
Amazon Aurora

User Guide for Aurora



Amazon Aurora: User Guide for Aurora

Copyright © 2019 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 Connection Management	4
Types of Aurora Endpoints	4
Viewing Endpoints	6
Using the Cluster Endpoint	6
Using the Reader Endpoint	7
Using Custom Endpoints	7
Creating a Custom Endpoint	9
Viewing Custom Endpoints	11
Editing a Custom Endpoint	16
Deleting a Custom Endpoint	18
End-to-End AWS CLI Example for Custom Endpoints	19
Using the Instance Endpoints	23
Endpoints and High Availability	23
Aurora Storage and Reliability	24
Overview of Aurora Storage	24
Cluster Volume Contents	24
Storage Growth	25
Data Billing	25
Reliability	25
Aurora Security	26
Using SSL with Aurora DB Clusters	27
High Availability for Aurora	27
Aurora Global Database	28
Overview of Global Database	28
Creating	30
Adding an AWS Region	35
Removing a Cluster	37
Deleting	39
Importing Data	41
Managing	42
Configuring	42
Connecting	44
Failover	44
Performance Insights for Global Database	45
Multiple Secondary Regions	45
Integration with Other AWS Services	45
Replication with Aurora	47
Aurora Replicas	47
Aurora MySQL	47
Aurora PostgreSQL	48
Setting Up Your Environment	49
Sign Up for AWS	49
Create an IAM User	49
Determine Requirements	51
Provide Access to the DB Cluster in the VPC by Creating a Security Group	52
Getting Started	54
Creating an Aurora MySQL DB Cluster and Connecting to It	54
Create an Aurora MySQL DB Cluster	54
Connect to an Instance in a DB Cluster	61
Delete the Sample DB Cluster, DB Subnet Group, and VPC	63
Creating an Aurora PostgreSQL DB Cluster and Connecting to It	63
Create an Aurora PostgreSQL DB Cluster	64

Connect to an Instance in an Aurora PostgreSQL DB Cluster	72
Delete the Sample DB Cluster, DB Subnet Group, and VPC	73
Configuring Your Aurora DB Cluster	75
Choosing the DB Instance Class	76
DB Instance Class Types	76
Terminology	76
Hardware Specifications	77
Choosing the Regions and Availability Zones	80
Region Availability	80
Local Time Zone for DB Clusters	83
DB Instance Billing for Aurora	87
On-Demand DB Instances	88
Reserved DB Instances	89
Creating a DB Cluster	100
Prerequisites	100
Creating a DB Cluster	101
Available Settings	110
Using Aurora Serverless	116
Advantages of Aurora Serverless	116
Use Cases for Aurora Serverless	116
Limitations of Aurora Serverless	117
Using TLS/SSL with Aurora Serverless	118
How Aurora Serverless Works	119
Creating an Aurora Serverless DB Cluster	124
Restoring an Aurora Serverless DB Cluster	129
Modifying an Aurora Serverless DB Cluster	132
Setting the Capacity of an Aurora Serverless DB Cluster	133
Viewing Aurora Serverless DB Clusters	135
Using the Data API	139
Logging Data API Calls with AWS CloudTrail	159
Using the Query Editor	161
Connecting to a DB Cluster	166
Aurora MySQL	166
Aurora PostgreSQL	168
Troubleshooting	171
Migrating Data to a DB Cluster	172
Aurora MySQL	172
Aurora PostgreSQL	172
Security	173
Data Protection	174
Data Encryption	175
Internetwork Traffic Privacy	190
Identity and Access Management	191
Audience	191
Authenticating With Identities	191
Managing Access Using Policies	193
How Amazon Aurora Works with IAM	195
Identity-Based Policy Examples	197
IAM Database Authentication	207
Troubleshooting	223
Logging and Monitoring	225
Compliance Validation	227
Resilience	228
Backup and Restore	228
Replication	228
Failover	228
Infrastructure Security	230

Security Groups	230
Public Accessibility	230
Security Best Practices	230
Controlling Access with Security Groups	231
VPC Security Groups	231
Security Group Scenario	232
Creating a VPC Security Group	232
Associating with a DB Instance	232
Associating with a DB Cluster	233
Master User Account Privileges	234
Service-Linked Roles	235
Service-Linked Role Permissions for Amazon Aurora	235
Creating a Service-Linked Role for Amazon Aurora	237
Editing a Service-Linked Role for Amazon Aurora	237
Deleting a Service-Linked Role for Amazon Aurora	237
Using Amazon Aurora with Amazon VPC	239
Creating a VPC for Aurora	239
Scenarios for Accessing a DB Instance in a VPC	245
Working with a DB Instance in a VPC	249
Tutorial: Create an Amazon VPC for Use with a DB Instance	255
Managing an Aurora DB Cluster	260
Stopping and Starting a Cluster	261
Overview of Stopping and Starting a Cluster	261
Limitations	261
Stopping a DB Cluster	261
While a DB Cluster Is Stopped	262
Starting a DB Cluster	263
Modifying an Aurora DB Cluster	264
Modifying the DB Cluster by Using the Console, CLI, and API	264
Modify a DB Instance in a DB Cluster	265
Available Settings	267
Non-Applicable Settings	278
Adding Aurora Replicas	280
Managing Performance and Scaling	284
Storage Scaling	284
Instance Scaling	284
Read Scaling	284
Managing Connections	285
Managing Query Execution Plans	285
Working with Parameter Groups	286
DB Cluster and DB Instance Parameters	288
Creating a DB Parameter Group	289
Creating a DB Cluster Parameter Group	290
Modifying Parameters in a DB Parameter Group	291
Modifying Parameters in a DB Cluster Parameter Group	295
Copying a DB Parameter Group	297
Copying a DB Cluster Parameter Group	299
Listing DB Parameter Groups	300
Listing DB Cluster Parameter Groups	301
Viewing Parameter Values for a DB Parameter Group	302
Viewing Parameter Values for a DB Cluster Parameter Group	303
Comparing Parameter Groups	304
DB Parameter Values	305
Managing Connections with Amazon RDS Proxy (Preview)	308
RDS Proxy Concepts and Terminology	308
Planning for and Setting Up RDS Proxy	311
Connecting through RDS Proxy	323

Managing RDS Proxy	324
Monitoring	327
Limitations	329
Examples	330
Troubleshooting	331
Cloning Databases in an Aurora DB Cluster	335
Limitations	335
Copy-on-Write Protocol for Database Cloning	335
Deleting Source Databases	337
Cloning an Aurora Cluster Through the AWS Management Console	337
Cloning an Aurora Cluster Through the AWS CLI	338
Cross-Account Cloning	339
Integrating with AWS Services	350
Aurora MySQL	350
Aurora PostgreSQL	350
Using Auto Scaling with Aurora Replicas	351
Using Machine Learning with Aurora	366
Backing Up and Restoring an Aurora DB Cluster	379
Overview of Backing Up and Restoring	380
Backup Storage	382
Creating a DB Cluster Snapshot	383
Restoring from a DB Cluster Snapshot	385
Copying a Snapshot	388
Sharing a Snapshot	399
Point-in-Time Recovery	405
Deleting a Snapshot	407
Maintaining an Aurora DB Cluster	408
Applying Updates	410
The Maintenance Window	411
Adjusting the Maintenance Window for a DB Cluster	412
Choosing the Frequency of Aurora MySQL Maintenance Updates	414
Rebooting a DB Instance	416
Deleting a DB Instance	418
Deletion Protection	418
Aurora Clusters with a Single DB Instance	418
Using the Console, CLI, and API	418
Tagging RDS Resources	420
Overview	420
Working with ARNs	425
Constructing an ARN	425
Getting an Existing ARN	428
Aurora Updates	432
Identifying Your Amazon Aurora Version	432
Monitoring an Aurora DB Cluster	433
Overview of Monitoring	434
Monitoring Tools	435
Monitoring with CloudWatch	436
Publishing to CloudWatch Logs	442
Viewing a DB Cluster	446
DB Cluster Status	452
DB Instance Status	454
Monitoring Aurora DB Cluster Metrics	457
Aurora MySQL Metrics	457
Viewing Metrics in the Amazon RDS Console	463
Aurora Metrics Available in the Amazon RDS Console	465
Related Topics	467
Enhanced Monitoring	469

Differences Between CloudWatch and Enhanced Monitoring Metrics	469
Setting Up for and Enabling Enhanced Monitoring	469
Viewing Enhanced Monitoring	471
Viewing Enhanced Monitoring by Using CloudWatch Logs	473
Performance Insights	476
Enabling Performance Insights	477
Access Control for Performance Insights	481
Using the Performance Insights Dashboard	482
Additional User Interface Features	495
Performance Insights API	496
Metrics Published to CloudWatch	509
Performance Insights Counters	510
Logging Performance Insights Calls by Using AWS CloudTrail	517
Using Amazon Aurora Recommendations	520
Responding to Recommendations	521
Using Database Activity Streams	525
Starting an Activity Stream	525
Getting Activity Stream Status	527
Stopping an Activity Stream	528
Monitoring Activity Streams	528
Managing Access to Activity Streams	540
Using Amazon RDS Event Notification	543
Amazon RDS Event Categories and Event Messages	544
Subscribing to Amazon RDS Event Notification	550
Listing Your Amazon RDS Event Notification Subscriptions	553
Modifying an Amazon RDS Event Notification Subscription	555
Adding a Source Identifier to an Amazon RDS Event Notification Subscription	557
Removing a Source Identifier from an Amazon RDS Event Notification Subscription	558
Listing the Amazon RDS Event Notification Categories	559
Deleting an Amazon RDS Event Notification Subscription	560
Viewing Amazon RDS Events	561
Console	561
AWS CLI	561
API	561
.....	562
Database Log Files	563
Viewing and Listing Database Log Files	563
Downloading a Database Log File	563
Watching a Database Log File	564
Publishing to CloudWatch Logs	565
Reading Log File Contents Using REST	565
MySQL Database Log Files	567
PostgreSQL Database Log Files	572
Logging Amazon RDS API Calls with AWS CloudTrail	574
Amazon RDS Information in CloudTrail	574
Understanding Amazon RDS Log File Entries	574
Working with Aurora MySQL	578
Overview of Aurora MySQL	578
Amazon Aurora MySQL Performance Enhancements	578
Aurora MySQL and Spatial Data	579
Comparison of Aurora MySQL 5.6 and Aurora MySQL 5.7	580
Comparison of Aurora MySQL 5.7 and MySQL 5.7	580
Security with Aurora MySQL	581
Master User Privileges with Aurora MySQL	581
Using SSL with Aurora MySQL DB Clusters	582
Updating Applications for New SSL/TLS Certificates	583

Determining Whether any Applications are Connecting to Your Aurora MySQL DB Cluster Using SSL	584
Determining Whether a Client Requires Certificate Verification to Connect	585
Updating Your Application Trust Store	585
Example Java Code for Establishing SSL Connections	587
Migrating Data to Aurora MySQL	587
Migrating from an External MySQL Database to Aurora MySQL	589
Migrating from a MySQL DB Instance to Aurora MySQL	607
Managing Aurora MySQL	623
Managing Performance and Scaling for Amazon Aurora MySQL	623
Backtracking a DB Cluster	625
Testing Amazon Aurora Using Fault Injection Queries	639
Altering Tables in Amazon Aurora Using Fast DDL	642
Displaying Volume Status for an Aurora DB Cluster	643
Parallel Query for Aurora MySQL	644
Overview of Parallel Query	645
Administration	647
Upgrading Parallel Query	647
Creating a Parallel Query Cluster	647
Enabling and Disabling Parallel Query	651
Performance Tuning	653
Creating Schema Objects	653
Verifying Parallel Query Usage	653
Monitoring	656
Parallel Query and SQL Constructs	658
Advanced Auditing with Aurora MySQL	669
Enabling Advanced Auditing	669
Viewing Audit Logs	671
Audit Log Details	671
Single-Master Replication with Aurora MySQL	672
Aurora Replicas	672
Options	673
Performance	673
High Availability	674
Monitoring	674
Replicating Amazon Aurora MySQL DB Clusters Across AWS Regions	674
Replication Between Aurora and MySQL or Between Aurora and Another Aurora DB Cluster	684
Using GTID-Based Replication	698
Working with Multi-Master Clusters	702
Overview of Multi-Master Clusters	702
Creating a Multi-Master Cluster	707
Managing Multi-Master Clusters	712
Application Considerations	715
Performance Considerations	724
Approaches to Multi-Master Clusters	726
Integrating Aurora MySQL with AWS Services	727
Authorizing Aurora MySQL to Access AWS Services	728
Loading Data From Text Files in Amazon S3	739
Saving Data into Text Files in Amazon S3	747
Invoking a Lambda Function from Aurora MySQL	752
Publishing Aurora MySQL Logs to CloudWatch Logs	759
Aurora MySQL Lab Mode	762
Aurora Lab Mode Features	762
Best Practices with Amazon Aurora MySQL	763
Determining Which DB Instance You Are Connected To	763
Using T2 Instances	763
Invoking an AWS Lambda Function	764

Working with Asynchronous Key Prefetch	765
Working with Multi-Threaded Replication Slaves in Amazon Aurora MySQL	767
Using Amazon Aurora to Scale Reads for Your MySQL Database	767
Using Amazon Aurora for Disaster Recovery with Your MySQL Databases	770
Migrating from MySQL to Amazon Aurora MySQL with Reduced Downtime	770
Using XA Transactions with Amazon Aurora MySQL	770
Working with Hash Joins	771
Working with Foreign Keys	772
Related Topics	773
Aurora MySQL Reference	773
Parameters	773
Inapplicable MySQL Parameters and Status Variables	785
Aurora MySQL Events	785
Aurora MySQL Isolation Levels	787
Stored Procedures	791
Aurora MySQL Updates	794
Aurora MySQL Versions	794
Aurora MySQL Engine Versions	794
Database Upgrades and Patches for Amazon Aurora MySQL	795
Zero-Downtime Patching	796
Aurora MySQL Long-Term Support (LTS) Releases	797
Related Topics	798
Database Engine Updates for Amazon Aurora MySQL 2.0	798
Database Engine Updates for Amazon Aurora MySQL 1.1	830
MySQL Bugs Fixed by Aurora MySQL Updates	868
Working with Aurora PostgreSQL	879
Security with Aurora PostgreSQL	879
Restricting Password Management	880
Securing Aurora PostgreSQL Data with SSL	881
Updating Applications for New SSL/TLS Certificates	883
Determining Whether Applications Are Connecting to Aurora PostgreSQL DB Clusters Using SSL	884
Determining Whether a Client Requires Certificate Verification in Order to Connect	884
Updating Your Application Trust Store	885
Using SSL/TLS Connections for Different Types of Applications	886
Migrating Data to Aurora PostgreSQL	887
Migrating an RDS PostgreSQL DB Snapshot	887
Migrating an RDS PostgreSQL DB Instance Using an Aurora Read Replica	890
Importing S3 Data into Aurora PostgreSQL	899
Managing Aurora PostgreSQL	910
Scaling Aurora PostgreSQL DB Instances	910
Maximum Connections	910
Replication with Aurora PostgreSQL	911
Aurora Replicas	911
Monitoring Replication	912
Using Logical Replication	912
Integrating Aurora PostgreSQL with AWS Services	916
Managing Query Execution Plans for Aurora PostgreSQL	916
Enabling Query Plan Management	917
Upgrading Query Plan Management	918
Basics	918
Best Practices for Query Plan Management	921
Examining plans in the dba_plans view	922
Capturing Execution Plans	925
Using Managed Plans	927
Maintaining Execution Plans	930
Parameter Reference for Query Plan Management	934

Function Reference for Query Plan Management	938
Publishing Aurora PostgreSQL Logs to CloudWatch Logs	943
Monitoring Log Events in Amazon CloudWatch	946
Fast Recovery after Failover	946
Configuring Cluster Cache Management	947
Monitoring the Buffer Cache	949
Upgrading the Engine Version	950
Manually Upgrading the Minor Engine Version	951
Automatically Upgrading the Minor Engine Version	952
Upgrading PostgreSQL Extensions	953
Best Practices with Aurora PostgreSQL	953
Fast Failover	953
Troubleshooting storage issues	960
Aurora PostgreSQL Reference	960
Parameters	960
Amazon Aurora PostgreSQL Events	969
Aurora PostgreSQL Updates	970
Identifying Your Version	970
Upgrading	971
Engine Versions for Aurora PostgreSQL	971
Best Practices with Aurora	987
Amazon Aurora Basic Operational Guidelines	987
DB Instance RAM Recommendations	987
Monitoring Amazon Aurora	988
Working with DB Parameter Groups and DB Cluster Parameter Groups	988
Amazon Aurora Best Practices Presentation Video	988
Performing an Aurora Proof of Concept	989
Overview of an Aurora Proof of Concept	989
1. Identify Your Objectives	989
2. Understand Your Workload Characteristics	990
3. Practice with the Console or CLI	991
Practice with the Console	991
Practice with the AWS CLI	991
4. Create Your Aurora Cluster	992
5. Set Up Your Schema	993
6. Import Your Data	993
7. Port Your SQL Code	994
8. Specify Configuration Settings	994
9. Connect to Aurora	995
10. Run Your Workload	996
11. Measure Performance	996
12. Exercise Aurora High Availability	998
13. What to Do Next	999
Limits	1001
Limits in Amazon Aurora	1001
Naming Constraints in Amazon Aurora	1002
Amazon Aurora File Size Limits	1003
Troubleshooting	1004
Cannot Connect to DB Instance	1004
Testing the DB Instance Connection	1004
Troubleshooting Connection Authentication	1005
Security Issues	1005
Error Message "Failed to retrieve account attributes, certain console functions may be impaired."	1005
Resetting the DB Instance Owner Role Password	1005
DB Instance Outage or Reboot	1006
Parameter Changes Not Taking Effect	1006

Aurora MySQL Out of Memory Issues	1006
Aurora MySQL Replication Issues	1007
Diagnosing and Resolving Lag Between Read Replicas	1007
Diagnosing and Resolving a MySQL or MariaDB Read Replication Failure	1008
Slave Down or Disabled Error	1009
No Space Left on Device Error	1010
Amazon RDS API Reference	1011
Using the Query API	1011
Query Parameters	1011
Query Request Authentication	1011
Troubleshooting Applications	1012
Retrieving Errors	1012
Troubleshooting Tips	1012
Document History	1013
AWS Glossary	1028

