is similar to the one described in this tutorial. For more information, see [Configure automatic network connectivity with an EC2 instance \(p. 127\)](#).

## 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. 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 Region to create your VPC in. This example uses the US West (Oregon) Region.
3. In the upper-left corner, choose **VPC dashboard**. To begin creating a VPC, choose **Create VPC**.
4. For **Resources to create** under **VPC settings**, choose **VPC and more**.
5. For the **VPC settings**, set these values:
  - **Name tag auto-generation** – `tutorial`
  - **IPv4 CIDR block** – `10.0.0.0/16`
  - **IPv6 CIDR block** – `No IPv6 CIDR block`
  - **Tenancy** – `Default`
  - **Number of Availability Zones (AZs)** – `2`
  - **Customize AZs** – Keep the default values.
  - **Number of public subnet** – `2`
  - **Number of private subnets** – `2`
  - **Customize subnets CIDR blocks** – Keep the default values.
  - **NAT gateways (\$)** – `None`
  - **VPC endpoints** – `None`
  - **DNS options** – Keep the default values.
6. Choose **Create VPC**.

## Create a VPC security group for a public web server

Next, you create a security group for public access. To connect to public EC2 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 EC2 instances in your VPC using Secure Shell (SSH). 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 for SSH access, you make it possible for all IP addresses to access your public instances using SSH. This approach is acceptable for a short time in a test environment, but it's unsafe for production environments. In production, authorize only a specific IP address or range of addresses to access your instances using SSH.

- b. In the **Inbound rules** section, choose **Add rule**.
  - c. Set the following values for your new inbound rule to allow SSH access to your Amazon EC2 instance. If you do this, you can connect to your Amazon 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. Choose **Create security group** to create the security group.

Note the security group ID because you need it later in this tutorial.

## Create a VPC security group for a private DB cluster

To keep your DB cluster private, create a second security group for private access. To connect to private DB clusters 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 Amazon EC2 instance. If you do this, you can connect from your web server to your DB cluster 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. Choose **Create security group** to create the 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 clusters. A DB subnet group makes it possible for you to specify a particular VPC when creating DB clusters.

### To create a DB subnet group

1. Identify the private subnets for your database in the VPC.
    - a. Open the Amazon VPC console at <https://console.aws.amazon.com/vpc/>.
    - b. Choose **VPC Dashboard**, and then choose **Subnets**.
    - c. Note the subnet IDs of the subnets named **tutorial-subnet-private1-us-west-2a** and **tutorial-subnet-private2-us-west-2b**.

You need the subnet IDs when you create your DB subnet group.
  2. Open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.

Make sure that you connect to the Amazon RDS console, not to the Amazon VPC console.
  3. In the navigation pane, choose **Subnet groups**.
  4. Choose **Create DB subnet group**.
  5. 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)`
  6. In the **Add subnets** section, choose the **Availability Zones** and **Subnets**.

For this tutorial, choose **us-west-2a** and **us-west-2b** for the **Availability Zones**. For **Subnets**, choose the private subnets you identified in the previous step.
  7. 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 in the details pane at the bottom of the window. These details include all of the subnets associated with the group.

#### Note

If you created this VPC to complete [Tutorial: Create a web server and an Amazon Aurora DB cluster \(p. 107\)](#), create the DB cluster by following the instructions in [Create an Amazon Aurora DB cluster \(p. 108\)](#).

## 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 VPC that you created for this tutorial, you might need to delete these resources before you can delete the VPC. For example, these resources might include Amazon EC2 instances or Amazon RDS DB clusters. 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 that you created, such as **tutorial-vpc**.
    - d. Note the **VPC ID** of the VPC that you created. You need the VPC ID in later 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. For **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. For **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 that you created. Use the VPC ID to identify the correct NAT gateway.
    - d. For **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. For **Actions**, choose **Delete VPC**.

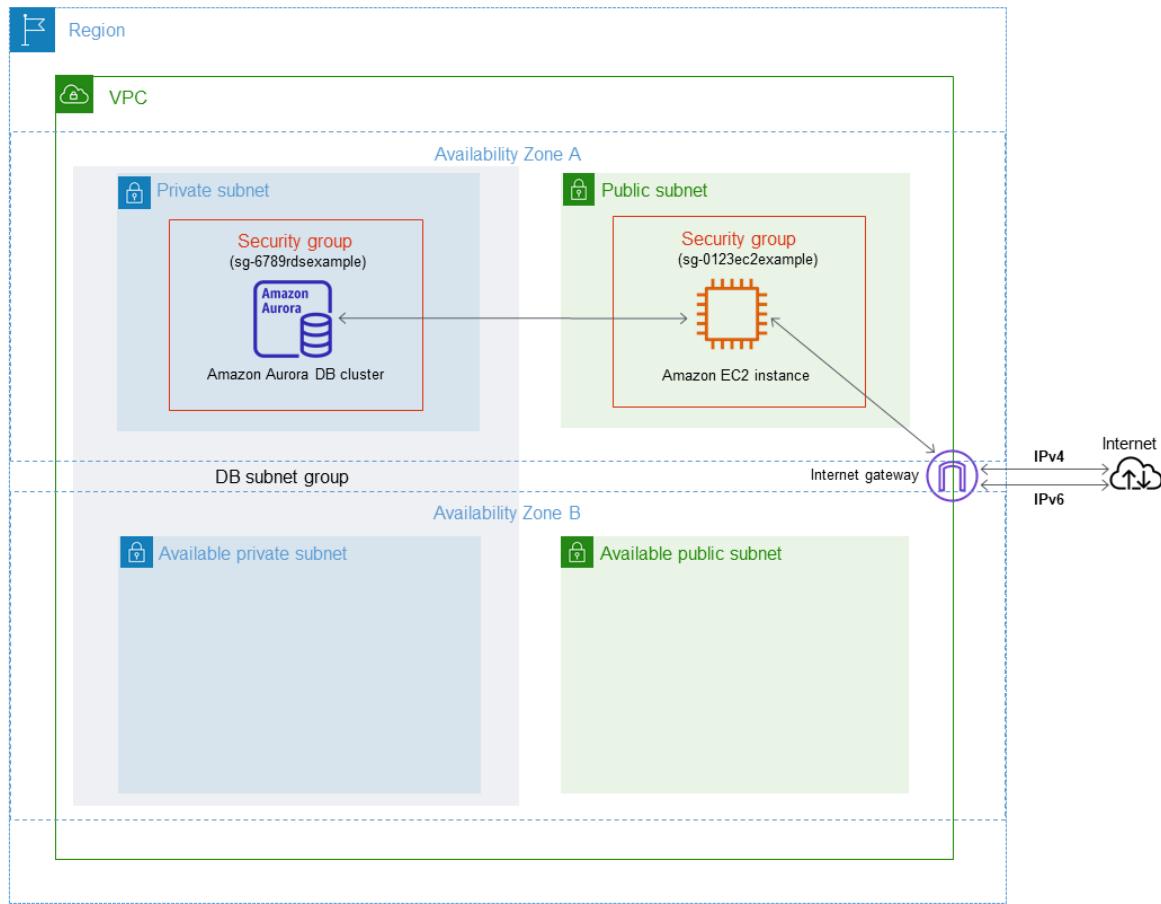
The confirmation page shows other resources that are associated with the VPC that will also be deleted, including the subnets associated with it.

    - 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 **Amazon EC2 Dashboard**, and then choose **Elastic IPs**.
    - c. Select the Elastic IP address you want to release.
    - d. For **Actions**, choose **Release Elastic IP addresses**.
    - e. On the confirmation page, choose **Release**.

## Tutorial: Create a VPC for use with a DB cluster (dual-stack mode)

A common scenario includes a DB cluster in a virtual private cloud (VPC) based on the Amazon VPC service. This VPC shares data with a public Amazon EC2 instance that is running in the same VPC. In this tutorial, you create the VPC for this scenario that works with a database running in dual-stack mode.

The following diagram shows this scenario.



For information about other scenarios, see [Scenarios for accessing a DB cluster in a VPC \(p. 1739\)](#).

Because your DB cluster needs to be available only to your Amazon EC2 instance, and not to the public internet, you create a VPC with both public and private subnets. The Amazon EC2 instance is hosted in the public subnet, so that it can reach the public internet. The DB cluster is hosted in a private subnet. The Amazon EC2 instance can connect to the DB cluster because it's hosted within the same VPC. However, the DB cluster is not available to the public internet, providing greater security.

This tutorial configures an additional public and private subnet in a separate Availability Zone. These subnets aren't used by the tutorial. An RDS DB subnet group requires a subnet in at least two Availability Zones. The additional subnet makes it easy to configure more than one Aurora DB instance.