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, up to 64 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 a 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 \(p. 49\)](#), 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 Connection Management \(p. 4\)](#)
- [Using the Instance Endpoints \(p. 23\)](#)
- [How Aurora Endpoints Work with High Availability \(p. 23\)](#)
- [Amazon Aurora Storage and Reliability \(p. 24\)](#)
- [Amazon Aurora Security \(p. 26\)](#)
- [High Availability for Aurora \(p. 27\)](#)
- [Working with Amazon Aurora Global Database \(p. 28\)](#)

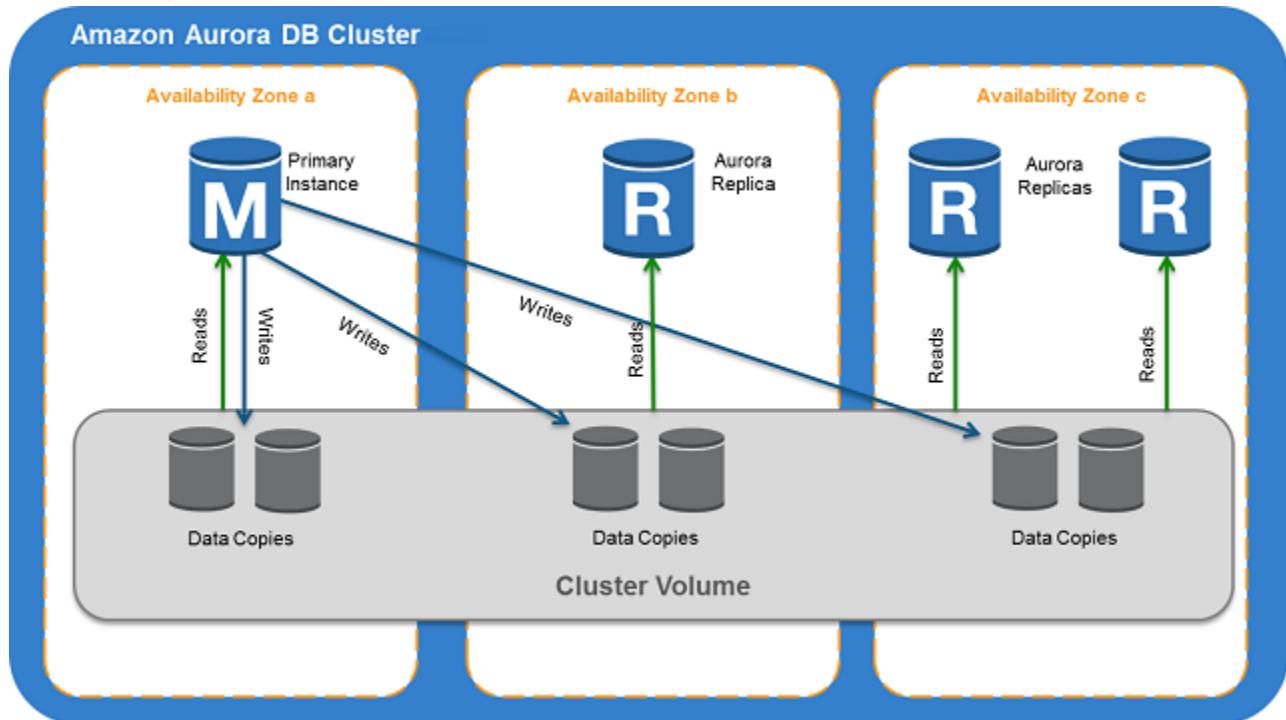
- [Replication with Amazon Aurora \(p. 47\)](#)

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.
- For Aurora multi-master clusters, all DB instances have read-write capability. In this case, the distinction between primary instance and Aurora Replica doesn't apply. For discussing replication topology where the clusters can use either single-master or multi-master replication, we call these *writer* and *reader* DB instances.

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 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 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 \(p. 4\)](#)
- [Viewing the Endpoints for an Aurora Cluster \(p. 6\)](#)
- [Using the Cluster Endpoint \(p. 6\)](#)
- [Using the Reader Endpoint \(p. 7\)](#)
- [Using Custom Endpoints \(p. 7\)](#)
- [Creating a Custom Endpoint \(p. 9\)](#)
- [Viewing Custom Endpoints \(p. 11\)](#)
- [Editing a Custom Endpoint \(p. 16\)](#)
- [Deleting a Custom Endpoint \(p. 18\)](#)
- [End-to-End AWS CLI Example for Custom Endpoints \(p. 19\)](#)

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 connects to one of the available Aurora Replicas for that DB cluster. Each Aurora DB cluster has one reader endpoint. If there is more than one Aurora Replica, the reader endpoint directs each connection request to one of the Aurora Replicas.

The reader endpoint provides load-balancing support for read-only connections to the DB cluster. Use the reader endpoint for read operations, such as queries. You can't use the reader endpoint for write operations.

The DB cluster distributes connection requests to the reader endpoint among the available Aurora Replicas. If the DB cluster contains only a primary DB instance, the reader endpoint serves connection requests from the primary DB instance. If one or more Aurora Replicas are created for that DB cluster, subsequent connections to the reader endpoint are load-balanced among the Replicas.

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 \(p. 953\)](#). 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 endpoints in the output of the `describe-db-clusters` command.

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 only 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.

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 \(p. 7\)](#)
- [Membership Rules for Custom Endpoints \(p. 8\)](#)
- [Managing Custom Endpoints \(p. 8\)](#)

Specifying Properties for Custom Endpoints

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

`endpointName.cluster-custom-customerDnsIdentifier.dnsSuffix`

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

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

- Only DB instances that are read-only Aurora Replicas can be part of a READER custom endpoint. The READER type applies only to clusters using single-master replication, because those clusters can include multiple read-only DB instances.

- Both read-only Aurora Replicas and the read-write primary instance can be part of an **ANY** custom endpoint. Aurora directs connections to cluster endpoints with type **ANY** to any associated DB instance with equal probability. Because you can't determine in advance if you are connecting to the primary instance of a read-only Aurora Replica, use this kind of endpoint for read-only connections only. The **ANY** type applies to clusters using any replication topology.
- The **WRITER** type applies only to multi-master clusters, because those clusters can include multiple read-write DB instances.
- If you try to create a custom endpoint with a type that isn't appropriate based on the replication configuration for a cluster, Aurora returns an error.

Membership Rules for Custom Endpoints

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

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

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

Aurora doesn't change the DB instances specified in these 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. However, you can only connect to a DB instance through a custom endpoint when that DB instance has a role compatible with the type of the custom endpoint (**READER**, **WRITER**, or **ANY**).

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](#)

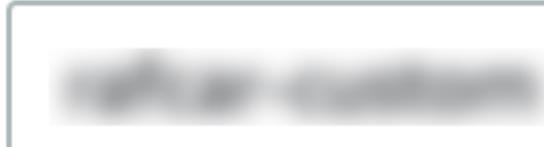
Creating a Custom Endpoint

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, OS X, 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

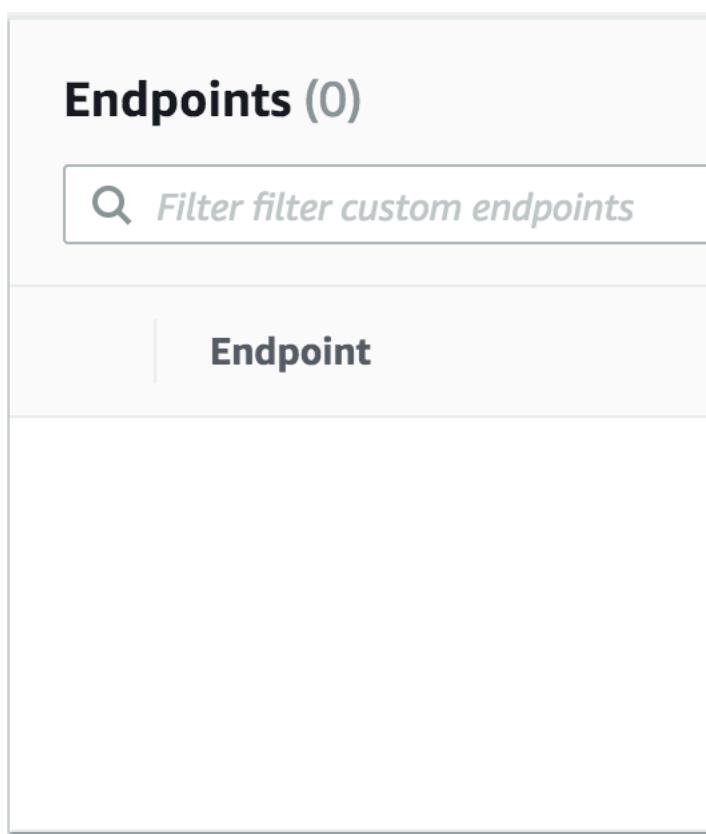
To create a custom endpoint with the RDS API, run the [CreateDBClusterEndpoint](#) operation.

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



14

Filter endpoint members

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

AWS CLI

To view custom endpoints with the AWS CLI, run the `describe-db-cluster-endpoints` command.

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

For Linux, OS X, 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 \(p. 8\)](#).

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

Console

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

RDS > Clusters: [REDACTED]

Edit endpoint:

Endpoint members



Filter database



DB instance name



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, OS X, 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
```

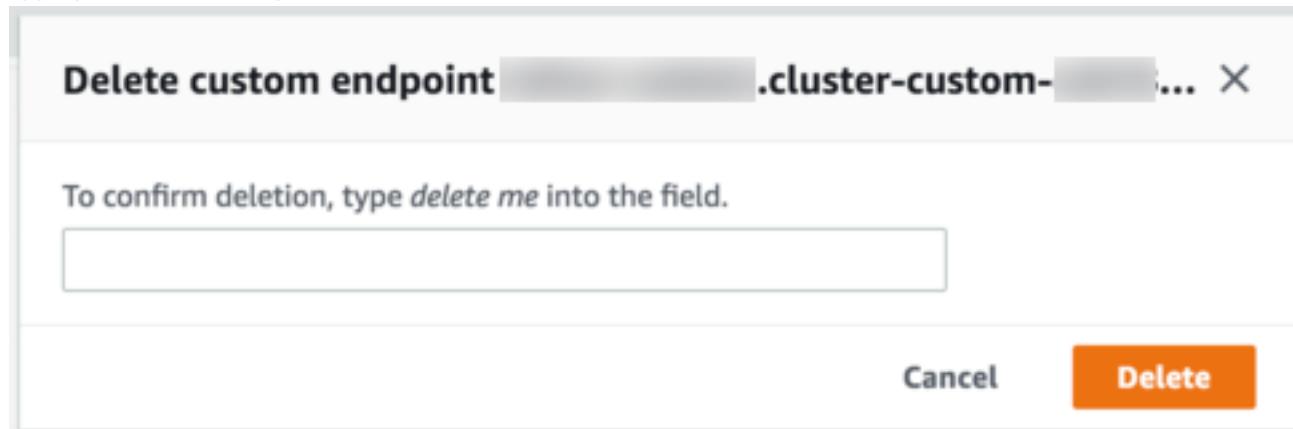
RDS API

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, OS X, 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-id custom-endpoint-demo-${i} \
        --region $REGION --promotion-tier 15
done
```

Suppose you want to use the two "bigger" instances only for the most resource-intensive queries. You can create a custom read-only endpoint, and then 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 would ever be promoted to the primary instance. If one of them was promoted to the primary instance, it would become 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

In day-to-day 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.

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.

How Aurora Endpoints Work with High Availability

For clusters where high availability is important, use the writer endpoint for read-write connections and the reader endpoint for read-only connections. These kinds of connections manage DB instance failover

better than instance endpoints do. The instance endpoints connect to a specific DB instance in a DB cluster, requiring logic in your application to choose a different endpoint if the DB instance 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 cluster endpoint to reconnect to the newly promoted or created primary DB instance, or use the reader endpoint to reconnect to one of the Aurora Replicas in the DB cluster. During a failover, the reader endpoint might direct connections to the new primary DB instance of a DB cluster for a short time after an Aurora Replica is promoted to the new primary DB instance.

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

For more information about failovers, see [Fault Tolerance for an Aurora DB Cluster \(p. 380\)](#).

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 \(p. 24\)](#)
- [What the Cluster Volume Contains \(p. 24\)](#)
- [How Aurora Storage Grows \(p. 25\)](#)
- [How Aurora Data Storage is Billed \(p. 25\)](#)
- [Amazon Aurora Reliability \(p. 25\)](#)

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 multiple Availability Zones in a single AWS Region. Because the data is automatically replicated across Availability Zones, your data is highly durable with less possibility of data loss. This replication also ensures that your database is more available during a failover. It does so because the data copies already exist in the other Availability Zones and continue to serve data requests to the DB instances in your DB cluster. The amount of replication is independent of the number of DB instances in your cluster.

What the Cluster Volume Contains

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

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

How Aurora Storage Grows

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 64 tebibytes (TiB). Table size is limited to the size of the cluster volume. That is, the maximum table size for a table in an Aurora DB cluster is 64 TiB.

This automatic storage scaling, combined with the high-performance and highly distributed storage subsystem, makes Aurora a good choice for your important enterprise data when your main objectives are reliability and high availability. See the following sections for ways to balance storage costs against these other priorities.

How Aurora Data Storage is Billed

Even though an Aurora cluster volume can grow to up to 64 TiB, you are only charged for the space that you use in an Aurora cluster volume. When Aurora data is removed, such as by dropping a table or partition, the overall allocated space remains the same. The free space is reused automatically when data volume increases in the future.

Note

Because storage costs are based on the storage "high water mark" (the maximum amount that was allocated for the Aurora cluster at any point in time), you can manage costs by avoiding ETL practices that create large volumes of temporary information, or that load large volumes of new data prior to removing unneeded older data.

If removing data from an Aurora cluster results in a substantial amount of allocated but unused space, resetting the high water mark requires doing a logical data dump and restore to a new cluster, using a tool such as `mysqldump`. Creating and restoring a snapshot does *not* reduce the allocated storage because the physical layout of the underlying storage remains the same in the restored snapshot.

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 \(p. 25\)](#)
- [Survivable Cache Warming \(p. 25\)](#)
- [Crash Recovery \(p. 26\)](#)

Storage Auto-Repair

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

Survivable Cache Warming

Aurora "warms" the buffer pool cache when a database starts up after it has been shut down or restarted after a failure. That is, Aurora preloads the buffer pool with the pages for known common queries that

are stored in an in-memory page cache. This provides a performance gain by bypassing the need for the buffer pool to "warm up" from normal database use.

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

Crash Recovery

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

For more information about crash recovery, see [Fault Tolerance for an Aurora DB Cluster \(p. 380\)](#).

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 \(p. 47\)](#). 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 IAM account must have IAM policies that grant the permissions required to perform Amazon RDS management operations. For more information, see [Identity and Access Management in Amazon Aurora \(p. 191\)](#).

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

- Aurora DB clusters must be created in an Amazon Virtual Private Cloud (VPC). To control which devices and Amazon EC2 instances can open connections to the endpoint and port of the DB instance for Aurora DB clusters in a VPC, you use a VPC security group. You can make these endpoint and port connections using Transport Layer Security (TLS) / Secure Sockets Layer (SSL). In addition, firewall rules at your company can control whether devices running at your company can open connections to a DB instance. For more information on VPCs, see [Amazon Virtual Private Cloud VPCs and Amazon Aurora \(p. 239\)](#).
- 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 \(p. 581\)](#) or [Security with Amazon Aurora PostgreSQL \(p. 879\)](#).

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

With IAM database authentication, you authenticate to your Aurora MySQL DB cluster by using an IAM user or IAM role and an authentication token. An *authentication token* is a unique value that is generated using the Signature Version 4 signing process. By using IAM database authentication, you can use the same credentials to control access to your AWS resources and your databases. For more information, see [IAM Database Authentication \(p. 207\)](#).

For information about configuring security, see [Security in Amazon Aurora \(p. 173\)](#).

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 \(p. 581\)](#), [Security with Amazon Aurora PostgreSQL \(p. 879\)](#), or [Using TLS/SSL with Aurora Serverless \(p. 118\)](#).

High Availability for Aurora

The 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 ensure that one or more DB instances are ready to handle database requests from your application.

Aurora stores copies of the data in a DB cluster across multiple Availability Zones in a single AWS Region, 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 \(p. 260\)](#). 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 [Choosing the Regions and Availability Zones \(p. 80\)](#).

For a cluster using single-master replication, after you create the primary instance, you can create up to 15 Aurora Replicas. These read-only DB instances support `SELECT` queries for read-intensive applications. We recommend that you distribute the primary instance and Aurora Replicas in your DB cluster over multiple Availability Zones to improve the availability of your DB cluster. When you create

reader instances across Availability Zones, Amazon RDS automatically provisions them and keeps them up-to-date with the primary instance.

Using the RDS console, you can create a Multi-AZ deployment by simply specifying Multi-AZ when creating a DB cluster. If a DB cluster is in a single Availability Zone, you can make it a Multi-AZ DB cluster adding another DB instance in a different Availability Zone.

You can specify a Multi-AZ deployment using the CLI as well. Use the AWS CLI [describe-db-instances](#) command, or the Amazon RDS API [DescribeDBInstances](#) operation to show the Availability Zone of the standby replica (called the secondary AZ).

For more information, see [Region Availability \(p. 80\)](#). Call the [create-db-instance](#) AWS CLI command to create an Aurora Replica in your DB cluster. Include the name of the DB cluster as the --db-cluster-identifier parameter value. You can optionally specify an Availability Zone for the Aurora Replica using the --availability-zone parameter.

For more information about failover to Aurora Replicas, see [Amazon Aurora Connection Management \(p. 4\)](#). For more information about creating a DB cluster, see [Creating an Amazon Aurora DB Cluster \(p. 100\)](#).

Working with Amazon Aurora Global Database

Following, you can find a description of Amazon Aurora Global Database. Each Aurora global database spans multiple AWS Regions, enabling low latency global reads and disaster recovery from region-wide outages.

Topics

- [Overview of Aurora Global Database \(p. 28\)](#)
- [Creating an Aurora Global Database \(p. 30\)](#)
- [Adding an AWS Region to an Aurora Global Database \(p. 35\)](#)
- [Removing a Cluster from an Aurora Global Database \(p. 37\)](#)
- [Deleting an Aurora Global Database \(p. 39\)](#)
- [Importing Data into an Aurora Global Database \(p. 41\)](#)
- [Managing an Aurora Global Database \(p. 42\)](#)
- [Configuring an Aurora Global Database \(p. 42\)](#)
- [Connecting to an Aurora Global Database \(p. 44\)](#)
- [Failover for Aurora Global Database \(p. 44\)](#)
- [Performance Insights for Aurora Global Database \(p. 45\)](#)
- [Using Multiple Secondary Regions with Aurora Global Database \(p. 45\)](#)
- [Using Aurora Global Database with Other AWS Services \(p. 45\)](#)

Overview of Aurora Global Database

An Aurora global database consists of one primary AWS Region where your data is mastered, and up to five read-only, secondary AWS Region. Aurora replicates data to the secondary AWS Region with typical latency of under a second. You issue write operations directly to the primary DB instance in the primary AWS Region. An Aurora global database uses dedicated infrastructure to replicate your data, leaving database resources available entirely to serve application workloads. Applications with a worldwide footprint can use reader instances in the secondary AWS Region for low latency reads. In the unlikely event your database becomes degraded or isolated in an AWS region, you can promote the secondary AWS Region to take full read-write workloads in under a minute.

The Aurora cluster in the primary AWS Region where your data is mastered performs both read and write operations. The cluster in the secondary region enables low-latency reads. You can scale up the secondary clusters independently by adding one or more DB instances (Aurora Replicas) to serve read-only workloads. For disaster recovery, you can remove and promote one of the secondary clusters to allow full read and write operations.

Only the primary cluster performs write operations. Clients that perform write operations connect to the DB cluster endpoint of the primary cluster.

Topics

- [Advantages of Aurora Global Database \(p. 29\)](#)
- [Requirements for Aurora Global Databases \(p. 29\)](#)

Advantages of Aurora Global Database

Aurora Global Database uses dedicated infrastructure to replicate changes between the primary DB cluster and any secondary clusters. Aurora Global Database provides the following advantages:

- The replication performed by an Aurora global database has limited to no performance impact on the primary DB cluster. The resources of the DB instances are fully devoted to serve read and write workloads.
- Changes are replicated between AWS Regions with minimal lag, typically less than 1 second.
- The secondary clusters enable fast failover for disaster recovery. You can typically promote a secondary cluster and make it available for writes in under a minute.
- Applications in remote AWS Regions experience lower query latency when they read from a secondary cluster.
- You can add up to 16 Aurora Replicas to any of the secondary clusters, allowing you to scale reads beyond the capacity of a single Aurora cluster.

Requirements for Aurora Global Databases

The following limitations currently apply to Aurora Global Database:

- You can't use `db.t2` instance classes for an Aurora global database. You have a choice of `db.r4` or `db.r5` instance classes.
- Currently, Aurora Global Database isn't available in the Europe (Stockholm) and Asia Pacific (Hong Kong) regions.
- The primary cluster and each of the secondary clusters must be in a separate AWS Region. You can't put the primary cluster in the same AWS Region as one of the secondary clusters. You can't put two secondary clusters in the same AWS Region.
- You can't create a cross-region Read Replica from the primary cluster in same region as the secondary. See [Replicating Amazon Aurora MySQL DB Clusters Across AWS Regions \(p. 674\)](#) for information about cross-region Read Replicas.
- The following features aren't supported for Aurora Global Database:
 - Cloning. For information about cloning, see [Cloning Databases in an Aurora DB Cluster \(p. 335\)](#).
 - Backtrack. For information about backtracking, see [Backtracking an Aurora DB Cluster \(p. 625\)](#).
 - Parallel query. For information about parallel query, see [Working with Parallel Query for Amazon Aurora MySQL \(p. 644\)](#).
- Aurora Serverless. For information about Aurora Serverless, see [Using Amazon Aurora Serverless \(p. 116\)](#).
- Stopping and starting the DB clusters within the global database. See [Stopping and Starting an Amazon Aurora DB Cluster \(p. 261\)](#) for information about stopping and starting Aurora clusters.

Creating an Aurora Global Database

An Aurora global database spans multiple AWS Regions. You create the global database itself along with the read-write primary cluster. Then you create one or more read-only secondary clusters in other AWS Regions.

- To create a new global database, you create the global database and the primary cluster that it contains, and then add one or more secondary clusters.
- If you have an existing Aurora cluster using Aurora MySQL 1.21 or lower, you can take a snapshot and restore it to a new Aurora global database. To do so, follow the procedure in [Importing Data into an Aurora Global Database \(p. 41\)](#). For clusters running Aurora MySQL 1.22 or higher, or 2.07 or higher, you can add them to a global database without any special procedures or cluster creation options.

When specifying names for the primary and secondary clusters, choose names that are different than your existing Aurora clusters, even if those clusters are in other AWS Regions.

Important

Before you create an Aurora global database, complete the setup tasks for your account, network, and security settings in [Setting Up for Amazon RDS](#) in the *Amazon Relational Database Service User Guide*.

Note

If you create an Aurora cluster through the AWS CLI or RDS API and intend to add it later to an Aurora global database, you must use the `global` value for the engine mode parameter.

Console

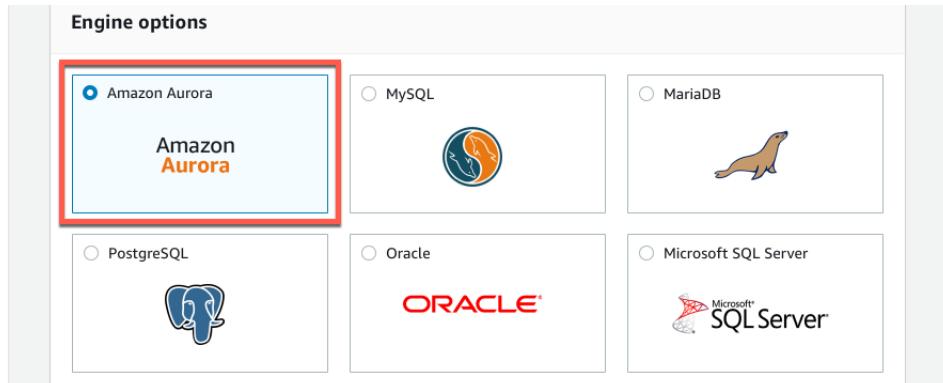
To create an Aurora global database using the AWS Management Console

1. Sign in to the AWS Management Console and open the Amazon RDS console for a region where Aurora global databases are available.
2. Choose **Create Database**.
3. On the **Select engine** page, choose the MySQL 5.6-compatible or 5.7-compatible Aurora engine and **Global for Database location**. For an example, see the following images.

Note

Make sure that **Standard Create** is chosen. **Standard create** makes visible the choices that you need for Aurora global databases. Don't choose **Easy Create**.

- a. Choose Aurora as the engine:



- b. Choose MySQL 5.6 or MySQL 5.7 compatibility.

- c. Choose **Global** as the location:

Database Location

Regional

You provision your Aurora database in a single region.

Global

You can provision your Aurora database in multiple regions. Writes in the primary region are replicated with typical latency of <1 sec to secondary regions.

Note

Choosing **Global** sets up the global database and the primary Aurora cluster. Once the global database is created and available, you can add the secondary AWS Region.

4. Fill in the remaining settings using the same decision process as for other Aurora clusters. For more information about creating an Aurora DB cluster, see [Creating an Amazon Aurora DB Cluster \(p. 100\)](#).

Global database settings

Settings

Global database identifier [Info](#)

Type a name for your global database. The name must be unique across all global databases in your AWS account.

global-db-identifier

The global database identifier is case-insensitive, but is stored as all lowercase (as in "myglobaldb").
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.

Master username [Info](#)

Type a login ID for the master user of your global database.

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

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](#)

IAM DB authentication [Info](#)

Enable IAM DB authentication

Manage your database user credentials through AWS IAM users and roles.

5. Choose **Create**.

When the primary DB cluster is created and available, create one or more secondary clusters by following the steps in [Adding an AWS Region to an Aurora Global Database \(p. 35\)](#).

AWS CLI

To create an Aurora global database using the CLI

1. Create an Aurora global database and then add a primary AWS Region and a DB instance in that AWS Region.

- a. Create the Aurora global database itself. Initially, no clusters are associated with it.

For Linux, OS X, or Unix:

```
aws rds create-global-cluster \
--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
```

To learn more about additional options, see [create-global-cluster](#).

- b. Check that the Aurora global database has become available to use by using the AWS CLI [describe-global-clusters](#) command, as shown in the following example.

```
aws rds --region primary_region \
describe-global-clusters \
--global-cluster-identifier global_database_id # optional
```

- c. When the [describe-global-clusters](#) results show a status of available, create the primary cluster for the Aurora global database. To do so, use the AWS CLI [create-db-cluster](#) command. Specify `aurora` for the `--engine` parameter and `5.6.10a` for the `--engine-version` parameter. Specify the same `--global-cluster-identifier` value as when you created the global database. The following example shows how.

For Linux, OS X, or Unix:

```
aws rds create-db-cluster \
--region primary_region \
--db-cluster-identifier db_cluster_id \
--master-username master_userid \
--master-user-password master_password \
--engine { aurora | aurora-mysql } \
--engine-version version \
--global-cluster-identifier global_database_id \
--engine-mode global # Optional
```

For Windows:

```
aws rds create-db-cluster ^
--region primary_region ^
--db-cluster-identifier db_cluster_id ^
--master-username master_userid ^
--master-user-password master_password ^
--engine { aurora | aurora-mysql } ^
--engine-version version ^
--global-cluster-identifier global_database_id ^
```

```
--engine-mode global # Optional
```

To learn more about additional options, see [create-db-cluster](#).

- d. Check that the Aurora cluster has become available to use by using the AWS CLI [describe-db-clusters](#) command, as shown in the following example.

```
aws rds --region primary_region \  
  describe-db-clusters \  
    --db-cluster-identifier db_cluster_id # optional
```

- e. When the [describe-db-clusters](#) results show a status of `available`, create the primary DB instance for the primary cluster. To do so, use the AWS CLI [create-db-instance](#) command.
- Specify `aurora` for the `--engine` parameter for MySQL 5.6 compatibility. Or specify `aurora-mysql` for the `--engine` parameter for MySQL 5.6 compatibility.
 - Optionally, specify the `--engine-version` parameter to choose a specific Aurora MySQL version to use.
 - Depending on your Aurora MySQL version, you might or might not need to specify the `--engine-mode` option.
 - For Aurora MySQL version 1, in 1.21 and lower, specify `--engine-mode global`. For 1.22 and higher, `--engine-mode` is optional.
 - For Aurora MySQL version 2, global database support isn't available in Aurora MySQL 2.06 and lower. For 2.07 and higher, the `--engine-mode` option is optional.
 - The Aurora MySQL versions 1.22 and higher and 2.07 and higher all include global database support for clusters you create with the default engine mode, `provisioned`. You can add such clusters to a global database without any special choices during cluster creation.

The following example shows how.

For Linux, OS X, 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-version version \  
  --engine-mode global # only needed if version 1.x is less than or equal to 1.21
```

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-version version ^ # Optional  
  --engine-mode global # only needed if version 2.x is less than or equal to 2.06
```

You don't need to include the `--master-username` and `--master-user-password` parameters, because those values are taken from the primary DB cluster.

To learn more about additional options, see [create-db-instance](#).

2. Check that the primary cluster of the Aurora global database 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-global-cluster
```

When the [describe-db-clusters](#) results show a status of `available`, create the primary instance for the primary 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, OS X, or Unix:

```
aws rds create-db-instance \
--db-cluster-identifier sample-global-cluster \
--db-instance-class instance_class \
--db-instance-identifier sample-replica-instance \
--engine aurora
```

For Windows:

```
aws rds create-db-instance ^
--db-cluster-identifier sample-global-cluster ^
--db-instance-class instance_class ^
--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.

3. Alternatively, you can create the primary cluster first, then create the global database afterwards, specifying which cluster to use as the primary cluster:
 - a. Create an empty Aurora cluster with the `global` engine mode, and a DB instance within that cluster. Engine mode `global` means that you can add this cluster to an Aurora global database.

For Linux, OS X, or Unix:

```
aws rds --region primary_region \
create-db-cluster \
--db-cluster-identifier primary_cluster_id \
--master-username master_userid \
--master-user-password master_password \
--engine aurora \
--engine-version 5.6.10a

aws rds --region primary_region \
create-db-instance \
--db-instance-class instance_class \
--db-cluster-identifier primary_cluster_id \
--db-instance-identifier db_instance_id \
--engine aurora
```

For Windows:

```
aws rds --region primary_region ^
```

```
create-db-cluster ^
--db-cluster-identifier primary_cluster_id ^
--master-username master_userid ^
--master-user-password master_password ^
--engine aurora ^
--engine-version 5.6.10a

aws rds --region primary_region ^
create-db-instance ^
--db-instance-class instance_class ^
--db-cluster-identifier primary_cluster_id ^
--db-instance-identifier db_instance_id ^
--engine aurora
```

- b. When the primary DB cluster and DB instance are created and available, create the Aurora global database by calling the AWS CLI [create-global-cluster](#) command in the AWS Region where you want to create the Aurora global database.

For Linux, OS X, or Unix:

```
aws rds create-global-cluster \
--global-cluster-identifier global_database_id \
--source-db-cluster-identifier primary_cluster_ARN
```

For Windows:

```
aws rds create-global-cluster ^
--global-cluster-identifier global_database_id ^
--source-db-cluster-identifier primary_cluster_ARN
```

RDS API

To create an Aurora global database with the RDS API, run the [CreateGlobalCluster](#) operation.

Adding an AWS Region to an Aurora Global Database

After you create an Aurora global database and its primary cluster, you increase its global footprint by adding one or more new AWS Regions. This process involves attaching one or more secondary Aurora clusters. Each secondary cluster must be in a different AWS Region than the primary cluster.

Note

Currently, the maximum number of secondary Aurora clusters you can attach to an Aurora global database is five. Thus, the most clusters you can associate with a global database is six: one read-write primary cluster, and five read-only secondary clusters.

Console

To add an AWS Region to an Aurora global database using the AWS Management Console

1. In the top-right corner of the AWS Management Console, choose any AWS Region where Aurora global databases are available.
2. In the navigation pane, choose **Databases**.
3. Choose the check box for the Aurora global database for which you want to create a secondary cluster. If the primary cluster or DB instances inside it are still in `Creating` state, wait until they are all `Available`.

4. For Actions, choose Add region.

The screenshot shows the 'Databases' section of the Amazon RDS console. A global database named 'db-global' is selected. Below it are two clusters: 'db-cluster-1' (Primary) and 'db-cluster-2' (Secondary). The 'Actions' dropdown menu is open, and the 'Add a region' option is highlighted with a red box.

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

The screenshot shows the 'Add a region' configuration page. In the 'Region' section, the 'Secondary region' dropdown is set to 'US West (Oregon)', which is highlighted with a red box. In the 'DB instance size' section, the 'db.r4.large' instance is selected, showing a price of '\$ 0.29 USD per hour'. There is also a checkbox for 'Include previous generation classes'.

6. Complete the remaining fields for the Aurora cluster in the new AWS Region, and then choose Create.

AWS CLI

To add an AWS Region to an Aurora global database with the AWS CLI, run the [create-db-cluster](#) command.

The following commands create a new Aurora cluster, attach it to a global database as a read-only secondary cluster, and then add a DB instance to the new cluster. The `--global-cluster-identifier` option for the `create-db-cluster` command specifies the global database to which to attach the new cluster.

For an encrypted cluster, specify the `--source-region` option with a value that matches the primary AWS Region.

For Linux, OS X, 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 }

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 }
```

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 }

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 }
```

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.

Removing a Cluster from an Aurora Global Database

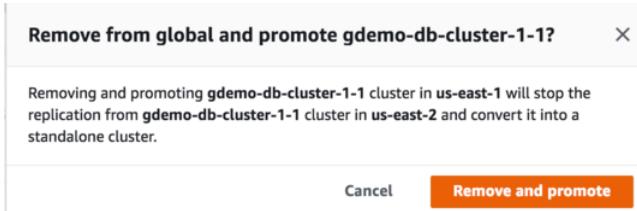
Removing an Aurora cluster from an Aurora global database turns it back into a regional cluster, with full read-write capability and no longer synchronized with the primary cluster.

You might remove an Aurora cluster from a global database in case the primary cluster becomes degraded or isolated. Performing a failover for an Aurora global database involves removing one of the secondary clusters from the original global database, and then using it as the primary cluster in a new Aurora global database.

If you no longer need the global database, you must remove all secondary clusters and then the primary cluster before deleting the global database itself. You can then delete some or all of the clusters you removed, or continue to use some or all as single-region Aurora clusters.

Console

To remove an Aurora cluster from an Aurora global database with the AWS Management Console, choose the cluster on the **Databases** page. For **Actions**, choose **Remove from Global**. The removed cluster becomes a regular Aurora cluster with full read-write capability. It is no longer kept synchronized with the primary cluster.



If any secondary clusters are still associated with the global database, you can't remove the primary cluster.

After you remove or delete all secondary clusters, then you can remove the primary cluster the same way.

After clusters are removed from an Aurora global database, you still see them in the **Databases** page, but without the **Global**, **Primary**, and **Secondary** labels.

Amazon RDS > Databases								
Databases		DB Name		Role		CPU	Activity	Info
	DB Name							
○	db-global	Global	-	-	-	Aurora MySQL	0 clusters	0 regions
○	db-cluster-1	Global compatible	-	-	-	Aurora MySQL	2 instances	us-east-2
○	instance-name	Writer	6.72%	1 Selects/Sec	-	Aurora MySQL	db.r4.xlarge	us-east-2a
○	instance-name-us-west-2b	Reader	6.72%	1 Selects/Sec	-	Aurora MySQL	db.r4.xlarge	us-east-2b
○	dbname-cluster2	Cluster	-	-	-	Aurora MySQL	2 instances	us-east-2
○	instance-name1-1	Writer	6.72%	1 Selects/Sec	-	Aurora MySQL	db.r4.xlarge	us-east-2a
○	instance-name1-2-us-west-2b	Reader	6.72%	1 Selects/Sec	-	Aurora MySQL	db.r4.xlarge	us-east-2b
○	instance-master-name4	Instance	6.72%	1 Selects/Sec	-	PostgreSQL	db.r4.xlarge	us-east-2
○	instance-name4	Replica	6.72%	1 Selects/Sec	-	PostgreSQL	db.r4.xlarge	us-east-2a
○	dbname3	Instance	6.72%	1 Selects/Sec	-	MySQL	db.r4.xlarge	us-east-2
○	dbname4	Instance	6.72%	1 Selects/Sec	-	Oracle Enterprise Edition	db.r4.xlarge	us-east-2

AWS CLI

To remove an Aurora cluster from an Aurora global database with the AWS CLI, run the [remove-from-global-cluster](#) command.