To create a DB cluster that uses dual-stack mode, specify **Dual-stack mode** for the **Network type** setting. You can also modify a DB cluster with the same setting. For more information about creating a DB cluster, see [Creating an Amazon Aurora DB cluster \(p. 127\)](#). For more information about modifying a DB cluster, see [Modifying an Amazon Aurora DB cluster \(p. 248\)](#).

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](#).

## 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. Open the Amazon VPC console at <https://console.aws.amazon.com/vpc/>.
2. In the upper-right corner of the AWS Management Console, choose the Region to create your VPC in. This example uses the US East (Ohio) Region.
3. In the upper-left corner, choose **VPC dashboard**. To begin creating a VPC, choose **Create VPC**.
4. For **Resources to create** under **VPC settings**, choose **VPC and more**.
5. For the remaining **VPC settings**, set these values:
  - **Name tag auto-generation** – **tutorial-dual-stack**
  - **IPv4 CIDR block** – **10.0.0.0/16**
  - **IPv6 CIDR block** – **Amazon-provided IPv6 CIDR block**
  - **Tenancy** – **Default**
  - **Number of Availability Zones (AZs)** – **2**
  - **Customize AZs** – Keep the default values.
  - **Number of public subnet** – **2**
  - **Number of private subnets** – **2**
  - **Customize subnets CIDR blocks** – Keep the default values.
  - **NAT gateways (\$)** – **None**
  - **Egress only internet gateway** – **No**
  - **VPC endpoints** – **None**
  - **DNS options** – Keep the default values.

#### Note

Amazon RDS requires at least two subnets in two different Availability Zones to support Multi-AZ DB instance deployments. This tutorial creates a Single-AZ deployment, but the requirement makes it easy to convert to a Multi-AZ DB instance deployment in the future.

6. Choose **Create VPC**.

## Create a VPC security group for a public Amazon EC2 instance

Next, you create a security group for public access. To connect to public EC2 instances in your VPC, 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-dual-stack-securitygroup**
  - **Description:** **Tutorial Dual-Stack Security Group**

- **VPC:** Choose the VPC that you created earlier, for example: `vpc-identifier (tutorial-dual-stack-vpc)`
- 4. Add inbound rules to the security group.

- a. Determine the IP address to use to connect to EC2 instances in your VPC using Secure Shell (SSH).

To determine your public IP address, in a different browser window or tab use the service at <https://checkip.amazonaws.com>. An example of an Internet Protocol version 4 (IPv4) address is 203.0.113.25/32. An example of an Internet Protocol version 6 (IPv6) address is 2001:DB8::/32.

If you are connecting through an internet service provider (ISP) or from behind your firewall without a static IP address, find out the range of IP addresses used by client computers.

**Warning**

If you use 0.0.0.0/0 for IPv4 or ::0 for IPv6, you make it possible for all IP addresses to access your public instances using SSH. This approach is acceptable for a short time in a test environment, but it's unsafe for production environments. In production, 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 Amazon EC2 instance. If you do this, you can connect to your EC2 instance to install SQL clients and other applications. Specify an IP address to allow to access your EC2 instance:
  - **Type: SSH**
  - **Source:** The IP address or range from step a. An example of an IPv4 IP address is **203.0.113.25/32**. An example of an IPv6 IP address is **2001:DB8::/32**.

5. Choose **Create security group** to create the security group.

Note the security group ID because you need it later in this tutorial.

## Create a VPC security group for a private DB cluster

To keep your DB cluster private, create a second security group for private access. To connect to private DB clusters in your VPC, add inbound rules to your VPC security group that allow traffic from your Amazon EC2 instance 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-dual-stack-db-securitygroup`
  - **Description:** `Tutorial Dual-Stack DB Instance Security Group`
  - **VPC:** Choose the VPC that you created earlier, for example: `vpc-identifier (tutorial-dual-stack-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 Amazon EC2 instance. If you do this, you can connect from your EC2 instance to your DB cluster to store and retrieve data from your EC2 instance to your database.
    - **Type: MySQL**
    - **Source:** The IP address or range from step a. An example of an IPv4 IP address is **203.0.113.25/32**. An example of an IPv6 IP address is **2001:DB8::/32**.
    - **Port:** **3306**

- **Type:** MySQL/Aurora
  - **Source:** The identifier of the **tutorial-dual-stack-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 clusters. By using a DB subnet group, you can specify a particular VPC when creating DB clusters. To create a DB subnet group that is DUAL compatible, all subnets must be DUAL compatible. To be DUAL compatible, a subnet must have an IPv6 CIDR associated with it.

### To create a DB subnet group

1. Identify the private subnets for your database in the VPC.
  - a. Open the Amazon VPC console at <https://console.aws.amazon.com/vpc/>.
  - b. Choose **VPC Dashboard**, and then choose **Subnets**.
  - c. Note the subnet IDs of the subnets named **tutorial-dual-stack-subnet-private1-us-west-2a** and **tutorial-dual-stack-subnet-private2-us-west-2b**.

You will need the subnet IDs when you create your DB subnet group.

2. Open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.

Make sure that you connect to the Amazon RDS console, not to the Amazon VPC console.

3. In the navigation pane, choose **Subnet groups**.
4. Choose **Create DB subnet group**.
5. On the **Create DB subnet group** page, set these values in **Subnet group details**:

- **Name:** **tutorial-dual-stack-db-subnet-group**
- **Description:** **Tutorial Dual-Stack DB Subnet Group**
- **VPC:** **tutorial-dual-stack-vpc (vpc-*identifier*)**

6. In the **Add subnets** section, choose values for the **Availability Zones** and **Subnets** options.

For this tutorial, choose **us-east-2a** and **us-east-2b** for the **Availability Zones**. For **Subnets**, choose the private subnets you identified in the previous step.

7. 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 its details. These include the supported addressing protocols and all of the subnets associated with the group and the network type supported by the DB subnet group.

## Create an Amazon EC2 instance in dual-stack mode

To create an Amazon EC2 instance, follow the instructions in [Launch an instance using the new launch instance wizard](#) in the *Amazon EC2 User Guide for Linux Instances*.

On the **Configure Instance Details** page, 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 **vpc-*identifier* | tutorial-dual-stack-vpc** created in [Create a VPC with private and public subnets \(p. 1750\)](#).

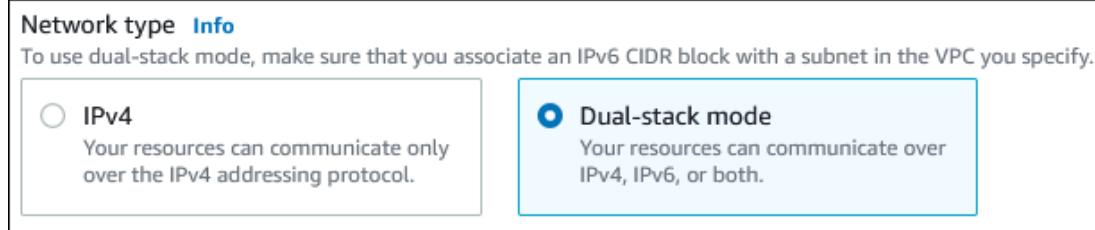
- **Subnet** – Choose an existing public subnet, such as **subnet-*identifier* | tutorial-dual-stack-subnet-public1-us-east-2a | us-east-2a** created in [Create a VPC security group for a public Amazon EC2 instance \(p. 1750\)](#).
- **Auto-assign Public IP** – Choose **Enable**.
- **Auto-assign IPv6 IP** – Choose **Enable**.
- **Firewall (security groups)** – Choose **Select an existing security group**.
- **Common security groups** – Choose choose an existing security group, such as the **tutorial-securitygroup** created in [Create a VPC security group for a public Amazon EC2 instance \(p. 1750\)](#). Make sure that the security group that you choose includes inbound rules for Secure Shell (SSH) and HTTP access.

## Create a DB cluster in dual-stack mode

In this step, you create a DB cluster that runs in dual-stack mode.

### To create 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 upper-right corner of the console, choose the AWS Region where you want to create the DB cluster. This example uses the US East (Ohio) Region.
3. In the navigation pane, choose **Databases**.
4. Choose **Create database**.
5. On the **Create database** page, make sure that the **Standard create** option is chosen, and then choose the Aurora MySQL DB engine type.
6. In the **Connectivity** section, set these values:
  - **Network type** – Choose **Dual-stack mode**.



- **Virtual private cloud (VPC)** – Choose an existing VPC with both public and private subnets, such as **tutorial-dual-stack-vpc** (**vpc-*identifier***) created in [Create a VPC with private and public subnets \(p. 1750\)](#).