The following commands remove a secondary cluster and then the primary cluster from an Aurora global database. The clusters to remove are identified by the `--db-cluster-identifier` parameter in each case. The Aurora global database is identified by the `--global-cluster-identifier` parameter.

For Linux, OS X, or Unix:

```
aws rds --region secondary_region \
    remove-from-global-cluster \
    --db-cluster-identifier secondary_cluster_ARN \
    --global-cluster-identifier global_database_id
... repeat above command with a different --region for all secondary clusters ...

aws rds --region primary_region \
    remove-from-global-cluster \
    --db-cluster-identifier primary_cluster_ARN \
    --global-cluster-identifier global_database_id
```

For Windows:

```
aws rds --region secondary_region ^
remove-from-global-cluster ^
--db-cluster-identifier secondary_cluster_ARN ^
--global-cluster-identifier global_database_id
... repeat above command with a different --region for all secondary clusters ...

aws rds --region primary_region ^
remove-from-global-cluster ^
--db-cluster-identifier primary_cluster_ARN ^
--global-cluster-identifier global_database_id
```

RDS API

To remove an Aurora cluster from an Aurora global database with the RDS API, run the [RemoveFromGlobalCluster](#) operation.

Deleting an Aurora Global Database

Before you can delete an Aurora global database, you must delete or remove the secondary and primary cluster associated with the global database. See [Removing a Cluster from an Aurora Global Database \(p. 37\)](#) for the procedure to remove a cluster from the global database and make it a standalone Aurora cluster again.

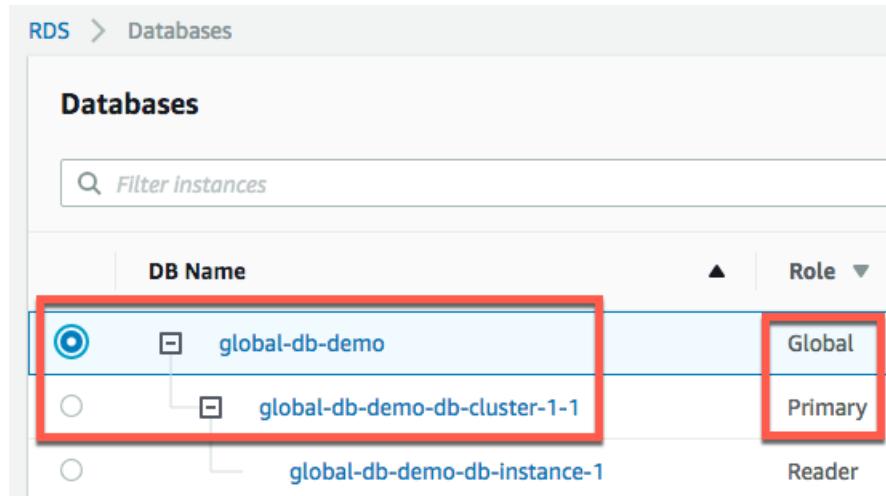
Console

To delete an Aurora global database using the AWS Management Console

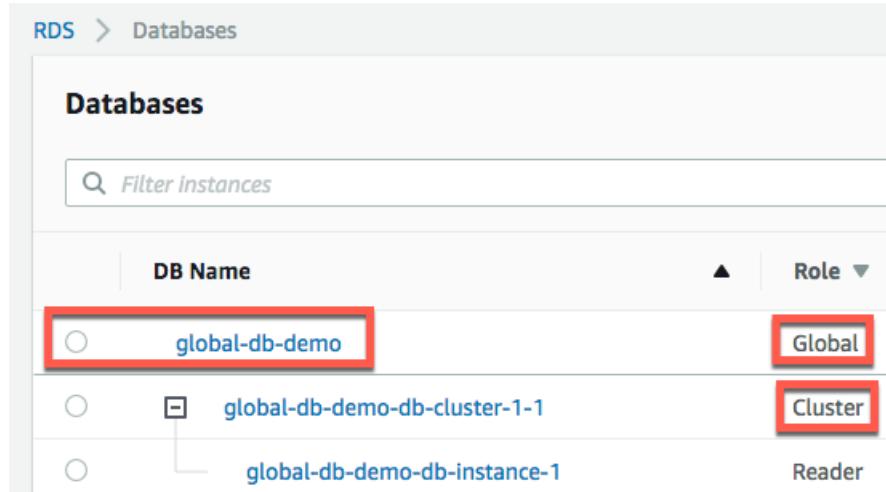
To delete a cluster that is part of an Aurora global database with the AWS Management Console, you remove or delete any secondary clusters associated with the global database, remove the primary cluster, and then delete the global database itself. Because a global database typically holds business-critical data, you can't delete the global database and the associated clusters in a single step. See [Removing a Cluster from an Aurora Global Database \(p. 37\)](#) for instructions on removing clusters from a global database. See [Deleting a DB Instance in an Aurora DB Cluster \(p. 418\)](#) for the procedure to delete clusters once they are removed. Deleting the primary instance from an Aurora cluster deletes the cluster itself.

1. Confirm that all other clusters are removed from the Aurora global database.

If the global database includes any clusters nested underneath it, as in the following screenshot, you can't delete the global database yet. Follow the procedure in [Removing a Cluster from an Aurora Global Database \(p. 37\)](#) for each cluster associated with the global database.



After the global database has no associated clusters, you can proceed to delete it. The following screenshot shows how the `global-db-demo` cluster is no longer associated with the global database after being removed.



- From the **Databases** page, or the detail page for the global database, choose **Actions** and then choose **Delete**.

AWS CLI

To delete a cluster that is part of an Aurora global database with the AWS CLI, run the [delete-global-cluster](#) command.

The following command deletes the Aurora global database with the specified global cluster ID:

For Linux, OS X, 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 with the RDS API, run the [DeleteGlobalCluster](#) operation.

Importing Data into an Aurora Global Database

To import data into an Aurora global database, use one of the following techniques:

- Create a snapshot of an Aurora MySQL cluster or Amazon RDS MySQL DB instance and restore it to the primary cluster of an Aurora global database. Currently, any snapshot must be from a source that is compatible with MySQL 5.6.

The screenshot shows the 'Restore DB Instance' wizard in the AWS RDS console. The current step is 'Instance specifications'. It includes fields for 'DB Engine' (set to 'Aurora MySQL'), 'DB Engine Version' (set to 'Aurora (MySQL)-5.6.10a (default)'), and 'Capacity type'. The 'Global' option is selected and highlighted with a red box. Other options shown are 'Provisioned', 'Provisioned with Aurora parallel query enabled', and 'Serverless'.

- Use point-in-time restore on the primary cluster to restore the cluster data to a previous state.
- Use a physical import technique, such as the Percona xtrabackup tool.
- Use a logical import technique, such as preparing data with the `mysqldump` command and loading it with the `mysql` command.

The crucial distinction for an Aurora global database is that you always import the data to the cluster that you designate as the primary cluster. You can do the initial data import before or after adding that cluster to the Aurora Global Database. All the data from the primary cluster is automatically replicated to any secondary clusters.

Managing an Aurora Global Database

You can perform most management operations on the individual clusters that make up an Aurora global database. When you select **Group related resources** on the **Databases** page in the console, you see the primary cluster and any secondary clusters grouped under the associated global database object.

The screenshot shows the 'Databases' page in the AWS Management Console. A red box highlights the 'Group related resources' toggle switch at the top of the table. The table lists various database instances with columns for DB Name, Role, CPU, Activity, Info, Engine, Size, and Region & AZ. A specific row for the 'db-global' global database is selected, and its child clusters ('db-cluster-1' and 'db-cluster-2') are also highlighted with a red box. The 'db-global' row shows '2 clusters' and '2 regions'.

DB Name	Role	CPU	Activity	Info	Engine	Size	Region & AZ
db-global	Global	-	-		Aurora MySQL	2 clusters	2 regions
db-cluster-1	Primary	-	-		Aurora MySQL	2 instances	us-east-2
db-cluster-2	Secondary	-	-		Aurora MySQL	2 instances	us-west-2
dbname-cluster2	Cluster	-	-		Aurora MySQL	2 instances	us-east-2
instance-name1-1	Writer	6.72%	1 Selects/Sec		Aurora MySQL	db.r4.xlarge	us-east-2a
instance-name1-2-us-west-2b	Reader	6.72%	1 Selects/Sec		Aurora MySQL	db.r4.xlarge	us-east-2b
instance-master-name4	Instance	6.72%	1 Selects/Sec		PostgreSQL	db.r4.xlarge	us-east-2
instance-name4	Replica	6.72%	1 Selects/Sec		PostgreSQL	db.r4.xlarge	us-east-2a
dbname3	Instance	6.72%	1 Selects/Sec		MySQL	db.r4.xlarge	us-east-2b
dbname4	Instance	6.72%	1 Selects/Sec		Oracle Enterprise Edition	db.r4.xlarge	us-east-2

To see the properties that apply to an entire Aurora global database, choose that global database.

The screenshot shows the details page for the 'db-global' global database. At the top, the 'db-global' name is selected. The 'Related' section shows the primary and secondary clusters under it. Below, the 'Maintenance' tab is selected, showing the status of Auto Minor Version Upgrade (Enabled), Maintenance Window (Mon: 01:01 - Mon: 02:01), and Pending Maintenance (None).

Configuring an Aurora Global Database

The **Databases** page in the AWS Management Console lists all your Aurora global databases, showing the primary cluster and any secondary clusters for each one. The Aurora global database is an object that has its own configuration settings, in particular the AWS Regions associated with the primary and any secondary clusters.

DB Name	Role	Region & AZ	Cross-region replication lag	Info	CPU
db-global	Global	2 regions	-	-	-
db-cluster-1	Primary	us-east-2	-	-	-
db-cluster-2	Secondary	us-west-2	20.565 Milliseconds	-	-

You can choose an Aurora global database and modify its settings, as shown in the following screenshot.

You can configure the parameter groups independently for each Aurora cluster within the Aurora global database. Most parameters work the same as for other kinds of Aurora clusters. The `aurora_enable_repl_bin_log_filtering` and `aurora_enable_replica_log_compression` configuration settings have no effect. We recommend keeping the settings consistent between all the clusters in a global database, to avoid unexpected behavior changes if you promote one of the secondary clusters 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.

Connecting to an Aurora Global Database

For read-only query traffic, 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 the Aurora global database in the AWS Management Console, you can see all the general-purpose endpoints associated with all of its clusters, as shown in the following screenshot. 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, which you use for read-only queries. Choose whichever reader endpoint is in your AWS Region or the AWS Region closest to you, to minimize latency.

Endpoint name	Status	Status	Port
gdemo-04-db-cluster-1-1.cluster- .us-east-2.rds.amazonaws.com	Available	Writer	3306
gdemo-04-db-cluster-1-1.cluster-ro- .us-east-2.rds.amazonaws.com	Available	Reader	3306

Failover for Aurora Global Database

Aurora global databases introduce a higher level of failover capability than a default Aurora cluster. If an entire cluster in one AWS Region becomes unavailable, you can promote another cluster in the global database to have read-write capability.

You can manually activate the failover mechanism if a cluster in a different AWS Region is a better choice to be the primary cluster. For example, you might increase the capacity of one of the secondary clusters and then promote it to be the primary cluster. Or the balance of activity among the AWS Regions might change, so that switching the primary cluster to a different AWS Region might give lower latency for write operations.

The following steps outline the sequence of events when you promote one of the secondary clusters in an Aurora global database:

1. The primary cluster becomes unavailable.
2. You one of the secondary clusters from the Aurora global database. Doing so promotes it to full read-write capability. To learn how to remove an Aurora cluster from a global database, see [Removing a Cluster from an Aurora Global Database \(p. 37\)](#).
3. You reconfigure your application to divert write traffic to the newly promoted cluster.

Important

At this point, you stop issuing any DML statements or other write operations to the cluster that became unavailable. To avoid data inconsistencies between the clusters, known as a "split-brain" scenario, avoid writing to the former primary cluster, even if it comes back online.

4. You create a new Aurora global database with the newly promoted cluster as the primary cluster. To learn how to create an Aurora global database, see [Creating an Aurora Global Database \(p. 30\)](#).
5. You add to the Aurora global database the AWS Region of the cluster that encountered the issue. Now that AWS Region contains a new secondary cluster. To learn how to add an AWS Region to an Aurora global database, see [Adding an AWS Region to an Aurora Global Database \(p. 35\)](#).

6. When the outage is resolved and you are ready to assign your original AWS Region as the primary cluster again, you perform the same steps in reverse:
 - a. You remove one of the secondary clusters from the Aurora global database.
 - b. You make that cluster the primary cluster of a new Aurora global database in the original AWS Region.
 - c. You redirect all the write traffic to the primary cluster in the original AWS Region.
 - d. You add an AWS Region to set up one or more secondary clusters in the same AWS Regions as before.

Performance Insights for Aurora Global Database

You can use Amazon RDS Performance Insights and Aurora Global Database together. When you do so, the Performance Insights reports apply to each cluster in the global database individually. You can enable or turn off Performance Insights for each cluster that's part of the global database. When you add a new secondary region to a global database, that's already using Performance Insights, you must 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 region might have different capacity.

For information about using Performance Insights, see [Using Amazon RDS Performance Insights \(p. 476\)](#).

Using Multiple Secondary Regions with Aurora Global Database

When your Aurora global database includes more than one secondary region, you can choose which AWS Region to fail over to in case of an outage affecting the primary AWS Region. To help determine which secondary region to choose, you can monitor the replication lag for all the secondary regions. If one secondary region has substantially less replication lag than the others, that's a good indication that the associated Aurora cluster has enough database and network capacity to take over the full read-write workload for your application.

When your global database has multiple secondary regions, you must detach all the secondary regions if the primary region experiences an outage. Then, you promote one of those secondary regions to be the new primary region. Finally, you create new clusters in each of the other secondary regions and attach those clusters to the global database.

Using Aurora Global Database with Other AWS Services

In some cases, you access other AWS services in combination with an Aurora global database. In these cases, you need the same privileges, external functions, and so on, in the corresponding AWS Regions for all the associated clusters. Even though an Aurora cluster in a global database might start out as a read-

only replication target, you might later promote it to the primary cluster. To prepare for that possibility, set up any necessary write privileges for other services ahead of time, for all Aurora clusters in the global database.

The following list summarizes the actions to take for each AWS service.

Invoking Lambda functions

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 \(p. 752\)](#).

For each cluster in the Aurora global database, set the `aws_default_lambda_role` cluster parameter to the Amazon Resource Name (ARN) of the new IAM role.

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 \(p. 734\)](#) with each cluster in the Aurora global database.

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 \(p. 738\)](#).

Loading data from S3

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 \(p. 739\)](#).

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`.

To permit database users in an Aurora global database to access Amazon S3, associate the role that you created in [Creating an IAM Role to Allow Amazon Aurora to Access AWS Services \(p. 734\)](#) with each Aurora cluster in the global database.

Configure each Aurora cluster in the global database to allow outbound connections to Amazon S3. For instructions, see [Enabling Network Communication from Amazon Aurora MySQL to Other AWS Services \(p. 738\)](#).

Saving queried data to S3

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 \(p. 747\)](#).

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`.

To permit database users in an Aurora global database to access Amazon S3, associate the role that you created in [Creating an IAM Role to Allow Amazon Aurora to Access AWS Services \(p. 734\)](#) with each Aurora cluster in the global database.

Configure each Aurora cluster in the global database to allow outbound connections to Amazon S3. For instructions, see [Enabling Network Communication from Amazon Aurora MySQL to Other AWS Services \(p. 738\)](#).

Replication with Amazon Aurora

There are several replication options with Aurora. The following sections explain how and when to choose each technique.

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. 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 instance and to Aurora Replicas in the DB cluster.

As a result, all Aurora Replicas return the same data for query results with minimal replica lag—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, and the Aurora Replicas are rebooted. If your Aurora DB cluster doesn't include any Aurora Replicas, then your DB cluster will be unavailable for the duration it takes your DB instance to recover from the failure event. However, promoting an Aurora Replica is much faster than recreating the primary instance. 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 on Aurora Replicas as failover targets, see [Fault Tolerance for an Aurora DB Cluster \(p. 380\)](#).

Note

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.

For details on how to create an Aurora Replica, see [Adding Aurora Replicas to a DB Cluster \(p. 280\)](#).

Replication with Aurora MySQL

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

- Two Aurora MySQL DB clusters in different AWS Regions, by creating an Aurora Read Replica of an Aurora MySQL DB cluster in a different AWS Region.
- Two Aurora MySQL DB clusters in the same region, by using MySQL binary log (binlog) replication.
- An Amazon RDS MySQL DB instance as the master and an Aurora MySQL DB cluster, by creating an Aurora Read Replica of an Amazon RDS MySQL DB instance. Typically, this approach is used 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 \(p. 672\)](#).

Replication with Aurora PostgreSQL

In addition to Aurora Replicas, you can set up replication between an Amazon RDS PostgreSQL DB instance as the master and an Aurora PostgreSQL DB cluster. You do so by creating an Aurora Read Replica of an Amazon RDS PostgreSQL DB instance.

For more information about replication with Aurora PostgreSQL, see [Replication with Amazon Aurora PostgreSQL \(p. 911\)](#).

Setting Up Your Environment for Amazon Aurora

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

1. [Sign Up for AWS \(p. 49\)](#)
2. [Create an IAM User \(p. 49\)](#)
3. [Determine Requirements \(p. 51\)](#)
4. [Provide Access to the DB Cluster in the VPC by Creating a Security Group \(p. 52\)](#)

Sign Up for AWS

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

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

If you have an AWS account already, skip to the next task. If you don't have an AWS account, use the following procedure to create one.

To create 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.

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

Create an IAM User

Services in AWS, such as Amazon RDS, require that you provide credentials when you access them, so that the service can determine whether you have permission to access its resources. The console requires your password. You can create access keys for your AWS account to access the command line interface or API. However, we don't recommend that you access AWS using the credentials for your AWS account; we recommend that you use AWS Identity and Access Management (IAM) instead. Create an IAM user, and then add the user to an IAM group with administrative permissions or grant this user administrative permissions. You can then access AWS using a special URL and the credentials for the IAM user.

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

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

1. Use your AWS account email address and password to sign in as the [AWS account root user](#) to the IAM console at <https://console.aws.amazon.com/iam/>.

Note

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

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

Note

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

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

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

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

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

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

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

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

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

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.

You must know your DB cluster and network needs before you create a security group and before you create a DB cluster. For example, you must know the following:

- 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 [Choosing the DB Instance Class \(p. 76\)](#).
- Your DB cluster is in a virtual private cloud (VPC). Security group rules must be configured to connect to a DB cluster. The follow list describes the rules for each VPC option:
 - **Default VPC** — If your AWS account has a default VPC in the region, that VPC is configured to support DB clusters. If you specify the default VPC when you create the DB cluster:
 - You must create a **VPC security group** that authorizes connections from the application or service to the Aurora DB cluster with the database. Note that you must use the [Amazon EC2 API](#) or the **Security Group** option on the VPC Console to create VPC security groups. For information, see [Step 4: Create a VPC Security Group \(p. 253\)](#).
 - You must specify the default DB subnet group. If this is the first DB cluster you have created in the 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:
 - You must create a **VPC security group** that authorizes connections from the application or service to the Aurora DB cluster with the database. Note that you must use the [Amazon EC2 API](#) or the **Security Group** option on the VPC Console to create VPC security groups. For information, see [Step 4: Create a VPC Security Group \(p. 253\)](#).
 - The VPC must meet certain requirements in order to host DB clusters, such as having at least two subnets, each in a separate availability zone. For information, see [Amazon Virtual Private Cloud VPCs and Amazon Aurora \(p. 239\)](#).
 - You must specify a DB subnet group that defines which subnets in that VPC can be used by the DB cluster. For information, see the DB Subnet Group section in [Working with a DB Instance in a VPC \(p. 250\)](#).
 - 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 Aurora \(p. 27\)](#).
 - Does your AWS account have policies that grant the permissions needed to perform Amazon RDS operations? If you are connecting to AWS using IAM credentials, your IAM account must have IAM policies that grant the permissions required to perform Amazon RDS operations. For more information, see [Identity and Access Management in Amazon Aurora \(p. 191\)](#).
 - What TCP/IP port will your database be listening on? The firewall at some companies may 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.

- What region do you want your database in? Having the database close in proximity to the application or web service could reduce network latency.

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 most likely 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.

The security group you need to create is a *VPC security group*, unless you have a legacy DB cluster not in a VPC that requires a *DB security group*. If you created your AWS account after March 2013, chances are very good that you have a default VPC, and your DB cluster will be created in that VPC. DB clusters in a VPC require that you add rules to a VPC security group to allow access to the cluster.

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

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>.
2. In the top right corner of the AWS Management Console, select the region in which you want to create the VPC security group and the DB cluster. In the list of Amazon VPC resources for that region, it should show that you have at least one VPC and several Subnets. If it does not, you do not have a default VPC in that region.
3. In the navigation pane, choose **Security Groups**.
4. Choose **Create Security Group**.
5. In the **Create Security Group** window, type the **Name tag**, **Group name**, and **Description** of your security group. Select the **VPC** that you want to create your DB cluster in. Choose **Yes, Create**.
6. The VPC security group you created should still be selected. The details pane at the bottom of the console window displays the details for the security group, and tabs for working with inbound and outbound rules. Choose the **Inbound Rules** tab.
7. On the **Inbound Rules** tab, choose **Edit**. Select **Custom TCP Rule** from the **Type** list. Type the port value you will use for your DB cluster in the **Port Range** text box, and then type the IP address range (CIDR value) from where you will access the cluster, or select a security group name in the **Source** text box.
8. If you need to add more IP addresses or different port ranges, choose **Add another rule**.
9. If you need to, you can use the **Outbound Rules** tab to add rules for outbound traffic.
10. When you have finished, choose **Save** on each tab with changes.

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

Finally, a quick note about VPC subnets: If you use a default VPC, a default subnet group spanning all of the VPC's subnets has already been created for you. When you create a DB cluster, you can select the default VPC and use **default** for the **DB Subnet Group**.

Once you have completed the setup requirements, you can use your requirements and the security group you created to launch a DB cluster. For information on creating a DB cluster, see the relevant documentation in the following table:

After setting up, you can create a test Amazon Aurora DB cluster. For instructions, see [Creating a DB Cluster and Connecting to a Database on an Aurora MySQL DB Cluster \(p. 54\)](#).

Getting Started with Amazon Aurora

This section shows you how to create and connect to an Aurora DB cluster using Amazon RDS.

These procedures are tutorials that demonstrate the basics of getting started with Aurora. Subsequent 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

You must complete the tasks in the [Setting Up Your Environment for Amazon Aurora \(p. 49\)](#) section before you can create a DB cluster.

Topics

- [Creating a DB Cluster and Connecting to a Database on an Aurora MySQL DB Cluster \(p. 54\)](#)
- [Creating a DB Cluster and Connecting to a Database on an Aurora PostgreSQL DB Cluster \(p. 63\)](#)

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 Amazon RDS 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.

Topics

- [Create an Aurora MySQL DB Cluster \(p. 54\)](#)
- [Connect to an Instance in a DB Cluster \(p. 61\)](#)
- [Delete the Sample DB Cluster, DB Subnet Group, and VPC \(p. 63\)](#)

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 makes it easy for you to create your own VPC for use with Amazon Aurora or use an existing VPC with your Aurora DB cluster.

If you want to create a VPC and DB subnet group for use with your Aurora DB cluster yourself, rather than having Amazon RDS create the VPC and DB subnet group for you, then follow the instructions in [How to Create a VPC for Use with Amazon Aurora \(p. 239\)](#). 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.

Note

A new console interface is available for database creation. Choose either the **New Console** or the **Original Console** instructions based on the console that you are using. The **New Console** instructions are open by default.

New Console

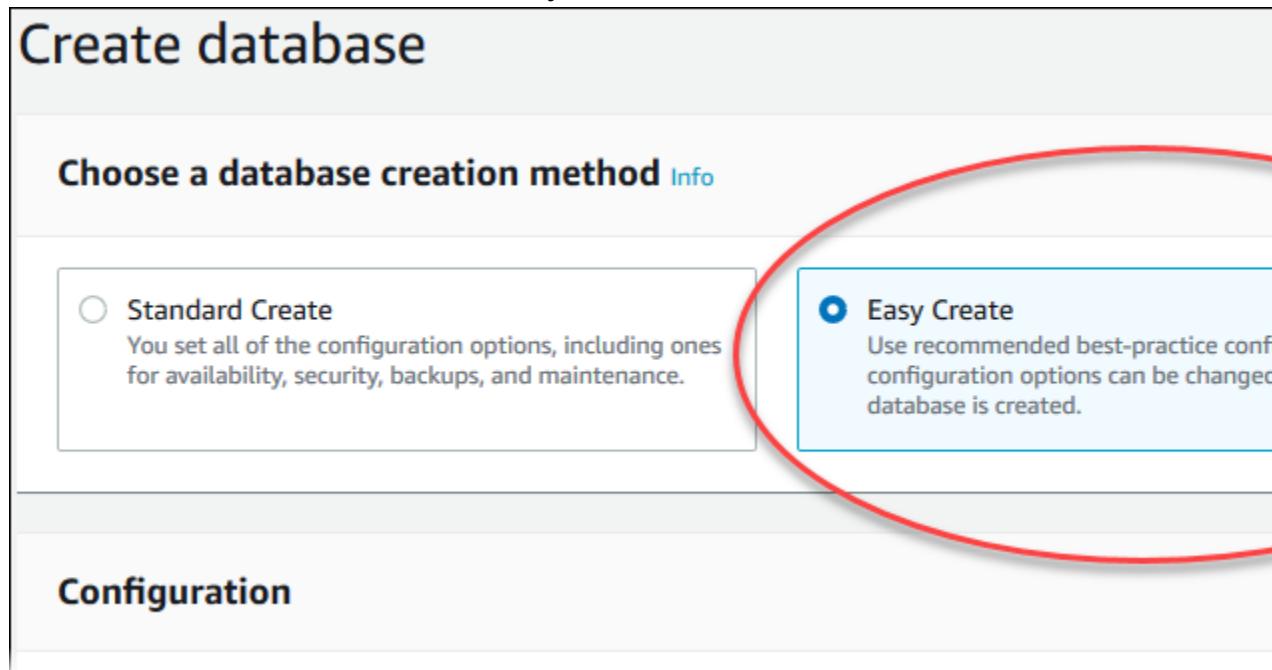
You can create an Aurora with MySQL compatibility DB cluster with the AWS Management Console with **Easy Create** enabled or disabled. With **Easy Create** enabled, you specify only the DB engine type, DB instance size, and DB instance identifier. **Easy Create** uses the default setting for other configuration options. With **Easy Create** disabled, you specify more configuration options when you create a database, including ones for availability, security, backups, and maintenance.

Note

For this example, **Easy Create** is enabled. For information about creating an Aurora MySQL DB cluster with **Easy Create** not enabled, see [Creating an Amazon Aurora DB Cluster \(p. 100\)](#).

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 instance.
Aurora is not available in all AWS Regions. For a list of AWS Regions where Aurora is available, see [Region Availability \(p. 80\)](#).
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 5.6 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 configuration options can be changed after the database is created.

Configuration

Engine type Info

Amazon Aurora



MySQL



MariaDB



PostgreSQL



Oracle

ORACLE®

Microsoft SQL Server



Edition

Amazon Aurora with MySQL 5.6 compatibility

Amazon Aurora with PostgreSQL compatibility

DB instance size

Production

db.r5.2xlarge
8 vCPUs
64 GiB RAM

Dev/Test

db.r5.large
2 vCPUs
16 GiB RAM

Serverless

4-64 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 this region.

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

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

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

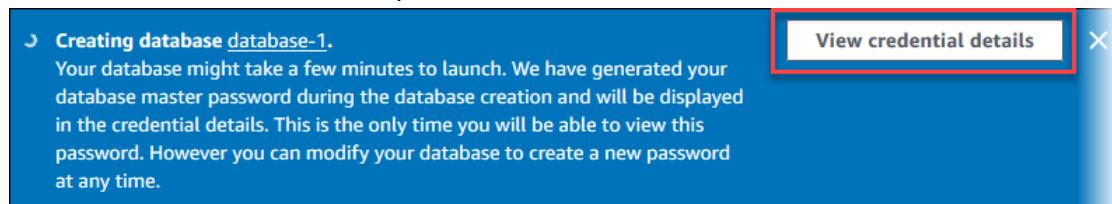
Configuration	Value	Editable after database is created
Database Location	Regional	No
Database Features	provisioned	No
Automatic Backups	Enabled	No

You can examine the default settings used when **Easy Create** is enabled. If you want to change one or more settings during database creation, choose **Standard Create** to set them. The **Editable after database creation** column shows which options you can change after database creation. To change a setting with **No** in that column, use **Standard Create**. For settings with **Yes** in that column, you can either use **Standard Create** or modify the DB instance after it's created to change the setting.

- 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 instance, choose **View credential details**.



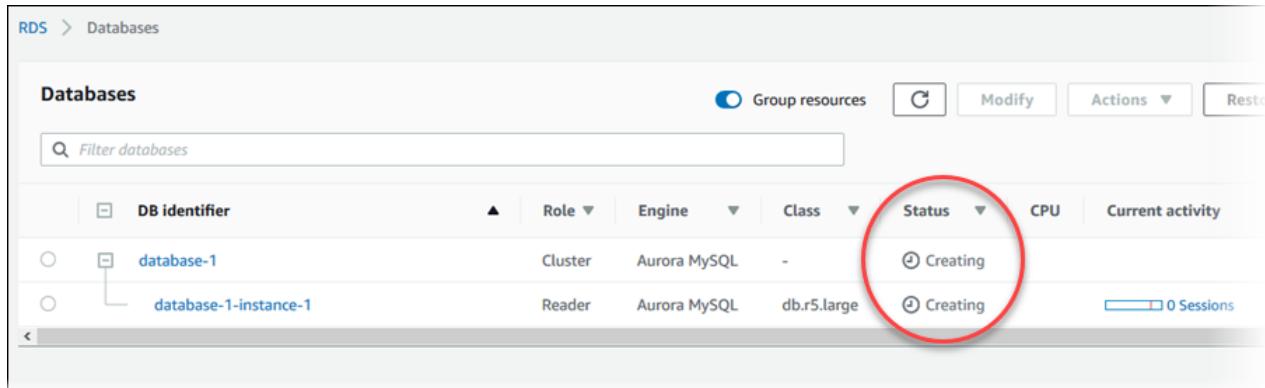
To connect to the DB instance as the master user, use the user name and password that appear.

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 \(p. 264\)](#).

- 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.



Original Console

To launch 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 top-right corner of the AWS Management Console, choose the AWS Region that you want to create your DB cluster in. For a list of AWS Regions where Aurora is available, see [Region Availability \(p. 80\)](#).
3. In the navigation pane, choose **Databases**.

If the navigation pane is closed, choose the menu icon at the top left to open it.

4. Choose **Create database** to open the **Select engine** page.
5. On the **Select engine** page, choose Amazon Aurora and choose the MySQL-compatible edition.

Select engine

Engine options

Amazon Aurora
Amazon Aurora

MySQL


MariaDB


PostgreSQL


Oracle
ORACLE

Microsoft SQL Server


Amazon Aurora
Amazon Aurora is a MySQL- and PostgreSQL-compatible enterprise-class database, starting at <\$1/day.

- Up to 5 times the throughput of MySQL and 3 times the throughput of PostgreSQL
- Up to 64TB of auto-scaling SSD storage
- 6-way replication across three Availability Zones
- Up to 15 Read Replicas with sub-10ms replica lag
- Automatic monitoring and failover in less than 30 seconds

Edition
 MySQL 5.6-compatible
 MySQL 5.7-compatible
 PostgreSQL-compatible

Only enable options eligible for RDS Free Usage Tier [info](#)

[Cancel](#) **Next**

6. Choose **Next**.

7. Set the following values on the **Specify DB details** page:

- **DB instance class:** db.r4.large
- **DB instance identifier:** gs-db-instance1
- **Master username:** Using alphanumeric characters, type a master user name, used to log on to your DB instances in the DB cluster.
- **Master password and Confirm Password:** Type a password in the **Master Password** box that contains from 8 to 41 printable ASCII characters (excluding /, ", and @) for your master user password, used to log on to your database. Then type the password again in the **Confirm Password** box.

Specify DB details

Instance specifications

Estimate your monthly costs for the DB Instance using the [AWS Simple Monthly Calculator](#).

DB engine
Aurora - compatible with MySQL 5.7.12

DB instance class [info](#)

Multi-AZ deployment [info](#)
 Create Replica in Different Zone
 No

Settings

DB instance identifier [info](#)
Specify a name that is unique for all DB instances owned by your AWS account in the current region.

DB instance identifier is case insensitive, but stored as all lower-case, as in "mydbinstance".

Master username [info](#)
Specify an alphanumeric string that defines the login ID for the master user.

Master Username must start with a letter.

Master password [info](#) Confirm password [info](#)

Master Password must be at least eight characters long, as in "mypassword".

[Cancel](#) [Previous](#) [Next](#)

8. Choose **Next** and set the following values on the **Configure Advanced Settings** page:

- **Virtual Private Cloud (VPC):** If you have an existing VPC, then you can use that VPC with your Amazon Aurora DB cluster by choosing your VPC identifier, for example `vpc-a464d1c1`. For information on using an existing VPC, see [How to Create a VPC for Use with Amazon Aurora \(p. 239\)](#).

Otherwise, you can choose to have Amazon RDS create a VPC for you by choosing **Create a new VPC**. This example uses the **Create a new VPC** option.

- **Subnet group:** If you have an existing subnet group, then you can use that subnet group with your Amazon Aurora DB cluster by choosing your subnet group identifier, for example, gs-subnet-group1.

Otherwise, you can choose to have Amazon RDS create a subnet group for you by choosing **Create a new subnet group**. This example uses the **Create a new subnet group** option.

- **Public accessibility:** 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:** No Preference

- **VPC security groups:** If you have one or more existing VPC security groups, then you can use one or more of those VPC security groups with your Amazon Aurora DB cluster by choosing your VPC security group identifiers, for example, gs-security-group1.

Otherwise, you can choose to have Amazon RDS create a VPC security group for you by choosing **Create new VPC security group**. This example uses the **Create new VPC security group** option.

- **DB cluster identifier:** gs-db-cluster1
- **Database name:** sampledb

Note

This creates the default database. To create additional databases, connect to the DB cluster and use the SQL command `CREATE DATABASE`.

- **Database port:** 3306

Note

You might be behind a corporate firewall that does not allow access to default ports such as the Aurora 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 DB cluster.

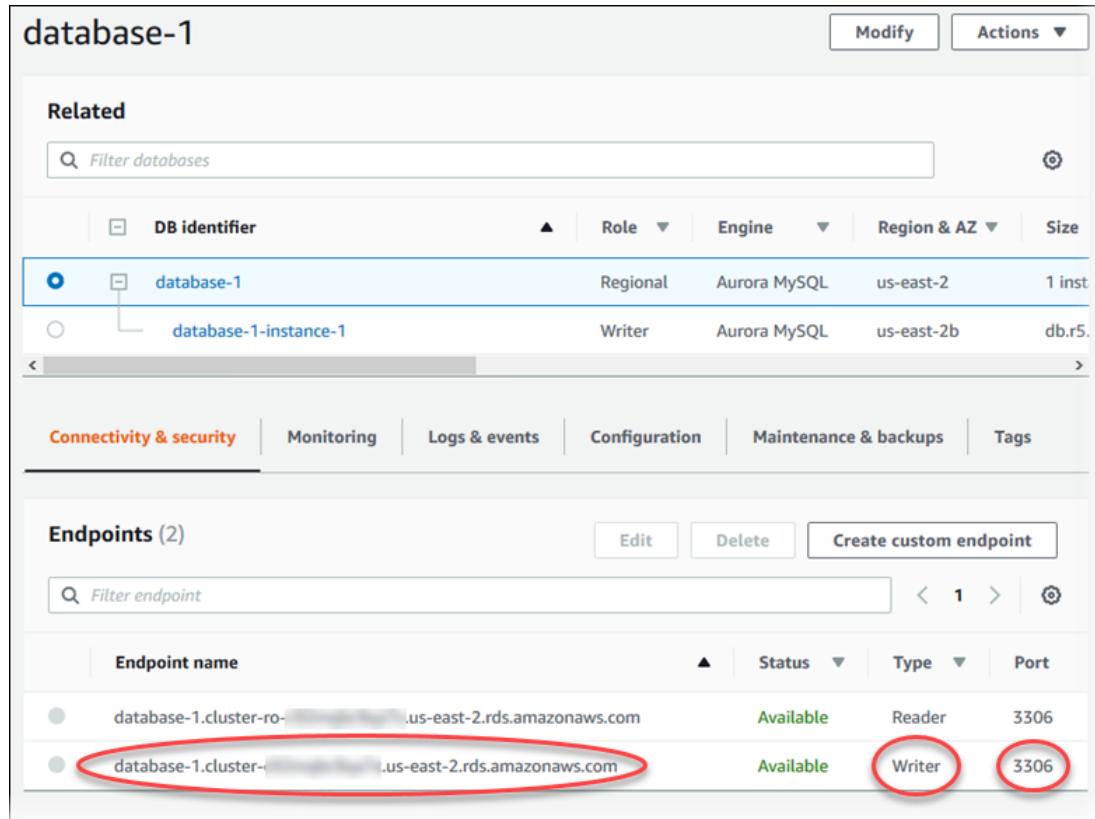
9. Leave the rest of the values as their defaults, and choose **Create database** to create the DB cluster and primary instance.