The VPC must have subnets in different Availability Zones.

- **DB subnet group** – Choose a DB subnet group for the VPC, such as **tutorial-dual-stack-db-subnet-group** created in [Create a DB subnet group \(p. 1752\)](#).
- **Public access** – Choose **No**.
- **VPC security group (firewall)** – Select **Choose existing**.
- **Existing VPC security groups** – Choose an existing VPC security group that is configured for private access, such as **tutorial-dual-stack-db-securitygroup** created in [Create a VPC security group for a private DB cluster \(p. 1751\)](#).

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

- **Availability Zone** – Choose **us-west-2a**.

To avoid cross-AZ traffic, make sure the DB instance and the EC2 instance are in the same Availability Zone.

7. For the remaining sections, specify your DB cluster settings. For information about each setting, see [Settings for Aurora DB clusters \(p. 137\)](#).

## Connect to your Amazon EC2 instance and DB cluster

After you create your Amazon EC2 instance and DB cluster in dual-stack mode, you can connect to each one using the IPv6 protocol. To connect to an Amazon EC2 instance using the IPv6 protocol, follow the instructions in [Connect to your Linux instance](#) in the *Amazon EC2 User Guide for Linux Instances*.

To connect to your DB cluster, follow the instructions in [Connecting to an Amazon Aurora DB cluster \(p. 207\)](#).

## Deleting the VPC

After you create the VPC and other resources for this tutorial, you can delete them if they are no longer needed.

If you added resources in the VPC that you created for this tutorial, such as Amazon EC2 instances or DB clusters, 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 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-dual-stack-vpc**.
  - d. Note the **VPC ID** value of the VPC that you created. You need this 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-dual-stack-db-securitygroup**.
  - d. For **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-dual-stack-securitygroup**.
  - f. For **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 that you created. Use the VPC ID to identify the correct NAT gateway.
  - d. For **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 that you want to delete, such as **tutorial-dual-stack-vpc**.
  - d. For **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 that you want to release.
  - d. For **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 \(p. 1756\)](#)
- [Naming constraints in Amazon Aurora \(p. 1759\)](#)
- [Amazon Aurora size limits \(p. 1759\)](#)

## 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	Adjust	Description
Authorizations per DB security group	Each supported Region: 20	No	Number of security group authorizations per DB security group
Custom engine versions	Each supported Region: 40	Yes	The maximum number of custom engine versions allowed in this account in the current Region
DB cluster parameter groups	Each supported Region: 50	No	The maximum number of DB cluster parameter groups
DB clusters	Each supported Region: 40	Yes	The maximum number of Aurora clusters allowed in this account in the current Region
DB instances	Each supported Region: 40	Yes	The maximum number of DB instances allowed in this account in the current Region
DB subnet groups	Each supported Region: 50	Yes	The maximum number of DB subnet groups
Data API HTTP request body size	Each supported Region: 4 Megabytes	No	The maximum size allowed for the HTTP request body.

Name	Default	Adjust	Description
Data API maximum concurrent cluster-secret pairs	Each supported Region: 30	No	The maximum number of unique pairs of Aurora Serverless DB clusters and secrets in concurrent Data API requests for the current account and AWS Region.
Data API maximum concurrent requests	Each supported Region: 500	No	The maximum number of Data API requests to an Aurora Serverless DB cluster that use the same secret and can be processed at the same time. Additional requests are queued and processed as in-process requests complete.
Data API maximum result set size	Each supported Region: 1 Megabytes	No	The maximum size of the database result set that can be returned by the Data API.
Data API maximum size of JSON response string	Each supported Region: 10 Megabytes	No	The maximum size of the simplified JSON response string returned by the RDS Data API.
Data API requests per second	Each supported Region: 1,000 per second	No	The maximum number of requests to the Data API per second allowed in this account in the current AWS Region.
Event subscriptions	Each supported Region: 20	Yes	The maximum number of event subscriptions
IAM roles per DB cluster	Each supported Region: 5	Yes	The maximum number of IAM roles associated with a DB cluster
IAM roles per DB instance	Each supported Region: 5	Yes	The maximum number of IAM roles associated with a DB instance
Manual DB cluster snapshots	Each supported Region: 100	Yes	The maximum number of manual DB cluster snapshots
Manual DB instance snapshots	Each supported Region: 100	Yes	The maximum number of manual DB instance snapshots
Option groups	Each supported Region: 20	Yes	The maximum number of option groups

Name	Default	Adjust	Description
Parameter groups	Each supported Region: 50	Yes	The maximum number of parameter groups
Proxies	Each supported Region: 20	Yes	The maximum number of proxies allowed in this account in the current AWS Region
Read replicas per master	Each supported Region: 5	Yes	The maximum number of read replicas per master
Reserved DB instances	Each supported Region: 40	Yes	The maximum number of reserved DB instances allowed in this account in the current AWS Region
Rules per security group	Each supported Region: 20	No	The maximum number of rules per DB security group
Security groups	Each supported Region: 25	Yes	The maximum number of DB security groups
Security groups (VPC)	Each supported Region: 5	No	The maximum number of DB security groups per Amazon VPC
Subnets per DB subnet group	Each supported Region: 20	No	The maximum number of subnets per DB subnet group
Tags per resource	Each supported Region: 50	No	The maximum number of tags per Amazon RDS resource
Total storage for all DB instances	Each supported Region: 100,000 Gigabytes	Yes	The maximum total storage (in GB) for all DB instances added together

### Note

The limits on total storage and number of read replicas don't apply to Aurora.

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"><li>Must contain 1–63 alphanumeric characters or hyphens.</li><li>First character must be a letter.</li><li>Can't end with a hyphen or contain two consecutive hyphens.</li><li>Must be unique for all DB instances per AWS account, per AWS Region.</li></ul>
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.
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"><li>Must contain 1–255 alphanumeric characters.</li><li>First character must be a letter.</li><li>Hyphens are allowed, but the name cannot end with a hyphen or contain two consecutive hyphens.</li></ul>
DB subnet group name	These names have these constraints: <ul style="list-style-type: none"><li>Must contain 1–255 characters.</li><li>Alphanumeric characters, spaces, hyphens, underscores, and periods are allowed.</li></ul>

## 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 14 and 13 versions; Aurora PostgreSQL 12.4, 11.7, 10.12, and 9.6.17 and higher minor versions of each of these versions.

For lower engine versions, the maximum size of an Aurora cluster volume is 64 TiB. For more information, see [How Aurora storage automatically resizes \(p. 67\)](#).

#### 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.

To monitor the remaining storage space, you can use the `AuroraVolumeBytesLeftTotal` metric. For more information, see [Cluster-level metrics for Amazon Aurora \(p. 525\)](#).

# Troubleshooting for Amazon Aurora

Use the following sections to help troubleshoot problems you have with DB instances in Amazon RDS and Amazon Aurora.

## Topics

- [Can't connect to Amazon RDS DB instance \(p. 1761\)](#)
- [Amazon RDS security issues \(p. 1763\)](#)
- [Resetting the DB instance owner password \(p. 1763\)](#)
- [Amazon RDS DB instance outage or reboot \(p. 1764\)](#)
- [Amazon RDS DB parameter changes not taking effect \(p. 1764\)](#)
- [Freeable memory issues in Amazon Aurora \(p. 1764\)](#)
- [Amazon Aurora MySQL out of memory issues \(p. 1765\)](#)
- [Amazon Aurora MySQL replication issues \(p. 1766\)](#)

For information about debugging problems using the Amazon RDS API, see [Troubleshooting applications on Aurora \(p. 1770\)](#).

## 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 \(p. 90\)](#).

### 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 cluster in a VPC from the internet \(p. 1735\)](#).

- **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.

If you're connecting to your instance using its IPv6 address, verify that there is a route for all IPv6 traffic (::/0) that points to the internet gateway. Otherwise, do the following:

- a. Choose the ID of the route table (rtb-xxxxxxx) 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.  
For IPv6, choose **Add route**, use ::/0 as the destination and the internet gateway as the target.
- c. Choose **Save routes**.

Also, if you are trying to connect to IPv6 endpoint, make sure that client IPv6 address range is authorized to connect to the DB instance.

For more information, see [Working with a DB cluster in a VPC \(p. 1729\)](#).

## 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 \(p. 87\)](#).

## 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 \(p. 1723\)](#).

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 \(p. 249\)](#).

## 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 \(p. 231\)](#).

## Freeable memory issues in Amazon Aurora

*Freeable memory* is the total random access memory (RAM) on a DB instance that can be made available to the database engine. It's the sum of the free operating-system (OS) memory and the available buffer and page cache memory. The database engine uses most of the memory on the host, but OS processes also use some RAM. Memory currently allocated to the database engine or used by OS processes isn't included in freeable memory. When the database engine is running out of memory, the DB instance can

use the temporary space that is normally used for buffering and caching. As previously mentioned, this temporary space is included in freeable memory.

You use the `FreeableMemory` metric in Amazon CloudWatch to monitor the freeable memory. For more information, see [Overview of monitoring metrics in Amazon Aurora \(p. 428\)](#).

If your DB instance consistently runs low on freeable memory or uses swap space, you might need to scale up to a larger DB instance class. For more information, see [Aurora DB instance classes \(p. 56\)](#).

You can also change the memory settings. For example, on Aurora MySQL, you might adjust the size of the `innodb_buffer_pool_size` parameter. This parameter is set by default to 75 percent of physical memory. For more MySQL troubleshooting tips, see [How can I troubleshoot low freeable memory in an Amazon RDS for MySQL database?](#)

For Aurora Serverless v2, `FreeableMemory` represents the amount of unused memory that's available when the Aurora Serverless v2 DB instance is scaled to its maximum capacity. You might have the instance scaled down to relatively low capacity, but it still reports a high value for `FreeableMemory`, because the instance can scale up. That memory isn't available right now, but you can get it if you need it.

For every Aurora capacity unit (ACU) that the current capacity is below the maximum capacity, `FreeableMemory` increases by approximately 2 GiB. Thus, this metric doesn't approach zero until the DB instance is scaled up as high as it can.

If this metric approaches a value of 0, the DB instance has scaled up as much as it can and is nearing the limit of its available memory. Consider increasing the maximum ACU setting for the cluster. If this metric approaches a value of 0 on a reader DB instance, consider adding additional reader DB instances to the cluster. That way, the read-only part of the workload can be spread across more DB instances, reducing the memory usage on each reader DB instance. For more information, see [Important Amazon CloudWatch metrics for Aurora Serverless v2 \(p. 1539\)](#).

For Aurora Serverless v1, you can change the capacity range to use more ACUs. For more information, see [Modifying an Aurora Serverless v1 DB cluster \(p. 1570\)](#).

## 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 \(p. 1766\)](#)
- [Diagnosing and resolving a MySQL read replication failure \(p. 1767\)](#)
- [Replication stopped error \(p. 1768\)](#)

## 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 \(p. 1767\)](#). 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 \(p. 594\)](#), [RDS-EVENT-0046 \(p. 594\)](#), and [RDS-EVENT-0047 \(p. 594\)](#). For more information about events and subscribing to events, see [Working with Amazon RDS event notification \(p. 572\)](#). 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 \(p. 1769\)](#)
- [Troubleshooting applications on Aurora \(p. 1770\)](#)

## 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 \(p. 1770\)](#)
- [Troubleshooting tips \(p. 1770\)](#)

For information about troubleshooting for Amazon RDS DB instances, see [Troubleshooting for Amazon Aurora \(p. 1761\)](#).

## 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 AWS re:Post

Amazon RDS has a development community where you can search for solutions to problems others have experienced along the way. To view the topics, go to [AWS re:Post](#).

# 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
<a href="#">Amazon Aurora supports automatically setting up connectivity with an EC2 instance (p. 1771)</a>	When you create an Aurora DB cluster, you can use the AWS Management Console to set up connectivity between an Amazon Elastic Compute Cloud instance and the new DB cluster. For more information, see <a href="#">Configure automatic network connectivity with an EC2 instance</a> .	August 22, 2022
<a href="#">Amazon Aurora supports dual-stack mode (p. 1771)</a>	DB clusters can now run in dual-stack mode. In dual-stack mode, resources can communicate with the DB cluster over IPv4, IPv6, or both. For more information, see <a href="#">Amazon Aurora IP addressing</a> .	August 17, 2022
<a href="#">Amazon Aurora supports in-place upgrade for PostgreSQL-compatible Aurora Serverless v1 (p. 1771)</a>	You can perform an in-place upgrade for a PostgreSQL 10-compatible Aurora Serverless v1 cluster to change an existing cluster into a PostgreSQL 11-compatible Aurora Serverless v1 cluster. For the in-place upgrade procedure, see <a href="#">Modifying an Aurora Serverless v1 DB cluster</a> .	August 8, 2022
<a href="#">Performance Insights supports the Asia Pacific (Jakarta) Region (p. 1771)</a>	Formerly, you couldn't use Performance Insights in the Asia Pacific (Jakarta) Region. This restriction has been removed. For more information, see <a href="#">AWS Region support for Performance Insights</a> .	July 21, 2022
<a href="#">Amazon Aurora supports a new DB instance class (p. 1771)</a>	You can now use the db.r6i DB instance class for Aurora.	July 14, 2022

	PostgreSQL DB clusters. For more information, see <a href="#">DB instance classes</a> .	
<a href="#">RDS Performance Insights supports additional retention periods (p. 1771)</a>	Previously, Performance Insights offered only two retention periods: 7 days (default) or 2 years (731 days). Now, if you need to retain your performance data for longer than 7 days, you can specify from 1–24 months. For more information, see <a href="#">Pricing and data retention for Performance Insights</a> .	July 1, 2022
<a href="#">Amazon Aurora supports in-place upgrade for MySQL-compatible Aurora Serverless v1 (p. 1771)</a>	You can perform an in-place upgrade for a MySQL 5.6-compatible Aurora Serverless v1 cluster to change an existing cluster into a MySQL 5.7-compatible Aurora Serverless v1 cluster. For the in-place upgrade procedure, see <a href="#">Modifying an Aurora Serverless v1 DB cluster</a> .	June 16, 2022
<a href="#">Aurora supports turning on Amazon DevOps Guru in the RDS console (p. 1771)</a>	You can turn on DevOps Guru coverage for your Aurora databases from within the RDS console. For more information, see <a href="#">Setting up DevOps Guru for RDS</a> .	June 9, 2022
<a href="#">Amazon Aurora supports publishing events to encrypted Amazon SNS topics (p. 1771)</a>	Amazon Aurora can now publish events to Amazon Simple Notification Service (Amazon SNS) topics that have server-side encryption (SSE) enabled, for additional protection of events that carry sensitive data. For more information, see <a href="#">Subscribing to Amazon RDS event notification</a> .	June 1, 2022
<a href="#">Amazon RDS publishes usage metrics to Amazon CloudWatch (p. 1771)</a>	The AWS/Usage namespace in Amazon CloudWatch includes account-level usage metrics for your Amazon RDS service quotas. For more information, see <a href="#">Amazon CloudWatch usage metrics for Amazon Aurora</a> .	April 28, 2022

<a href="#">Data API result sets in JSON format (p. 1771)</a>	An optional parameter for the <code>ExecuteStatement</code> function causes the query result set to be returned as a string in JSON format. The JSON result set is simple and convenient to transform into a data structure in your application's language. For more information, see <a href="#">Processing query results in JSON format</a> .	April 27, 2022
<a href="#">Amazon Aurora Serverless v2 is now generally available (p. 1771)</a>	Amazon Aurora Serverless v2 is generally available for all users. For more information, see <a href="#">Using Aurora Serverless v2</a> .	April 21, 2022
<a href="#">Aurora MySQL supports configurable cipher suites (p. 1771)</a>	With Aurora MySQL, you can now use configurable cipher suites to control the connection encryption that your database server accepts. For more information, see <a href="#">Configuring cipher suites for connections to Aurora MySQL DB clusters</a> .	April 15, 2022
<a href="#">Aurora PostgreSQL supports RDS Proxy with PostgreSQL 13 (p. 1771)</a>	You can now create an RDS Proxy with an Aurora PostgreSQL 13 DB cluster. For more information about RDS Proxy, see <a href="#">Using Amazon RDS Proxy</a> .	April 4, 2022
<a href="#">Release Notes for Aurora PostgreSQL (p. 1771)</a>	There is now a separate guide for the Amazon Aurora PostgreSQL release notes. For more information, see <a href="#">Release Notes for Aurora PostgreSQL</a> .	March 22, 2022
<a href="#">Release Notes for Aurora MySQL (p. 1771)</a>	There is now a separate guide for the Amazon Aurora MySQL release notes. For more information, see <a href="#">Release Notes for Aurora MySQL</a> .	March 22, 2022
<a href="#">Aurora PostgreSQL supports multi-major version upgrades (p. 1771)</a>	You can now perform version upgrades of Aurora PostgreSQL DB clusters across multiple major versions. For more information, see <a href="#">How to perform a major version upgrade</a> .	March 4, 2022