Connect to an Instance in a DB Cluster

Once 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 this example, you connect to a database on the Aurora MySQL DB cluster using MySQL monitor commands. One GUI-based application that you can use to connect is MySQL Workbench. For more information, go to the [Download MySQL Workbench](#) page.

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

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 cluster name to show its details. On the **Connectivity & security** tab, copy the value for the **Endpoint name** of the **Writer** endpoint. Also, note the port number for the endpoint.



The screenshot shows the AWS Aurora console for the 'database-1' cluster. At the top, there are 'Modify' and 'Actions' buttons. Below that is a 'Related' section with a search bar. The main table lists database identifiers, roles, engines, regions, and sizes. Under 'database-1', it shows 'database-1-instance-1' with a 'Writer' role. 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: 'database-1.cluster-ro...' (Status: Available, Type: Reader, Port: 3306) and 'database-1.cluster-...' (Status: Available, Type: Writer, Port: 3306). The 'Writer' and '3306' entries are circled in red.

3. Type 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 <mymasteruser> -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 \(p. 166\)](#).

Delete the Sample DB Cluster, DB Subnet Group, and VPC

Once 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.

Topics

- [Create an Aurora PostgreSQL DB Cluster \(p. 64\)](#)
- [Connect to an Instance in an Aurora PostgreSQL DB Cluster \(p. 72\)](#)
- [Delete the Sample DB Cluster, DB Subnet Group, and VPC \(p. 73\)](#)

Create an Aurora PostgreSQL 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 makes it easy for you to create your own VPC for use with Amazon Aurora or use an existing VPC with your Aurora DB cluster.

If you want to create a VPC and DB subnet group for use with your Amazon Aurora DB cluster yourself, rather than having Amazon RDS create the VPC and DB subnet group for you, follow the instructions in [How to Create a VPC for Use with Amazon Aurora \(p. 239\)](#). 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.

Note

A new console interface is available for database creation. Choose either the **New Console** or the **Original Console** instructions based on the console that you are using. The **New Console** instructions are open by default.

New Console

You can create an Aurora PostgreSQL DB cluster with the AWS Management Console with **Easy Create** enabled or disabled. 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** disabled, you specify more configuration options when you create a database, including ones for availability, security, backups, and maintenance.

Note

For this example, **Easy Create** is enabled. For information about creating an Aurora PostgreSQL DB cluster with **Easy Create** not enabled, see [Creating an Amazon Aurora DB Cluster \(p. 100\)](#).

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 instance.

Aurora is not available in all AWS Regions. For a list of AWS Regions where Aurora is available, see [Region Availability \(p. 80\)](#).
3. In the navigation pane, choose **Databases**.
4. Choose **Create database** and make sure that **Easy Create** is chosen.

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 configuration options can be changed after the database is created.

Configuration

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 configuration options can be changed after the database is created.

Configuration

Engine type Info

Amazon Aurora



MySQL



MariaDB



PostgreSQL



Oracle

ORACLE®

Microsoft SQL Server



Edition

Amazon Aurora with MySQL 5.6 compatibility

Amazon Aurora with PostgreSQL compatibility

DB instance size

Production

db.r4.2xlarge
8 vCPUs
61 GiB RAM

Dev/Test

db.r4.large
2 vCPUs
15.25 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 region.

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

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

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

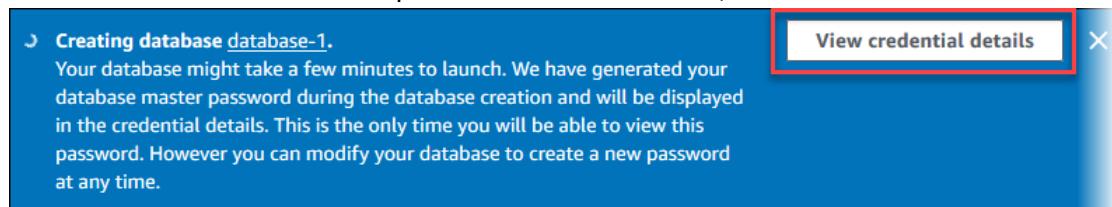
Configuration	Value	Editable after database is created
Database Location	Regional	No
Database Features	provisioned	No
Automatic Backups	Enabled	No

You can examine the default settings used when **Easy Create** is enabled. If you want to change one or more settings during database creation, choose **Standard Create** to set them. The **Editable after database creation** column shows which options can be changed after database creation. To change a setting with **No** in that column, use **Standard Create**. For settings with **Yes** in that column, you can either use **Standard Create**, or you can modify the DB instance after it is created to change the setting.

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 instance, choose **View credential details**.



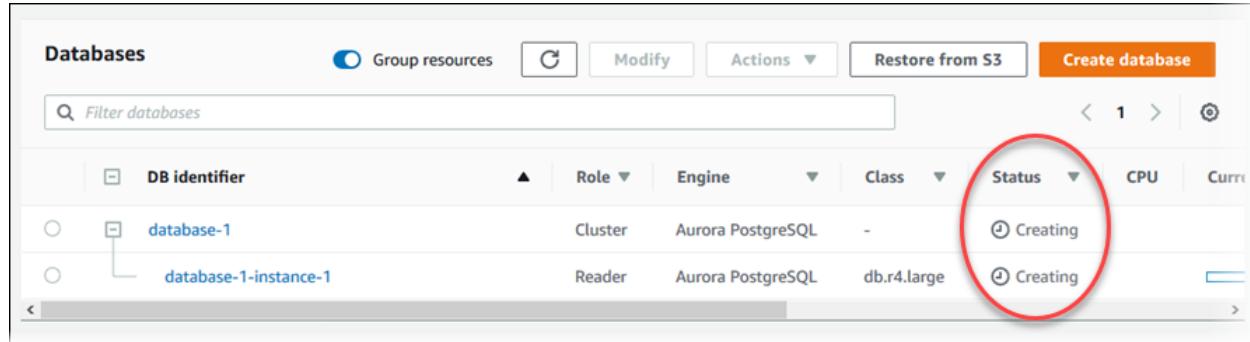
To connect to the DB instance as the master user, use the user name and password that appear.

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 \(p. 264\)](#).

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.



The screenshot shows the 'Databases' page in the AWS Management Console. At the top, there are buttons for 'Group resources', 'Modify', 'Actions', 'Restore from S3', and a prominent orange 'Create database' button. Below this is a search bar labeled 'Filter databases'. The main area is a table with columns: 'DB identifier', 'Role', 'Engine', 'Class', 'Status', 'CPU', and 'Current storage'. There are two entries: 'database-1' (Cluster, Aurora PostgreSQL, db.r4.large, Status: Creating) and 'database-1-instance-1' (Reader, Aurora PostgreSQL, db.r4.large, Status: Creating). A red circle highlights the 'Status' column header and the 'Creating' status for both entries.

DB identifier	Role	Engine	Class	Status	CPU	Current storage
database-1	Cluster	Aurora PostgreSQL	-	Creating		
database-1-instance-1	Reader	Aurora PostgreSQL	db.r4.large	Creating		

Original Console

To launch an 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 top-right corner of the AWS Management Console, choose the AWS Region that you want to create your DB cluster in. For a list of AWS Regions where Aurora is available, see [Region Availability \(p. 80\)](#).
3. In the navigation pane, choose **Databases**.
If the navigation pane is closed, choose the menu icon at the top left to open it.
4. Choose **Create database** to open the **Select engine** page.
5. On the **Select engine** page, choose Amazon Aurora and choose the PostgreSQL-compatible edition.

RDS > Create database

Select engine

Engine options

Amazon Aurora

Amazon Aurora

MySQL



MariaDB



PostgreSQL



Oracle

ORACLE®

Microsoft SQL Server



Amazon Aurora

Amazon Aurora is a MySQL- and PostgreSQL-compatible enterprise-class database, starting at <\$1/day.

- Up to 5 times the throughput of MySQL and 3 times the throughput of PostgreSQL
- Up to 64TiB of auto-scaling SSD storage
- 6-way replication across three Availability Zones
- Up to 15 Read Replicas with sub-10ms replica lag
- Automatic monitoring and failover in less than 30 seconds

Edition

MySQL 5.6-compatible

Aurora Serverless and Parallel Query capacities are only available with this edition.

MySQL 5.7-compatible

PostgreSQL-compatible

i **Aurora global database feature is now available.**
This feature is now available in our new database creation flow.

Try it now

Only enable options eligible for RDS Free Usage Tier [Info](#)

[Cancel](#) **Next**

6. Choose **Next**.

7. Set the following values on the **Specify DB details** page:

- **DB instance class:** db.r4.large
- **DB instance identifier:** aurora-postgres-db-instance1
- **Master username:** Using alphanumeric characters, type a master user name, used to log on to your DB instances in the DB cluster.
- **Master password and Confirm Password:** Type a password in the **Master Password** box that contains from 8 to 41 printable ASCII characters (excluding /, ", and @) for your master user password, used to log on to your database. Then type the password again in the **Confirm Password** box.

Specify DB details

Instance specifications

Estimate your monthly costs for the DB Instance using the [AWS Simple Monthly Calculator](#)

DB engine

Aurora PostgreSQL

Capacity type [Info](#)

Provisioned

You provision and manage the server instance sizes.

Provisioned with Aurora parallel query enabled [Info](#)

You provision and manage the server instance sizes, and Aurora improves the performance of analytic queries by pushing processing down to the Aurora storage layer (currently available for Aurora MySQL 5.6)

Serverless [Info](#)

You specify the minimum and maximum of resources for a DB cluster. Aurora scales the capacity based on database load.

DB engine version [Info](#)

Aurora PostgreSQL (compatible with PostgreSQL 9.6.9)

DB instance class [Info](#)

db.r4.large — 2 vCPU, 15.25 GiB RAM

Multi-AZ deployment [Info](#)

Create Replica in Different Zone

No

Settings

DB instance identifier [Info](#)

Specify a name that is unique for all DB instances owned by your AWS account in the current region.

aurora-postgres-db-instance1

DB instance identifier is case insensitive, but stored as all lower-case, as in "mydbinstance". Must contain from 1 to 63 alphanumeric characters or hyphens (1 to 15 for SQL Server). First character must be a letter. Cannot end with a hyphen or contain two consecutive hyphens.

Master username [Info](#)

Specify an alphanumeric string that defines the login ID for the master user.

myawsuser

Master Username must start with a letter. Must contain 1 to 16 alphanumeric characters.

Master password [Info](#)

Confirm password [Info](#)

Master Password must be at least eight characters long, as in "mypassword". Can be any printable ASCII character except "/", "", or "@".

8. Choose **Next** and set the following values on the **Configure Advanced Settings** page:

- **Virtual Private Cloud (VPC):** If you have an existing VPC, then you can use that VPC with your Amazon Aurora DB cluster by choosing your VPC identifier, for example `vpc-a464d1c1`. For information on using an existing VPC, see [How to Create a VPC for Use with Amazon Aurora \(p. 239\)](#).

Otherwise, you can choose to have Amazon RDS create a VPC for you by choosing **Create a new VPC**. This example uses the **Create a new VPC** option.

- **Subnet group:** If you have an existing subnet group, then you can use that subnet group with your Amazon Aurora DB cluster by choosing your subnet group identifier, for example, `gs-subnet-group1`.

Otherwise, you can choose to have Amazon RDS create a subnet group for you by choosing **Create a new subnet group**. This example uses the **Create a new subnet group** option.

- **Public accessibility:** 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:** No Preference

- **VPC security groups:** If you have one or more existing VPC security groups, then you can use one or more of those VPC security groups with your Amazon Aurora DB cluster by choosing your VPC security group identifiers, for example, `gs-security-group1`.

Otherwise, you can choose to have Amazon RDS create a VPC security group for you by choosing **Create new VPC security group**. This example uses the **Create new VPC security group** option.

- **DB cluster identifier:** `aurora-postgres-db-cluster1`
- **Database name:** `sampledb`

Note

This creates the default database. To create additional databases, connect to the DB cluster and use the SQL command `CREATE DATABASE`.

- **Database port:** 5432

Note

You might be behind a corporate firewall that does not allow access to default ports such as the Aurora 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 DB cluster.

9. Leave the rest of the values as their defaults, and choose **Create database** to create the DB cluster and primary instance.

Connect to an Instance in an Aurora PostgreSQL DB Cluster

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

- 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** 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	5432
database-1.cluster-...us-west-1.rds.amazonaws.com	Available	Writer	5432

- For information about connecting to the DB cluster using the endpoint and port, see [Connecting to an Amazon Aurora PostgreSQL DB Cluster \(p. 168\)](#)

Delete the Sample DB Cluster, DB Subnet Group, and VPC

Once 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

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

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

- [Choosing the DB Instance Class \(p. 76\)](#)
- [Choosing the Regions and Availability Zones \(p. 80\)](#)
- [DB Instance Billing for Aurora \(p. 87\)](#)
- [Creating an Amazon Aurora DB Cluster \(p. 100\)](#)
- [Using Amazon Aurora Serverless \(p. 116\)](#)
- [Connecting to an Amazon Aurora DB Cluster \(p. 166\)](#)
- [Migrating Data to an Amazon Aurora DB Cluster \(p. 172\)](#)

Choosing the DB Instance Class

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

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

Topics

- [DB Instance Class Types \(p. 76\)](#)
- [Terminology for DB Instance Class Hardware Specifications \(p. 76\)](#)
- [Hardware Specifications for All Available DB Instance Classes for Aurora \(p. 77\)](#)

DB Instance Class Types

Amazon Aurora supports two types of instance classes: Memory Optimized and Burstable Performance. For more information about Amazon EC2 instance types, see [Instance Type](#) in the Amazon EC2 documentation.

The following are the Memory Optimized DB instance classes available:

- **db.r5** – Latest-generation instance classes optimized for memory-intensive applications. These offer improved networking performance. They are powered by the AWS Nitro System, a combination of dedicated hardware and lightweight hypervisor.
- **db.r4** – Current-generation instance classes optimized for memory-intensive applications. These offer improved networking performance.
- **db.r3** – Previous-generation instance classes that provide memory optimization. The db.r3 instances classes are not available in the Europe (Paris) region.

The following are the Burstable Performance DB instance classes available:

- **db.t3** – Latest-generation instance classes that provide a baseline performance level, with the ability to burst to full CPU usage. 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.
- **db.t2** – Current-generation instance classes that provide a baseline performance level, with the ability to burst to full CPU usage. We recommend using these instance classes only for development and test servers, or other nonproduction servers.

Note

The DB instance classes that use the AWS Nitro System (db.r5, db.t3) are throttled on combined read plus write workload.

Terminology for DB Instance Class Hardware Specifications

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.
- **Network Performance** – The network speed relative to other DB instance classes.

Hardware Specifications for All Available DB Instance Classes for Aurora

In the following table, you can find details about the Amazon RDS DB instance classes available for Amazon Aurora. For a more detailed explanation of the table column terminology, see [Terminology for DB Instance Class Hardware Specifications \(p. 76\)](#).

The following are DB engine considerations for DB instance classes:

- **Aurora Support for db.r5** – These instance classes are available in all Aurora regions except AWS GovCloud (US-West), AWS GovCloud (US-East), and China (Beijing).
 - Aurora MySQL versions support the db.r5 instance classes as specified in the following table.
 - For Aurora PostgreSQL, only versions compatible with PostgreSQL 10.6 or later support the db.r5 instance classes.
- **Aurora Support for db.t3**
 - Aurora MySQL supports the db.t3.medium and db.t3.small instance classes for Aurora MySQL 1.15 and higher, and all Aurora MySQL 2.x versions. These instance classes are available for Aurora MySQL in all Aurora regions except AWS GovCloud (US-West), AWS GovCloud (US-East), and China (Beijing).
 - For Aurora MySQL db.r5, db.r4, and db.t3 DB instance classes, no instances in the cluster can have pending instance-level system updates. To see pending system updates, use the following AWS CLI command.

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

- Aurora PostgreSQL supports only the db.t3.medium instance class for versions compatible with PostgreSQL 10.7 or later. These instance classes are available for Aurora PostgreSQL in all Aurora regions except China (Ningxia).