Aurora PostgreSQL supports configurable cipher suites (p. 1771)	With Aurora PostgreSQL versions 11.8 and higher, you can now use configurable cipher suites to control the connection encryption that your database server accepts. For information about using configurable cipher suites with Aurora PostgreSQL, see <a href="#">Configuring cipher suites for connections to Aurora PostgreSQL DB clusters</a> .	March 4, 2022
AWS JDBC Driver for MySQL generally available (p. 1771)	The AWS JDBC Driver for MySQL is a client driver designed for the high availability of Aurora MySQL. The AWS JDBC Driver for MySQL is now generally available. For more information, see <a href="#">Connecting with the Amazon Web Services JDBC Driver for MySQL</a> .	March 2, 2022
Aurora PostgreSQL 13.5 supports Babelfish for Aurora PostgreSQL 1.1.0 (p. 1771)	Aurora PostgreSQL 13.5 supports Babelfish 1.1.0. For a list of new features, see <a href="#">13.5</a> . For a list of features supported in each Babelfish release, see <a href="#">Supported functionality in Babelfish by version</a> . For usage information, see <a href="#">Working with Babelfish for Aurora PostgreSQL</a> .	February 28, 2022
Amazon Aurora supports Database Activity Streams in the Asia Pacific (Jakarta) Region (p. 1771)	For more information, see <a href="#">Support for AWS Regions for database activity streams</a> .	February 16, 2022
Performance Insights supports new API operations (p. 1771)	Performance Insights now supports the following API operations: <code>GetResourceMetadata</code> , <code>ListAvailableResourceDimensions</code> , and <code>ListAvailableResourceMetrics</code> . For more information, see <a href="#">Retrieving metrics with the Performance Insights API</a> in this manual and the <a href="#">Amazon RDS Performance Insights API Reference</a> .	January 12, 2022
Amazon RDS Proxy supports events (p. 1771)	RDS Proxy now generates events that you can subscribe to and view in CloudWatch Events or configure to send to Amazon EventBridge. For more information, see <a href="#">Working with RDS Proxy events</a> .	January 11, 2022

RDS Proxy available in additional AWS Regions (p. 1771)	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 <a href="#">Using Amazon RDS Proxy</a> .	January 5, 2022
Amazon Aurora available in the Asia Pacific (Jakarta) Region (p. 1771)	Aurora is now available in the Asia Pacific (Jakarta) Region. For more information, see <a href="#">Regions and Availability Zones</a> .	December 13, 2021
DevOps Guru for Amazon RDS provides detailed insights and recommendations for Amazon Aurora (p. 1771)	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 <a href="#">Analyzing performance anomalies with DevOps Guru for RDS</a> in this guide and see <a href="#">Overview of DevOps Guru for RDS</a> in the <a href="#">Amazon DevOps Guru User Guide</a> .	December 1, 2021
Aurora PostgreSQL supports RDS Proxy with PostgreSQL 12 (p. 1771)	You can now create an RDS Proxy with an Aurora PostgreSQL 12 database cluster. For more information about RDS Proxy, see <a href="#">Using Amazon RDS Proxy</a> .	November 22, 2021
Aurora supports AWS Graviton2 instance classes for Database Activity Streams (p. 1771)	You can use database activity streams with the db.r6g instance class for Aurora MySQL and Aurora PostgreSQL. For more information, see <a href="#">Supported DB instance classes</a> .	November 3, 2021
Amazon Aurora support for cross-account AWS KMS keys (p. 1771)	You can use a KMS key from a different AWS account for encryption when exporting DB snapshots to Amazon S3. For more information, see <a href="#">Exporting DB snapshot data to Amazon S3</a> .	November 3, 2021

<a href="#">Amazon Aurora supports Babelfish for Aurora PostgreSQL (p. 1771)</a>	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 <a href="#">Working with Babelfish for Aurora PostgreSQL</a> .	October 28, 2021
<a href="#">Aurora Serverless v1 can require SSL for connections (p. 1771)</a>	The Aurora cluster parameters <code>force_ssl</code> for PostgreSQL and <code>require_secure_transport</code> for MySQL are supported now for Aurora Serverless v1. For more information, see <a href="#">Using TLS/SSL with Aurora Serverless v1</a> .	October 26, 2021
<a href="#">Amazon Aurora supports Performance Insights in additional AWS Regions (p. 1771)</a>	Performance Insights is available in the Middle East (Bahrain), Africa (Cape Town), Europe (Milan), and Asia Pacific (Osaka) Regions. For more information, see <a href="#">AWS Region support for Performance Insights</a> .	October 5, 2021
<a href="#">Configurable autoscaling timeout for Aurora Serverless v1 (p. 1771)</a>	You can choose how long Aurora Serverless v1 waits to find an autoscaling point. If no autoscaling point is found during that period, Aurora Serverless v1 cancels the scaling event or forces the capacity change, depending on the timeout action that you selected. For more information, see <a href="#">Autoscaling for Aurora Serverless v1</a> .	September 10, 2021
<a href="#">Aurora supports X2g and T4g instance classes (p. 1771)</a>	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 <a href="#">DB instance classes</a> .	September 10, 2021
<a href="#">Amazon RDS supports RDS Proxy in a shared VPC (p. 1771)</a>	You can now create an RDS Proxy in a shared virtual private cloud (VPC). For more information about RDS Proxy, see "Managing Connections with Amazon RDS Proxy" in the <a href="#">Amazon RDS User Guide</a> or the <a href="#">Aurora User Guide</a> .	August 6, 2021

Aurora version policy page (p. 1771)	The <i>Amazon Aurora User Guide</i> now includes a section with general information about Aurora versions and associated policies. For details, see <a href="#">Amazon Aurora versions</a> .	July 14, 2021
Exclude Data API events from an AWS CloudTrail trail (p. 1771)	You can exclude Data API events from a CloudTrail trail. For more information, see <a href="#">Excluding Data API events from an AWS CloudTrail trail</a> .	July 2, 2021
Amazon Aurora PostgreSQL-Compatible Edition supports additional extensions (p. 1771)	Newly supported extensions include pg_bigm, pg_cron, pg_partman, and pg_proctab. For more information, see <a href="#">Extension versions for Amazon Aurora PostgreSQL-Compatible Edition</a> .	June 17, 2021
Cloning for Aurora Serverless clusters (p. 1771)	You can now create cloned clusters that are Aurora Serverless. For information about cloning, see <a href="#">Cloning a volume for an Aurora DB cluster</a> .	June 16, 2021
Aurora global databases available in China (Beijing) and China (Ningxia) Regions (p. 1771)	You can now create Aurora global databases in the China (Beijing) and China (Ningxia) Regions. For information about Aurora global databases, see <a href="#">Working with Amazon Aurora global databases</a> .	May 19, 2021
FIPS 140-2 support for Data API (p. 1771)	The Data API supports the Federal Information Processing Standard Publication 140-2 (FIPS 140-2) for SSL/TLS connections. For more information, see <a href="#">Data API availability</a> .	May 14, 2021
AWS JDBC Driver for PostgreSQL (preview) (p. 1771)	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 <a href="#">Connecting with the Amazon Web Services JDBC Driver for PostgreSQL (preview)</a> .	April 27, 2021
The Data API available in additional AWS Regions (p. 1771)	The Data API is now available in the Asia Pacific (Seoul) and Canada (Central) Regions. For more information, see <a href="#">Availability of the Data API</a> .	April 9, 2021

Amazon Aurora supports the Graviton2 DB instance classes (p. 1771)	You can now use the Graviton2 DB instance classes db.r6g.x to create DB clusters running MySQL or PostgreSQL. For more information, see <a href="#">DB instance class types</a> .	March 12, 2021
RDS Proxy endpoint enhancements (p. 1771)	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 <a href="#">Amazon RDS User Guide</a> or the <a href="#">Aurora user guide</a> .	March 8, 2021
Amazon Aurora available in the Asia Pacific (Osaka) Region (p. 1771)	Aurora is now available in the Asia Pacific (Osaka) Region. For more information, see <a href="#">Regions and Availability Zones</a> .	March 1, 2021
Aurora PostgreSQL supports enabling both IAM and Kerberos authentication on the same DB cluster (p. 1771)	Aurora PostgreSQL now supports enabling both IAM authentication and Kerberos authentication on the same DB cluster. For more information, see <a href="#">Database authentication with Amazon Aurora</a> .	February 24, 2021
Aurora global database now supports managed planned failover (p. 1771)	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 <a href="#">Disaster recovery and Amazon Aurora global databases</a> . For reference information, see <a href="#">FailoverGlobalCluster</a> in the <a href="#">Amazon RDS API Reference</a> .	February 11, 2021