Instance Class	vCPU	ECU	Memory (GiB)	Max. Bandwidth (Mbps)	Network Performance	Aurora MySQL	Aurora PostgreSQL
db.r5 – Latest Generation Memory Optimized Instance Classes							
db.r5.24xlarge	96	347	768	19,000	25 Gbps	1.22 and later, 2.06	Yes

							and later	
db.r5.16xlarge	64	264	512	13,600	20 Gbps	1.22 and later, 2.06 and later	No	
db.r5.12xlarge	48	173	384	9,500	10 Gbps	1.14.4 and later	Yes	
db.r5.8xlarge	32	132	256	6,800	10 Gbps	1.22 and later, 2.06 and later	No	
db.r5.4xlarge	16	71	128	4,750	Up to 10 Gbps	1.14.4 and later	Yes	
db.r5.2xlarge*	8	38	64	Up to 4,750	Up to 10 Gbps	1.14.4 and later	Yes	
db.r5.xlarge*	4	19	32	Up to 4,750	Up to 10 Gbps	1.14.4 and later	Yes	
db.r5.large*	2	10	16	Up to 4,750	Up to 10 Gbps	1.14.4 and later	Yes	

db.r4 – Current Generation Memory Optimized Instance Classes

db.r4.16xlarge	64	195	488	14,000	25 Gbps	1.14.4 and later	Yes
db.r4.8xlarge	32	99	244	7,000	10 Gbps	1.14.4 and later	Yes
db.r4.4xlarge	16	53	122	3,500	Up to 10 Gbps	1.14.4 and later	Yes
db.r4.2xlarge	8	27	61	1,700	Up to 10 Gbps	1.14.4 and later	Yes
db.r4.xlarge	4	13.5	30.5	850	Up to 10 Gbps	1.14.4 and later	Yes

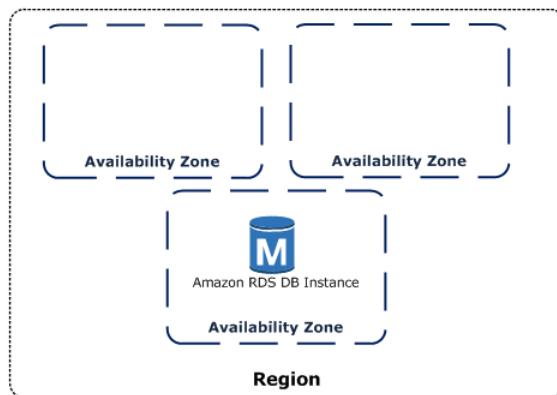
db.r4.large	2	7	15.25	425	Up to 10 Gbps	1.14.4 and later	Yes
db.r3 – Previous Generation Memory Optimized Instance Classes							
db.r3.8xlarge	32	104	244	—	10 Gbps	Yes	No
db.r3.4xlarge	16	52	122	2,000	High	Yes	No
db.r3.2xlarge	8	26	61	1,000	High	Yes	No
db.r3.xlarge	4	13	30.5	500	Moderate	Yes	No
db.r3.large	2	6.5	15.25	—	Moderate	Yes	No
Instance Class	vCPU	ECU	Memory (GiB)	Max. Bandwidth (Mbps)	Network Performance	Aurora MySQL	Aurora PostgreSQL
db.t3 – Latest Generation Burstable Performance Instance Classes							
db.t3.2xlarge*	8	Variable	32	Up to 2,048	Up to 5 Gbps	No	No
db.t3.xlarge*	4	Variable	16	Up to 2,048	Up to 5 Gbps	No	No
db.t3.large*	2	Variable	8	Up to 2,048	Up to 5 Gbps	No	No
db.t3.medium*	2	Variable	4	Up to 1,536	Up to 5 Gbps	1.14.4 and later	Yes
db.t3.small*	2	Variable	2	Up to 1,536	Up to 5 Gbps	1.14.4 and later	No
db.t3.micro*	2	Variable	1	Up to 1,536	Up to 5 Gbps	No	No
db.t2 – Current Generation Burstable Performance Instance Classes							
db.t2.medium	2	Variable	4	—	Moderate	Yes	No
db.t2.small	1	Variable	2	—	Low	Yes	No

* These DB instance classes can support maximum performance for 30 minutes at least once every 24 hours. For more information on baseline performance of these instance types, see [Amazon EBS-Optimized Instances](#) in the *Amazon EC2 User Guide for Linux Instances*.

Choosing the 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*. Amazon RDS provides you the ability to place resources, such as instances, and data in multiple locations. Resources aren't replicated across AWS Regions unless you do so specifically.

Amazon operates state-of-the-art, highly-available data centers. Although rare, failures can occur that affect the availability of instances that are in the same location. If you host all your instances in a single location that is affected by such a failure, none of your instances would 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 EC2_REGION environment variable, or it can be overridden by using the --region parameter with the AWS Command Line Interface. See [Configuring the AWS Command Line Interface](#), specifically, the sections on Environment Variables and Command Line Options for more information.

Amazon RDS supports a special AWS Region called AWS GovCloud (US-West) that is designed to allow US government agencies and customers to move more sensitive workloads into the cloud. AWS GovCloud (US-West) addresses the US government's specific regulatory and compliance requirements. For more information about AWS GovCloud (US-West), see [What Is AWS GovCloud \(US-West\)?](#)

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

Region Availability

Topics

- [Aurora MySQL Region Availability \(p. 80\)](#)
- [Aurora PostgreSQL Region Availability \(p. 82\)](#)

Aurora MySQL Region Availability

The following table shows the regions where Aurora MySQL is currently available.

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
Asia Pacific (Hong Kong)	ap-east-1	rds.ap-east-1.amazonaws.com	HTTPS
Asia Pacific (Mumbai)	ap-south-1	rds.ap-south-1.amazonaws.com	HTTPS
Asia Pacific (Seoul)	ap-northeast-2	rds.ap-northeast-2.amazonaws.com	HTTPS
Asia Pacific (Singapore)	ap-southeast-1	rds.ap-southeast-1.amazonaws.com	HTTPS
Asia Pacific (Sydney)	ap-southeast-2	rds.ap-southeast-2.amazonaws.com	HTTPS
Asia Pacific (Tokyo)	ap-northeast-1	rds.ap-northeast-1.amazonaws.com	HTTPS
Canada (Central)	ca-central-1	rds.ca-central-1.amazonaws.com	HTTPS
China (Ningxia)	cn-northwest-1	rds.cn-northwest-1.amazonaws.com.cn	HTTPS
EU (Frankfurt)	eu-central-1	rds.eu-central-1.amazonaws.com	HTTPS
EU (Ireland)	eu-west-1	rds.eu-west-1.amazonaws.com	HTTPS
EU (London)	eu-west-2	rds.eu-west-2.amazonaws.com	HTTPS
EU (Paris)	eu-west-3	rds.eu-west-3.amazonaws.com	HTTPS
EU (Stockholm)	eu-north-1	rds.eu-north-1.amazonaws.com	HTTPS

Region Name	Region	Endpoint	Protocol	
Middle East (Bahrain)	me-south-1	rds.me-south-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 regions where Aurora PostgreSQL is currently available.

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	
Asia Pacific (Hong Kong)	ap-east-1	rds.ap-east-1.amazonaws.com	HTTPS	
Asia Pacific (Mumbai)	ap-south-1	rds.ap-south-1.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	

Region Name	Region	Endpoint	Protocol	
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	
China (Ningxia)	cn-northwest-1	rds.cn-northwest-1.amazonaws.com.cn	HTTPS	
EU (Frankfurt)	eu-central-1	rds.eu-central-1.amazonaws.com	HTTPS	
EU (Ireland)	eu-west-1	rds.eu-west-1.amazonaws.com	HTTPS	
EU (London)	eu-west-2	rds.eu-west-2.amazonaws.com	HTTPS	
EU (Paris)	eu-west-3	rds.eu-west-3.amazonaws.com	HTTPS	
EU (Stockholm)	eu-north-1	rds.eu-north-1.amazonaws.com	HTTPS	
Middle East (Bahrain)	me-south-1	rds.me-south-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	

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. 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 \(p. 288\)](#).

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	
America/Guatemala	
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	
America/Matamoros	
America/Monterrey	
America/Montevideo	
America/Phoenix	
America/Tijuana	
Asia/Ashgabat	
Asia/Baghdad	
Asia/Baku	
Asia/Bangkok	

Time Zone	Notes
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	
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	

Time Zone	Notes
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	

DB Instance Billing for Aurora

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

- DB instance hours (per hour) – Based on the DB instance class of the DB instance (for example, db.t2.small or db.m4.large). Pricing is listed on a per-hour basis, but bills are calculated down to the second and show times in decimal form. RDS usage is billed in one second increments, with a minimum of 10 minutes. For more information, see [Choosing the DB Instance Class \(p. 76\)](#).
- 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 \(p. 379\)](#).

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

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

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

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

Topics

- [On-Demand DB Instances for Aurora \(p. 88\)](#)
- [Reserved DB Instances for Aurora \(p. 89\)](#)

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). Partial DB instance hours consumed are billed as full hours. 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 [DB Instance Status \(p. 454\)](#).

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.

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

Offering Types

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

No Upfront

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

Partial Upfront

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

All Upfront

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

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

Size-Flexible Reserved DB Instances

When you purchase a reserved DB instance, one thing that you specify is the instance class, for example db.m4.large. For more information about instance classes, see [Choosing the DB Instance Class \(p. 76\)](#).

If you have a DB instance, and you need to scale it to larger capacity, your reserved DB instance is automatically applied to your scaled DB instance. That is, your reserved DB instances are automatically

applied across all DB instance class sizes. Size-flexible reserved DB instances are available for DB instances with the same AWS Region and database engine. Size-flexible reserved DB instances can only scale in their instance class type. For example, a reserved DB instance for a db.m4.large can apply to a db.m4.xlarge, but not to a db.m5.large, because db.m4 and db.m5 are different instance class types. Reserved DB instance benefits also apply for both Multi-AZ and Single-AZ configurations.

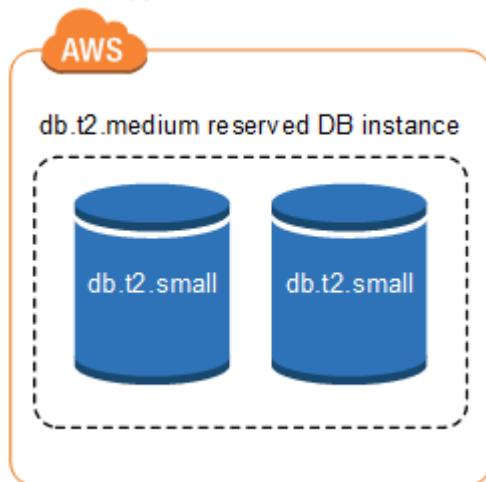
Size-flexible reserved DB instances are available for the following Aurora database engines:

- Aurora MySQL
- Aurora PostgreSQL

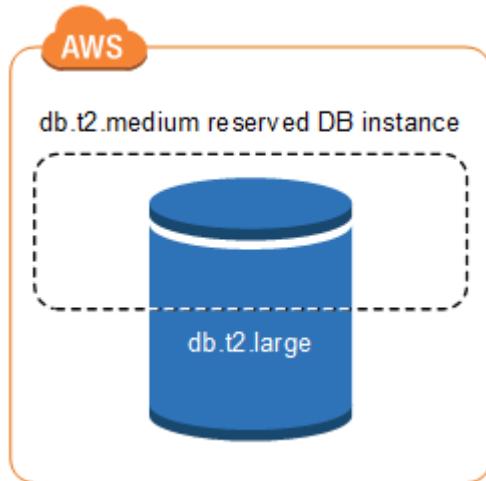
You can compare usage for different reserved DB instance sizes by using normalized units. For example, one unit of usage on two db.m3.large DB instances is equivalent to eight normalized units of usage on one db.m3.small. The following table shows the number of normalized units for each DB instance size.

Instance Size	Single-AZ Normalized Units	Multi-AZ Normalized Units
micro	0.5	1
small	1	2
medium	2	4
large	4	8
xlarge	8	16
2xlarge	16	32
4xlarge	32	64
8xlarge	64	128
10xlarge	80	160
16xlarge	128	256

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.

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.

Your upfront payment for a reserved DB instance reserves the resources for your use. Because these resources are reserved for you, you are billed for the resources regardless of whether you use them.

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).

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.

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.

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, as shown following.

Purchase Reserved DB Instances

Choose from the options below, then enter the number of DB instances you wish to reserve with this order. When you are done, click the Continue button.

Options

Product description

aurora-mysql

DB instance class

db.r4.4xlarge — 16 vCPU, 122 GiB RAM

Multi AZ deployment

Multi-AZ deployment model is not applicable for this database engine and edition

- Yes
 No

Term

1 year

Offering type

All Upfront

Reserved Id (optional)

Optional tag to track your reservation

Number of DB instances

1

Pricing details

One-time payment (per instance)

Total one-time payment*

*Additional taxes may apply

Normalized units per hour [info](#)

32

Usage charges*

USD (hourly)

*Additional taxes may apply

This hourly rate is charged for every hour for each instance in the Reserved Instance term you purchase, regardless of instance usage

Charges for your usage will appear on your monthly bill.

Cancel

Continue

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 Instance** dialog box appears, with a summary of the reserved DB instance attributes that you've selected and the payment due, as shown following.

Purchase Reserved DB Instances

Summary of Purchase
You are about to purchase a Reserved DB Instance with the following information.

Region	US East (N. Virginia)
Product Description	aurora-mysql
DB Instance Class	db.r4.4xlarge
Offering Type	All Upfront
Multi AZ Deployment	No
Term	1 year
Reserved DB Instance	default
Quantity	1
Price Per Instance	[REDACTED]
Total Payment Due Now	[REDACTED]

⚠️ 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?

Cancel **Back** **Purchase**

11. On the confirmation page, review your reserved DB instance. If the information is correct, choose **Purchase** 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 Get Available Reserved DB Instance Offerings

To get information about available reserved DB instance offerings, call the AWS CLI command `describe-reserved-db-instances-offerings`.

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

This call returns output similar to the following:

```
OFFERING OfferingId          Class      Multi-AZ Duration Fixed
Price Usage Price Description Offering Type
OFFERING 438012d3-4052-4cc7-b2e3-8d3372e0e706 db.m1.large  y     1y    1820.00
USD   0.368 USD   mysql      Partial Upfront
OFFERING 649fd0c8-cf6d-47a0-bfa6-060f8e75e95f db.m1.small  n     1y    227.50
USD   0.046 USD   mysql      Partial Upfront
OFFERING 123456cd-ab1c-47a0-bfa6-12345667232f db.m1.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.m1.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.m1.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 as shown in the following example.

Example Purchase a Reserved DB Instance

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 you track them.

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

For Linux, OS X, or Unix:

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

For Windows:

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

The command returns output similar to the following:

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

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

Example Get Your Reserved DB Instances

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

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

The command returns output similar to the following:

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

RDS API

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

Example Get Available Reserved DB Instance Offerings

To get information about available reserved DB instance offerings, call the Amazon RDS API function [DescribeReservedDBInstancesOfferings](#).

```
https://rds.us-east-1.amazonaws.com/
?Action=DescribeReservedDBInstancesOfferings
&SignatureMethod=HmacSHA256
&SignatureVersion=4
```

```
&Version=2014-09-01
&X-Amz-Algorithm=AWS4-HMAC-SHA256
&X-Amz-Credential=AKIADQKE4SARGYLE/20140411/us-east-1/rds/aws4_request
&X-Amz-Date=20140411T203327Z
&X-Amz-SignedHeaders=content-type;host;user-agent;x-amz-content-sha256;x-amz-date
&X-Amz-Signature=545f04acffeb4b80d2e778526b1c9da79d0b3097151c24f28e83e851d65422e2
```

This call returns output similar to the following:

```
<DescribeReservedDBInstancesOfferingsResponse xmlns="http://rds.amazonaws.com/doc/2014-10-31/">
  <DescribeReservedDBInstancesOfferingsResult>
    <ReservedDBInstancesOfferings>
      <ReservedDBInstancesOffering>
        <Duration>31536000</Duration>
        <OfferingType>Partial Upfront</OfferingType>
        <CurrencyCode>USD</CurrencyCode>
        <RecurringCharges/>
        <FixedPrice>1820.0</FixedPrice>
        <ProductDescription>mysql</ProductDescription>
        <UsagePrice>0.368</UsagePrice>
        <MultiAZ>true</MultiAZ>
        <ReservedDBInstancesOfferingId>438012d3-4052-4cc7-b2e3-8d3372e0e706</
      ReservedDBInstancesOfferingId>
        <DBInstanceClass>db.m1.large</DBInstanceClass>
      </ReservedDBInstancesOffering>
      <ReservedDBInstancesOffering>
        <Duration>31536000</Duration>
        <OfferingType>Partial Upfront</OfferingType>
        <CurrencyCode>USD</CurrencyCode>
        <RecurringCharges/>
        <FixedPrice>227.5</FixedPrice>
        <ProductDescription>mysql</ProductDescription>
        <UsagePrice>0.046</UsagePrice>
        <MultiAZ>false</MultiAZ>
        <ReservedDBInstancesOfferingId>649fd0c8-cf6d-47a0-bfa6-060f8e75e95f</
      ReservedDBInstancesOfferingId>
        <DBInstanceClass>db.m1.small</DBInstanceClass>
      </ReservedDBInstancesOffering>
    </ReservedDBInstancesOfferings>
  </DescribeReservedDBInstancesOfferingsResult>
  <ResponseMetadata>
    <RequestId>5e4ec40b-2978-11e1-9e6d-771388d6ed6b</RequestId>
  </ResponseMetadata>
</DescribeReservedDBInstancesOfferingsResponse>
```

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 example.

Example Purchase a Reserved DB Instance

To purchase a reserved DB instance, call the Amazon RDS API operation [PurchaseReservedDBInstancesOffering](#) 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 you track them.

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

```
https://rds.us-east-1.amazonaws.com/
?Action=PurchaseReservedDBInstancesOffering
&ReservedDBInstanceId=MyReservation
&ReservedDBInstancesOfferingId=438012d3-4052-4cc7-b2e3-8d3372e0e706
&DBInstanceCount=10
&SignatureMethod=HmacSHA256
&SignatureVersion=4
&Version=2014-09-01
&X-Amz-Algorithm=AWS4-HMAC-SHA256
&X-Amz-Credential=AKIADQKE4SARGYLE/20140415/us-east-1/rds/aws4_request
&X-Amz-Date=20140415T232655Z
&X-Amz-SignedHeaders=content-type;host;user-agent;x-amz-content-sha256;x-amz-date
&X-Amz-Signature=c2ac761e8c8f54a8c0727f5a87ad0a766fbb0024510b9aa34ea6d1f7df52fb11
```

This call returns output similar to the following:

```
<PurchaseReservedDBInstancesOfferingResponse xmlns="http://rds.amazonaws.com/
doc/2014-10-31/">
  <PurchaseReservedDBInstancesOfferingResult>
    <ReservedDBInstance>
      <OfferingType>Partial Upfront</OfferingType>
      <CurrencyCode>USD</CurrencyCode>
      <RecurringCharges/>
      <ProductDescription>mysql</ProductDescription>
      <ReservedDBInstancesOfferingId>649fd0c8-cf6d-47a0-bfa6-060f8e75e95f</
      ReservedDBInstancesOfferingId>
      <MultiAZ>true</MultiAZ>
      <State>payment-pending</State>
      <ReservedDBInstanceId>MyReservation</ReservedDBInstanceId>
      <DBInstanceCount>10</DBInstanceCount>
      <StartTime>2011-12-18T23:24:56.577Z</StartTime>
      <Duration>31536000</Duration>
      <FixedPrice>123.0</FixedPrice>
      <UsagePrice>0.123</UsagePrice>
      <DBInstanceClass>db.m1.small</DBInstanceClass>
    </ReservedDBInstance>
  </PurchaseReservedDBInstancesOfferingResult>
  <ResponseMetadata>
    <RequestId>7f099901-29cf-11e1-bd06-6fe008f046c3</RequestId>
  </ResponseMetadata>
</PurchaseReservedDBInstancesOfferingResponse>
```

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

Example Get Your Reserved DB Instances

To get information about reserved DB instances for your AWS account, call the Amazon RDS API operation [DescribeReservedDBInstances](#).