<a href="#">Data API for Aurora Serverless now supports more data types (p. 1771)</a>	With the Data API for Aurora Serverless, 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, you can now have a <code>LONG</code> type value returned from your database as a <code>STRING</code> value. To learn more, see <a href="#">Calling the Data API</a> . For reference information about supported data types, see <a href="#">SqlParameter</a> in the <i>Amazon RDS Data Service API Reference</i> .	February 2, 2021
<a href="#">Aurora PostgreSQL supports major version upgrades to PostgreSQL 12 (p. 1771)</a>	With Aurora PostgreSQL, you can now upgrade the DB engine to major version 12. For more information, see <a href="#">Upgrading the PostgreSQL DB engine for Aurora PostgreSQL</a> .	January 28, 2021
<a href="#">Aurora MySQL supports in-place upgrade (p. 1771)</a>	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 <a href="#">Upgrading the major version of an Aurora MySQL DB cluster from 1.x to 2.x</a> .	January 11, 2021
<a href="#">AWS JDBC Driver for MySQL (preview) (p. 1771)</a>	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 <a href="#">Connecting with the Amazon Web Services JDBC Driver for MySQL (preview)</a> .	January 7, 2021
<a href="#">Aurora supports database activity streams on secondary clusters of a global database (p. 1771)</a>	You can start a database activity stream on a primary or secondary cluster of Aurora PostgreSQL or Aurora MySQL. For supported engine versions, see <a href="#">Limitations of Aurora global databases</a> .	December 22, 2020

<a href="#">Multi-master clusters with 4 DB instances (p. 1771)</a>	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 <a href="#">Working with Aurora Multi-Master Clusters</a> .	December 17, 2020
<a href="#">Aurora PostgreSQL supports AWS Lambda functions (p. 1771)</a>	You can now invoke AWS Lambda function for your Aurora PostgreSQL DB clusters. For more information, see <a href="#">Invoking a Lambda function from an Aurora PostgreSQL DB cluster</a> .	December 11, 2020
<a href="#">Amazon Aurora supports the Graviton2 DB instance classes in preview (p. 1771)</a>	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 <a href="#">DB instance class types</a> .	December 11, 2020
<a href="#">Amazon Aurora Serverless v2 is now available in preview. (p. 1771)</a>	Amazon Aurora Serverless v2 is available in preview. To work with Amazon Aurora Serverless v2, apply for access. For more information, see the <a href="#">Aurora Serverless v2 page</a> .	December 1, 2020
<a href="#">Aurora PostgreSQL is now available for Aurora Serverless in more AWS Regions. (p. 1771)</a>	Aurora PostgreSQL is now available for Aurora Serverless in more AWS Regions. You can now choose to run Aurora PostgreSQL Serverless v1 in the same AWS Regions that offer Aurora MySQL Serverless v1. Additional AWS Regions with Aurora Serverless 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, see <a href="#">Aurora Serverless</a> . Amazon RDS Data API for Aurora Serverless is also now available in these same AWS Regions. For a list of all Regions with support for the Data API for Aurora Serverless, see <a href="#">Data API for Aurora Serverless</a>	November 24, 2020

Amazon RDS Performance Insights introduces new dimensions (p. 1771)	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 <a href="#">Top load table</a> .	November 24, 2020
Aurora Serverless supports Aurora PostgreSQL version 10.12 (p. 1771)	Aurora PostgreSQL for Aurora Serverless has been upgraded to Aurora PostgreSQL version 10.12 throughout the AWS Regions where Aurora PostgreSQL for Aurora Serverless is supported. For more information, see <a href="#">Aurora Serverless</a> .	November 4, 2020
The Data API now supports tag-based authorization (p. 1771)	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 <a href="#">Authorizing access to the Data API</a> .	October 27, 2020
Amazon Aurora extends support for exporting snapshots to Amazon S3 (p. 1771)	You can now export DB snapshot data to Amazon S3 in all commercial AWS Regions. For more information, see <a href="#">Exporting DB snapshot data to Amazon S3</a> .	October 22, 2020
Aurora global database supports cloning (p. 1771)	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 <b>Create clone</b> 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 <a href="#">Cloning an Aurora DB cluster volume</a> .	October 19, 2020

Amazon Aurora supports dynamic resizing for the cluster volume (p. 1771)	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 <a href="#">Managing Performance and Scaling for Aurora DB Clusters</a> .	October 13, 2020
Amazon Aurora supports volume sizes up to 128 TiB (p. 1771)	New and existing Aurora cluster volumes can now grow to a maximum size of 128 tebibytes (TiB). For more information, see <a href="#">How Aurora storage grows</a> .	September 22, 2020
Aurora PostgreSQL supports the db.r5 and db.t3 DB instance classes in the China (Ningxia) Region (p. 1771)	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 <a href="#">DB instance classes</a> .	September 3, 2020

[Aurora parallel query enhancements \(p. 1771\)](#)

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 `aurora_parallel_query` 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 `DYNAMIC` 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 `READ COMMITTED` 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

[Changes to Aurora MySQL parameter binlog\\_rows\\_query\\_log\\_events \(p. 210\)](#)

You can now change the value of the Aurora MySQL configuration parameter `binlog_rows_query_log_events`. Formerly, this parameter wasn't modifiable.

August 26, 2020

<a href="#">Support for automatic minor version upgrades for Aurora MySQL (p. 1771)</a>	With Aurora MySQL, the setting <b>Enable auto minor version upgrade</b> 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 <a href="#">Database Upgrades and Patches for Amazon Aurora MySQL</a> .	August 3, 2020
<a href="#">Aurora PostgreSQL supports major version upgrades to PostgreSQL version 11 (p. 1771)</a>	With Aurora PostgreSQL, you can now upgrade the DB engine to major version 11. For more information, see <a href="#">Upgrading the PostgreSQL DB engine for Aurora PostgreSQL</a> .	July 28, 2020
<a href="#">Amazon Aurora supports AWS PrivateLink (p. 1771)</a>	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 <a href="#">Amazon Aurora and interface VPC endpoints (AWS PrivateLink)</a> .	July 9, 2020
<a href="#">RDS Proxy generally available (p. 1771)</a>	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 <a href="#">Amazon RDS User Guide</a> or the Aurora user guide.	June 30, 2020

Aurora global database write forwarding (p. 1771)	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 <a href="#">Write forwarding for secondary AWS Regions with an Aurora global database</a> .	June 18, 2020
Aurora supports integration with AWS Backup (p. 1771)	You can use AWS Backup to manage backups of Aurora DB clusters. For more information, see <a href="#">Overview of backing up and restoring an Aurora DB cluster</a> .	June 10, 2020
Aurora PostgreSQL supports db.t3.large DB instance classes (p. 1771)	You can now create Aurora PostgreSQL DB clusters that use the db.t3.large DB instance classes. For more information, see <a href="#">DB instance classes</a> .	June 5, 2020
Aurora global database supports PostgreSQL version 11.7 and managed recovery point objective (RPO) (p. 1771)	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 <a href="#">Cross-Region Disaster Recovery for Aurora global databases</a> .	June 4, 2020
Aurora MySQL supports database monitoring with database activity streams (p. 1771)	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 <a href="#">Using database activity streams</a> .	June 2, 2020
The query editor available in additional AWS Regions (p. 1771)	The query editor for Aurora Serverless is now available in additional AWS Regions. For more information, see <a href="#">Availability of the query editor</a> .	May 28, 2020
The Data API available in additional AWS Regions (p. 1771)	The Data API is now available in additional AWS Regions. For more information, see <a href="#">Availability of the Data API</a> .	May 28, 2020

RDS Proxy available in Canada (Central) Region (p. 1771)	You can now use the RDS Proxy preview in the Canada (Central) Region. For more information about RDS Proxy, see <a href="#">Managing connections with Amazon RDS proxy (preview)</a> .	May 28, 2020
Aurora global database and cross-Region read replicas (p. 1771)	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 <a href="#">Working with Amazon Aurora global database and Replicating Amazon Aurora MySQL DB</a> .	May 18, 2020
RDS Proxy available in more AWS Regions (p. 1771)	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 <a href="#">Managing connections with Amazon RDS proxy (preview)</a> .	May 13, 2020
Aurora PostgreSQL-Compatible Edition supports on-premises or self-hosted Microsoft active directory (p. 1771)	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 <a href="#">Using Kerberos authentication with Aurora PostgreSQL</a> .	May 7, 2020
Aurora MySQL multi-master clusters available in more AWS Regions (p. 1771)	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 <a href="#">Working with Aurora multi-master clusters</a> .	May 7, 2020

<a href="#">Performance Insights supports analyzing statistics of running Aurora MySQL queries (p. 1771)</a>	You can now analyze statistics of running queries with Performance Insights for Aurora MySQL DB instances. For more information, see <a href="#">Analyzing statistics of running queries</a> .	May 5, 2020
<a href="#">Java client library for Data API generally available (p. 1771)</a>	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 <a href="#">Using the Java client library for Data API</a> .	April 30, 2020
<a href="#">Amazon Aurora available in the Europe (Milan) Region (p. 1771)</a>	Aurora is now available in the Europe (Milan) Region. For more information, see <a href="#">Regions and Availability Zones</a> .	April 28, 2020
<a href="#">Amazon Aurora available in the Europe (Milan) Region (p. 1771)</a>	Aurora is now available in the Europe (Milan) Region. For more information, see <a href="#">Regions and Availability Zones</a> .	April 27, 2020
<a href="#">Amazon Aurora available in the Africa (Cape Town) Region (p. 1771)</a>	Aurora is now available in the Africa (Cape Town) Region. For more information, see <a href="#">Regions and Availability Zones</a> .	April 22, 2020
<a href="#">Aurora PostgreSQL now supports db.r5.16xlarge and db.r5.8xlarge DB instance classes (p. 1771)</a>	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 <a href="#">Hardware specifications for DB instance classes for Aurora</a> .	April 8, 2020
<a href="#">Amazon RDS proxy for PostgreSQL (p. 1771)</a>	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 <a href="#">Managing connections with Amazon RDS proxy (preview)</a> .	April 8, 2020

Aurora global databases now support Aurora PostgreSQL (p. 1771)	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 <a href="#">Working with Amazon Aurora global database</a> .	March 10, 2020
Support for major version upgrades for Aurora PostgreSQL (p. 1771)	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 <a href="#">Upgrading the PostgreSQL DB engine for Aurora PostgreSQL</a> .	March 4, 2020
Aurora PostgreSQL supports Kerberos authentication (p. 1771)	You can now use Kerberos authentication to authenticate users when they connect to your Aurora PostgreSQL DB clusters. For more information, see <a href="#">Using Kerberos authentication with Aurora PostgreSQL</a> .	February 28, 2020
The Data API supports AWS PrivateLink (p. 1771)	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 <a href="#">Creating an Amazon VPC endpoint (AWS PrivateLink) for the Data API</a> .	February 6, 2020
Aurora machine learning support in Aurora PostgreSQL (p. 1771)	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 <a href="#">Using machine learning (ML) capabilities with Aurora</a> .	February 5, 2020

Aurora PostgreSQL supports exporting data to Amazon S3 (p. 1771)	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 <a href="#">Exporting data from an Aurora PostgreSQL DB cluster to Amazon S3</a> .	February 5, 2020
Support for exporting DB snapshot data to Amazon S3 (p. 1771)	Amazon Aurora supports exporting DB snapshot data to Amazon S3 for MySQL and PostgreSQL. For more information, see <a href="#">Exporting DB snapshot data to Amazon S3</a> .	January 9, 2020
Aurora MySQL release notes in document history (p. 1771)	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. 1771)	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 <a href="#">Managing connections with Amazon RDS proxy (preview)</a> .	December 3, 2019
Data API for Aurora Serverless v1 supports data type mapping hints (p. 1771)	You can now use a hint to instruct the Data API for Aurora Serverless v1 to send a String value to the database as a different type. For more information, see <a href="#">Calling the data API</a> .	November 26, 2019

<a href="#">Data API for Aurora Serverless v1 supports a Java client library (preview) (p. 1771)</a>	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 <a href="#">Using the Java client library for Data API</a> .	November 26, 2019
<a href="#">Aurora PostgreSQL is FedRAMP HIGH eligible (p. 1771)</a>	Aurora PostgreSQL is FedRAMP HIGH eligible. For details about AWS and compliance efforts, see <a href="#">AWS services in scope by compliance program</a> .	November 26, 2019
<a href="#">READ COMMITTED isolation level enabled for Amazon Aurora MySQL replicas (p. 1771)</a>	You can now enable the <code>READ COMMITTED</code> 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 <code>READ COMMITTED</code> 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 <code>READ COMMITTED</code> . For more information, see <a href="#">Aurora MySQL isolation levels</a> .	November 25, 2019
<a href="#">Performance Insights supports analyzing statistics of running Aurora PostgreSQL queries (p. 1771)</a>	You can now analyze statistics of running queries with Performance Insights for Aurora PostgreSQL DB instances. For more information, see <a href="#">Analyzing statistics of running queries</a> .	November 25, 2019
<a href="#">More clusters in an Aurora global database (p. 1771)</a>	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 <a href="#">Working with Amazon Aurora global databases</a> .	November 25, 2019

Aurora machine learning support in Aurora MySQL (p. 1771)	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 <a href="#">Using machine learning (ML) capabilities with Aurora</a> .	November 25, 2019
Aurora global database no longer requires engine mode setting (p. 1771)	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 <a href="#">Working with Amazon Aurora global databases</a> .	November 25, 2019
Aurora global database is available for Aurora MySQL version 2 (p. 1771)	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 <a href="#">Working with Amazon Aurora global database</a> .	November 25, 2019
Aurora MySQL hot row contention optimization available without lab mode (p. 1771)	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. 1771)	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 <a href="#">Working with hash joins in Aurora MySQL</a> .	November 19, 2019
Aurora MySQL 2.* support for more db.r5 instance classes (p. 1771)	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 <a href="#">Choosing the DB instance class</a> .	November 19, 2019
Aurora MySQL 2.* support for backtracking (p. 1771)	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 <a href="#">Backtracking an Aurora DB cluster</a> .	November 19, 2019
Billing tag support for Aurora (p. 1771)	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 <a href="#">Tagging Amazon RDS resources</a> . For general information about tags and ways to use them for cost analysis, see <a href="#">Using cost allocation tags</a> and <a href="#">User-defined cost allocation tags</a> .	October 23, 2019
Data API for Aurora PostgreSQL (p. 1771)	Aurora PostgreSQL now supports using the Data API with Amazon Aurora Serverless v1 DB clusters. For more information, see <a href="#">Using the Data API for Aurora Serverless v1</a> .	September 23, 2019

Aurora PostgreSQL supports uploading database logs to CloudWatch logs (p. 1771)	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 <a href="#">Publishing Aurora PostgreSQL logs to Amazon CloudWatch Logs</a> .	August 9, 2019
Multi-master clusters for Aurora MySQL (p. 1771)	You can set up Aurora MySQL multi-master clusters. In these clusters, each DB instance has read/write capability. For more information, see <a href="#">Working with Aurora multi-master clusters</a> .	August 8, 2019
Aurora PostgreSQL supports Aurora Serverless v1 (p. 1771)	You can now use Amazon Aurora Serverless v1 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 <a href="#">Using Amazon Aurora Serverless v1</a> .	July 9, 2019
Cross-account cloning for Aurora MySQL (p. 1771)	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 <a href="#">Cloning databases in an Aurora DB cluster</a> .	July 2, 2019
Aurora PostgreSQL supports db.t3 DB instance classes (p. 1771)	You can now create Aurora PostgreSQL DB clusters that use the db.t3 DB instance classes. For more information, see <a href="#">DB instance class</a> .	June 20, 2019

<a href="#">Support for importing data from Amazon S3 for Aurora PostgreSQL (p. 1771)</a>	You can now import data from an Amazon S3 file into a table in an Aurora PostgreSQL DB cluster. For more information, see <a href="#">Importing Amazon S3 data into an Aurora PostgreSQL DB cluster</a> .	June 19, 2019
<a href="#">Aurora PostgreSQL now provides fast failover recovery with cluster cache management (p. 1771)</a>	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 <a href="#">Fast recovery after failover with cluster cache management</a> .	June 11, 2019
<a href="#">Data API for Aurora Serverless v1 generally available (p. 1771)</a>	You can access Aurora Serverless v1 clusters with web services-based applications using the Data API. For more information, see <a href="#">Using the Data API for Aurora Serverless v1</a> .	May 30, 2019
<a href="#">Aurora PostgreSQL supports database monitoring with database activity streams (p. 1771)</a>	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 <a href="#">Using database activity streams</a> .	May 30, 2019
<a href="#">Amazon Aurora recommendations (p. 1771)</a>	Amazon Aurora now provides automated recommendations for Aurora resources. For more information, see <a href="#">Using Amazon Aurora recommendations</a> .	May 22, 2019
<a href="#">Performance Insights support for Aurora global database (p. 1771)</a>	You can now use Performance Insights with Aurora Global Database. For information about Performance Insights for Aurora, see <a href="#">Using Amazon RDS performance insights</a> . For information about Aurora global databases, see <a href="#">Working with Aurora global database</a> .	May 13, 2019
<a href="#">Performance Insights is available for Aurora MySQL 5.7 (p. 1771)</a>	Amazon RDS Performance Insights is now available for Aurora MySQL 2.x versions, which are compatible with MySQL 5.7. For more information, see <a href="#">Using Amazon RDS performance insights</a> .	May 3, 2019

Aurora global databases available in more AWS Regions (p. 1771)	You can now create Aurora global databases in most AWS Regions where Aurora is available. For information about Aurora global databases, see <a href="#">Working with Amazon Aurora global databases</a> .	April 30, 2019
Minimum capacity of 1 for Aurora Serverless v1 (p. 1771)	The minimum capacity setting you can use for an Aurora Serverless v1 cluster is 1. Formerly, the minimum was 2. For information about specifying Aurora Serverless capacity values, see <a href="#">Setting the capacity of an Aurora Serverless v1 DB cluster</a> .	April 29, 2019
Aurora Serverless v1 timeout action (p. 1771)	You can now specify the action to take when an Aurora Serverless v1 capacity change times out. For more information, see <a href="#">Timeout action for capacity changes</a> .	April 29, 2019
Per-second billing (p. 1771)	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 <a href="#">DB instance billing for Aurora</a> .	April 25, 2019
Sharing Aurora Serverless v1 snapshots across AWS Regions (p. 1771)	With Aurora Serverless v1, 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 v1 DB clusters, see <a href="#">Aurora Serverless v1 and snapshots</a> .	April 17, 2019
Restore MySQL 5.7 backups from Amazon S3 (p. 1771)	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 <a href="#">Migrating data from an external MySQL database to an Aurora MySQL DB cluster</a> .	April 17, 2019

<a href="#">Sharing Aurora Serverless v1 snapshots across regions (p. 1771)</a>	With Aurora Serverless v1, 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 v1 DB clusters, see <a href="#">Aurora Serverless and snapshots</a> .	April 16, 2019
<a href="#">Aurora proof-of-concept tutorial (p. 1771)</a>	You can learn how to perform a proof of concept to try your application and workload with Aurora. For the full tutorial, see <a href="#">Performing an Aurora proof of concept</a> .	April 16, 2019
<a href="#">Aurora Serverless v1 supports restoring from an Amazon S3 backup (p. 1771)</a>	You can now import backups from Amazon S3 to an Aurora Serverless cluster. For details about that procedure, see <a href="#">Migrating data from MySQL by using an Amazon S3 bucket</a> .	April 16, 2019
<a href="#">New modifiable parameters for Aurora Serverless v1 (p. 1771)</a>	You can now modify the following DB parameters for an Aurora Serverless v1 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 v1 clusters, see <a href="#">Aurora Serverless v1 and parameter groups</a> .	April 4, 2019
<a href="#">Aurora PostgreSQL supports db.r5 DB instance classes (p. 1771)</a>	You can now create Aurora PostgreSQL DB clusters that use the db.r5 DB instance classes. For more information, see <a href="#">DB instance class</a> .	April 4, 2019

Aurora PostgreSQL logical replication (p. 1771)	You can now use PostgreSQL logical replication to replicate parts of a database for an Aurora PostgreSQL DB cluster. For more information, see <a href="#">Using PostgreSQL logical replication</a> .	March 28, 2019
GTID support for Aurora MySQL 2.04 (p. 1771)	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 <a href="#">Using GTID-based replication for Aurora MySQL</a> .	March 25, 2019
Uploading Aurora Serverless v1 logs to Amazon CloudWatch (p. 1771)	You can now have Aurora upload database logs to CloudWatch for an Aurora Serverless v1 cluster. For more information, see <a href="#">Viewing Aurora Serverless DB clusters</a> . 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 <a href="#">Working with DB parameter groups and DB cluster parameter groups</a> .	February 25, 2019
Aurora MySQL supports db.t3 DB instance classes (p. 1771)	You can now create Aurora MySQL DB clusters that use the db.t3 DB instance classes. For more information, see <a href="#">DB instance class</a> .	February 25, 2019
Aurora MySQL supports db.r5 DB instance classes (p. 1771)	You can now create Aurora MySQL DB clusters that use the db.r5 DB instance classes. For more information, see <a href="#">DB instance class</a> .	February 25, 2019

<a href="#">Performance Insights counters for Aurora MySQL (p. 1771)</a>	You can now add performance counters to your Performance Insights charts for Aurora MySQL DB instances. For more information, see <a href="#">Performance Insights dashboard components</a> .	February 19, 2019
<a href="#">Amazon RDS Performance Insights supports viewing more SQL text for Aurora MySQL (p. 1771)</a>	Amazon RDS Performance Insights now supports viewing more SQL text in the Performance Insights dashboard for Aurora MySQL DB instances. For more information, see <a href="#">Viewing more SQL text in the Performance Insights dashboard</a> .	February 6, 2019
<a href="#">Amazon RDS Performance Insights supports viewing more SQL text for Aurora PostgreSQL (p. 1771)</a>	Amazon RDS Performance Insights now supports viewing more SQL text in the Performance Insights dashboard for Aurora PostgreSQL DB instances. For more information, see <a href="#">Viewing more SQL text in the Performance Insights dashboard</a> .	January 24, 2019
<a href="#">Aurora backup billing (p. 1771)</a>	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 <a href="#">Overview of monitoring</a> . For more information about how to manage storage for backup data, see <a href="#">Understanding Aurora backup storage usage</a> .	January 3, 2019
<a href="#">Performance Insights counters (p. 1771)</a>	You can now add performance counters to your Performance Insights charts. For more information, see <a href="#">Performance Insights dashboard components</a> .	December 6, 2018
<a href="#">Aurora global database (p. 1771)</a>	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 <a href="#">Working with Amazon Aurora global database</a> .	November 28, 2018

<a href="#">Query plan management in Aurora PostgreSQL (p. 1771)</a>	Aurora PostgreSQL now provides query plan management that you can use to manage PostgreSQL query execution plans. For more information, see <a href="#">Managing query execution plans for Aurora PostgreSQL</a> .	November 20, 2018
<a href="#">Query editor for Aurora Serverless v1 (beta) (p. 1771)</a>	You can run SQL statements in the Amazon RDS console on Aurora Serverless v1 clusters. For more information, see <a href="#">Using the query editor for Aurora Serverless v1</a> .	November 20, 2018
<a href="#">Data API for Aurora Serverless v1 (beta) (p. 1771)</a>	You can access Aurora Serverless v1 clusters with web services-based applications using the Data API. For more information, see <a href="#">Using the Data API for Aurora Serverless</a> .	November 20, 2018
<a href="#">TLS support for Aurora Serverless v1 (p. 1771)</a>	Aurora Serverless v1 clusters support TLS/SSL encryption. For more information, see <a href="#">TLS/SSL for Aurora Serverless</a> .	November 19, 2018
<a href="#">Custom endpoints (p. 1771)</a>	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 <a href="#">Amazon Aurora connection management</a> .	November 12, 2018
<a href="#">IAM authentication support in Aurora PostgreSQL (p. 1771)</a>	Aurora PostgreSQL now supports IAM authentication. For more information see <a href="#">IAM database authentication</a> .	November 8, 2018
<a href="#">Custom parameter groups for restore and point in time recovery (p. 1771)</a>	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 <a href="#">Restoring from a DB cluster snapshot</a> and <a href="#">Restoring a DB cluster to a specified time</a> .	October 15, 2018

<a href="#">Deletion protection for Aurora DB clusters (p. 1771)</a>	When you enable deletion protection for a DB cluster, the database cannot be deleted by any user. For more information, see <a href="#">Deleting a DB cluster</a> .	September 26, 2018
<a href="#">Stop/Start feature Aurora (p. 1771)</a>	You can now stop or start an entire Aurora cluster with a single operation. For more information, see <a href="#">Stopping and starting an Aurora cluster</a> .	September 24, 2018
<a href="#">Parallel query feature for Aurora MySQL (p. 1771)</a>	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 <a href="#">Working with parallel query for Aurora MySQL</a> .	September 20, 2018
<a href="#">New guide (p. 1771)</a>	This is the first release of the <i>Amazon Aurora User Guide</i> .	August 31, 2018

# AWS glossary

For the latest AWS terminology, see the [AWS glossary](#) in the *AWS General Reference*.