```
https://rds.us-west-2.amazonaws.com/
?Action=DescribeReservedDBInstances
&SignatureMethod=HmacSHA256
&SignatureVersion=4
&Version=2014-09-01
&X-Amz-Algorithm=AWS4-HMAC-SHA256
&X-Amz-Credential=AKIADQKE4SARGYLE/20140420/us-west-2/rds/aws4_request
&X-Amz-Date=20140420T162211Z
&X-Amz-SignedHeaders=content-type;host;user-agent;x-amz-content-sha256;x-amz-date
&X-Amz-Signature=3312d17a4c43bcd209bc22a0778dd23e73f8434254abbd7ac53b89ade3dae88e
```

The API returns output similar to the following:

```
<DescribeReservedDBInstancesResponse xmlns="http://rds.amazonaws.com/doc/2014-10-31/">
<DescribeReservedDBInstancesResult>
  <ReservedDBInstances>
    <ReservedDBInstance>
      <OfferingType>Partial Upfront</OfferingType>
      <CurrencyCode>USD</CurrencyCode>
      <RecurringCharges/>
      <ProductDescription>mysql</ProductDescription>
      <ReservedDBInstancesOfferingId>649fd0c8-cf6d-47a0-bfa6-060f8e75e95f</
      ReservedDBInstancesOfferingId>
      <MultiAZ>false</MultiAZ>
      <State>payment-failed</State>
      <ReservedDBInstanceId>MyReservation</ReservedDBInstanceId>
      <DBInstanceCount>1</DBInstanceCount>
      <StartTime>2010-12-15T00:25:14.131Z</StartTime>
      <Duration>31536000</Duration>
      <FixedPrice>227.5</FixedPrice>
      <UsagePrice>0.046</UsagePrice>
      <DBInstanceClass>db.m1.small</DBInstanceClass>
    </ReservedDBInstance>
    <ReservedDBInstance>
      <OfferingType>Partial Upfront</OfferingType>
      <CurrencyCode>USD</CurrencyCode>
      <RecurringCharges/>
      <ProductDescription>mysql</ProductDescription>
      <ReservedDBInstancesOfferingId>649fd0c8-cf6d-47a0-bfa6-060f8e75e95f</
      ReservedDBInstancesOfferingId>
      <MultiAZ>false</MultiAZ>
      <State>payment-failed</State>
      <ReservedDBInstanceId>MyReservation</ReservedDBInstanceId>
      <DBInstanceCount>1</DBInstanceCount>
      <StartTime>2010-12-15T01:07:22.275Z</StartTime>
      <Duration>31536000</Duration>
      <FixedPrice>227.5</FixedPrice>
      <UsagePrice>0.046</UsagePrice>
      <DBInstanceClass>db.m1.small</DBInstanceClass>
    </ReservedDBInstance>
  </ReservedDBInstances>
</DescribeReservedDBInstancesResult>
<ResponseMetadata>
  <RequestId>23400d50-2978-11e1-9e6d-771388d6ed6b</RequestId>
</ResponseMetadata>
</DescribeReservedDBInstancesResponse>
```

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 represents the data for the DB cluster, copied across three Availability Zones as a single, virtual volume. By default, the DB cluster contains a primary writer DB instance and, optionally, up to 15 Aurora Replicas (reader DB instances). For more information about Aurora DB clusters, see [Amazon Aurora DB Clusters \(p. 3\)](#).

In the following topic, you can find out how to create an Aurora DB cluster. To get started, first see [DB Cluster Prerequisites \(p. 100\)](#).

For simple instructions on connecting to your Aurora DB cluster, see [Connecting to an Amazon Aurora DB Cluster \(p. 166\)](#).

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 \(p. 49\)](#).

The following are prerequisites to create a DB cluster.

VPC

You can only create an Amazon Aurora DB cluster 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 are using the AWS Management Console to create your Aurora DB cluster, you can have Amazon RDS automatically create a VPC for you. Or you can use an existing VPC or create a new VPC for your Aurora DB cluster. 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. For more information, see [How to Create a VPC for Use with Amazon Aurora \(p. 239\)](#). For information on VPCs, see [Amazon Virtual Private Cloud VPCs](#) and [Amazon Aurora \(p. 239\)](#).

Note

You can communicate with an EC2 instance that is not in a VPC and an Amazon Aurora DB cluster using ClassicLink. For more information, see [A DB Instance in a VPC Accessed by an EC2 Instance Not in a VPC \(p. 248\)](#).

If you don't have a default VPC or you haven't created a VPC, you can have Amazon RDS automatically create a VPC for you when you create an Aurora DB cluster using the console. Otherwise, you must do the following:

- Create a VPC with at least one subnet in each of at least two of the Availability Zones in the AWS Region where you want to deploy your DB cluster. For more information, see [How to Create a VPC for Use with Amazon Aurora \(p. 239\)](#).
- Specify a VPC security group that authorizes connections to your Aurora DB cluster. For more information, see [Working with a DB Instance in a VPC \(p. 250\)](#).
- Specify an RDS DB subnet group that defines at least two subnets in the VPC that can be used by the Aurora DB cluster. For more information, see [Working with DB Subnet Groups \(p. 250\)](#).

Additional Prerequisites

If you are connecting to AWS using AWS Identity and Access Management (IAM) credentials, your IAM account must have IAM policies that grant the permissions required to perform Amazon RDS operations. For more information, see [Identity and Access Management in Amazon Aurora \(p. 191\)](#).

If you are using an IAM account to access the Amazon RDS console, you must first sign on to the AWS Management Console with your IAM account. Then go to the Amazon RDS console at <https://console.aws.amazon.com/rds/>.

If you want to tailor the configuration parameters for your DB cluster, you must specify a DB cluster parameter group and DB parameter group with the required parameter settings. For information about creating or modifying a DB cluster parameter group or DB parameter group, see [Working with DB Parameter Groups and DB Cluster Parameter Groups \(p. 286\)](#).

You must determine the TCP/IP port number to specify for your DB cluster. The firewalls at some companies block connections to the default ports (3306 for MySQL, 5432 for PostgreSQL) for Aurora. If your company firewall blocks the default port, choose another port for your DB cluster. All instances in a DB cluster use the same port.

Creating a DB Cluster

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

Note

If you are using the console, a new console interface is available for database creation. Choose either the **New Console** or the **Original Console** instructions based on the console that you are using. The **New Console** instructions are open by default.

New Console

You can create a DB instance running MySQL with the AWS Management Console with **Easy create** enabled or not enabled. With **Easy create** enabled, you specify only the DB engine type, DB instance size, and DB instance identifier. **Easy Create** uses the default setting for other configuration options. With **Easy create** not enabled, you specify more configuration options when you create a database, including ones for availability, security, backups, and maintenance.

Note

For this example, **Standard Create** is enabled, and **Easy Create** isn't enabled. For information about creating an Aurora MySQL DB cluster with **Easy create** enabled, see [Getting Started with Amazon Aurora \(p. 54\)](#).

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 \(p. 80\)](#).
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**.

Create database

Choose a database creation method

- ## Standard Create

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

- ## Easy Create

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

Engine options

Engine type Info

- Amazon Aurora



- MySQL



- MariaDB



- ## ○ PostgreSQL



- Oracle

ORACLE®

- Microsoft SC



Edition

- Amazon Aurora with MySQL compatibility
 - Amazon Aurora with PostgreSQL compatibility

Version Info

7. In **Edition**, choose one of the following:

- Amazon Aurora with MySQL compatibility

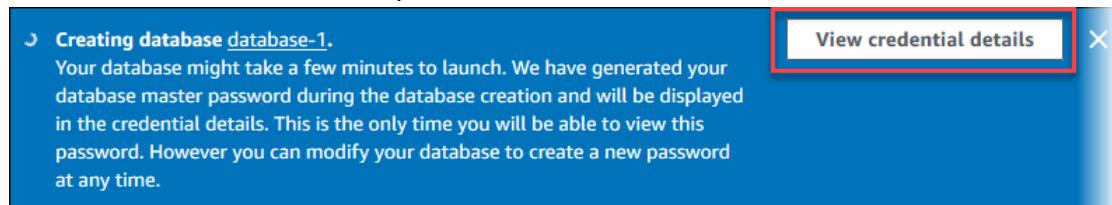
- **Amazon Aurora with PostgreSQL compatibility**
8. If you chose **Amazon Aurora with MySQL compatibility**, choose one of the following in **Database features**:
- **One writer and multiple readers**
For more information, see [Amazon Aurora DB Clusters \(p. 3\)](#).
 - **Serverless**
For more information, see [Using Amazon Aurora Serverless \(p. 116\)](#).
9. In **Templates**, choose the template that matches your use case.
10. 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.

11. For the remaining sections, specify your DB cluster settings. For information about each setting, see [Settings for Aurora DB Clusters \(p. 110\)](#).
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 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 \(p. 264\)](#).

13. 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. 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.

The screenshot shows the 'Databases' section of the Amazon RDS console. There are two entries in the table:

DB identifier	Role	Engine	Class	Status	CPU	Current activity
database-1	Cluster	Aurora MySQL	-	Creating		
database-1-instance-1	Reader	Aurora MySQL	db.r5.large	Creating		0 Sessions

The 'Status' column for both rows is circled in red.

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 **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 \(p. 446\)](#).

The screenshot shows the 'database-1' DB cluster details page. On the 'Connectivity & security' tab, the 'Endpoints' section lists two endpoints:

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

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

On the **Connectivity & security** tab, note the port and the endpoint of the writer DB instance. Use the endpoint and port of the cluster in your JDBC and ODBC connection strings for any application that performs write or read operations.

Original Console

To create an Aurora DB cluster using the AWS Management Console

1. Sign in to the AWS Management Console and open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. In the top-right corner of the AWS Management Console, choose the AWS Region in which you want to create the Aurora DB cluster.
3. In the navigation pane, choose **Databases**.
If the navigation pane is closed, choose the menu icon at the top left to open it.
4. Choose **Create database** to open the **Select engine** page.
5. On the **Select engine** page, choose an edition of Aurora. Choose either MySQL 5.6-compatible, MySQL 5.7-compatible, or PostgreSQL-compatible.

Select engine

Engine options

Amazon Aurora
Amazon Aurora

MySQL


MariaDB


PostgreSQL


Oracle
ORACLE

Microsoft SQL Server


Amazon Aurora
Amazon Aurora is a MySQL- and PostgreSQL-compatible enterprise-class database, starting at <\$1/day.

- Up to 5 times the throughput of MySQL and 3 times the throughput of PostgreSQL
- Up to 64TB of auto-scaling SSD storage
- 6-way replication across three Availability Zones
- Up to 15 Read Replicas with sub-10ms replica lag
- Automatic monitoring and failover in less than 30 seconds

Edition
 MySQL 5.6-compatible
 MySQL 5.7-compatible
 PostgreSQL-compatible

Only enable options eligible for RDS Free Usage Tier [info](#)

[Cancel](#) [Next](#)

6. Choose **Next**.
7. On the **Specify DB details** page, specify your DB instance information. For information about each setting, see [Settings for Aurora DB Clusters \(p. 110\)](#).

A typical **Specify DB details** page looks like the following.

Specify DB details

Instance specifications

Estimate your monthly costs for the DB Instance using the [AWS Simple Monthly Calculator](#).

DB engine
Aurora - compatible with MySQL 5.7.12

DB instance class [info](#)

Multi-AZ deployment [info](#)
 Create Replica in Different Zone
 No

Settings

DB instance identifier [info](#)
Specify a name that is unique for all DB instances owned by your AWS account in the current region.

DB instance identifier is case insensitive, but stored as all lower-case, as in "mydbinstance".

Master username [info](#)
Specify an alphanumeric string that defines the login ID for the master user.

Master Username must start with a letter.

Master password [info](#) Confirm password [info](#)

Master Password must be at least eight characters long, as in "mypassword".

[Cancel](#) [Previous](#) [Next](#)

8. Confirm your master password and choose **Next**.
9. On the **Configure advanced settings** page, you can customize additional settings for your Aurora DB cluster. For information about each setting, see [Settings for Aurora DB Clusters \(p. 110\)](#).
10. Choose **Create database** to create your Aurora DB cluster, and then choose **Close**.

On the Amazon RDS console, the new DB cluster appears in the list of DB clusters. The DB cluster will have a status of **creating** until the DB cluster is created and ready for use. When the state

changes to available, you can connect to the writer instance for your DB cluster. Depending on the DB cluster class and store allocated, it can take several minutes for the new cluster to be available.

To view the newly created cluster, choose **Databases** from the navigation pane in the Amazon RDS console and choose the DB cluster to show the DB cluster details. For more information, see [Viewing an Amazon Aurora DB Cluster \(p. 446\)](#).

The screenshot shows the AWS RDS console for a DB cluster named "gs-db-cluster1".

Related:

- DB identifier: gs-db-cluster1 (selected)
- Role: Regional
- Engine: Aurora MySQL
- Instances:
 - gs-db-instance1 (Writer)
 - gs-db-instance1-us-east-2b (Reader)

Connectivity & security:

- Endpoints (2):
 - gs-db-cluster1.rds.amazonaws.com (Available, Writer)
 - gs-db-cluster1.us-east-2.rds.amazonaws.com (Available, Reader)
- Buttons: Edit, Delete, Create

Note the ports and the endpoints of the cluster. Use the endpoint and port of the writer DB cluster in your JDBC and ODBC connection strings for any application that performs write or read operations.

AWS CLI

Note

Before you can create an Aurora DB cluster using the AWS CLI, you must fulfill the required prerequisites, such as creating a VPC and an RDS DB subnet group. For more information, see [DB Cluster Prerequisites \(p. 100\)](#).

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 5.7 DB cluster or DB instance, you must specify `aurora-mysql` for the `--engine` option.
- When you create an Aurora MySQL 5.6 DB cluster or DB instance, you must 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 5.7-compatible DB cluster named `sample-cluster`.

For Linux, OS X, 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, OS X, 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 DB instance named `sample-instance`.

For Linux, OS X, or Unix:

```
aws rds create-db-instance --db-instance-identifier sample-instance \
    --db-cluster-identifier sample-cluster --engine aurora-mysql --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-mysql --db-instance-class
db.r4.large
```

The following command creates a new MySQL 5.6-compatible DB instance named `sample-instance`.

For Linux, OS X, or Unix:

```
aws rds create-db-instance --db-instance-identifier sample-instance \
    --db-cluster-identifier sample-cluster --engine aurora --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 --db-instance-class
db.r4.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, OS X, 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, OS X, 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 \(p. 100\)](#).

Identify the DB subnet group and VPC security group ID for your new DB cluster, and then call the [CreateDBInstance](#) 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.

For This Option	Do This
Availability zone	Determine if you want to specify a particular Availability Zone. For more information about Availability Zones, see Choosing the Regions and Availability Zones (p. 80) .
Auto minor version upgrade	<p>Choose Enable auto minor version upgrade if you want to enable your Aurora DB cluster to receive preferred minor version upgrades to the DB engine automatically when they become available.</p> <p>The Auto minor version upgrade setting only applies to Aurora PostgreSQL DB clusters.</p> <p>For more information about engine updates for Aurora PostgreSQL, see Database Engine Updates for Amazon Aurora PostgreSQL (p. 970).</p> <p>For more information about engine updates for Aurora MySQL, see Database Engine Updates for Amazon Aurora MySQL (p. 794).</p>
Backtrack	Applies only to Aurora MySQL. Choose Enable Backtrack to enable backtracking or Disable Backtrack 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 Backtracking an Aurora DB Cluster (p. 625) .
Copy tags to snapshots	<p>Choose this option to copy any DB instance tags to a DB snapshot when you create a snapshot.</p> <p>For more information, see Tagging Amazon RDS Resources (p. 420).</p>
Database authentication	<p>The database authentication option you want to use.</p> <p>Choose Password authentication to authenticate database users with database passwords only.</p> <p>Choose Password and IAM DB authentication to authenticate database users with database passwords and user credentials through IAM users and roles. For more information, see IAM Database Authentication (p. 207).</p>
Database port	Specify the port for applications and utilities to use to access the database. Aurora MySQL DB clusters default to the default MySQL port, 3306, 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.
DB cluster identifier	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 This Option	Do This
	<p>For information on the cluster endpoint, see Amazon Aurora Connection Management (p. 4).</p> <p>The DB cluster identifier has the following constraints:</p> <ul style="list-style-type: none"> • It must contain from 1 to 63 alphanumeric characters or hyphens. • Its first character must be a letter. • It cannot end with a hyphen or contain two consecutive hyphens. • It must be unique for all DB clusters per AWS account, per AWS Region.
DB cluster parameter group	Choose a DB cluster parameter group. Aurora has a default DB cluster parameter group you can use, or you can create your own DB cluster parameter group. For more information about DB cluster parameter groups, see Working with DB Parameter Groups and DB Cluster Parameter Groups (p. 286) .
DB engine version	Applies only to the provisioned capacity type. Choose the version number of your DB engine.
DB instance class	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 Choosing the DB Instance Class (p. 76) .
DB instance identifier	<p>Enter a name for the primary instance in your DB cluster. This identifier is used in the endpoint address for the primary instance of your DB cluster.</p> <p>The DB instance identifier has the following constraints:</p> <ul style="list-style-type: none"> • It must contain from 1 to 63 alphanumeric characters or hyphens. • Its first character must be a letter. • It 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 parameter group	Choose a parameter group. Aurora has a default parameter group you can use, or you can create your own parameter group. For more information about parameter groups, see Working with DB Parameter Groups and DB Cluster Parameter Groups (p. 286) .
Enable deletion protection	Choose Enable deletion protection to prevent your DB cluster from being deleted. If you create a production DB cluster with the console, deletion protection is enabled by default.

For This Option	Do This
Enable encryption	Choose Enable encryption to enable encryption at rest for this DB cluster. For more information, see Encrypting Amazon Aurora Resources (p. 175) .
Enable Enhanced Monitoring	Choose Enable enhanced monitoring to enable gathering metrics in real time for the operating system that your DB cluster runs on. For more information, see Enhanced Monitoring (p. 469) .
Enable Performance Insights	Choose Enable Performance Insights to enable Amazon RDS Performance Insights. For more information, see Using Amazon RDS Performance Insights (p. 476) .
Failover priority	Choose a failover priority for the instance. If you don't choose a value, the default is tier-1 . This priority determines the order in which Aurora Replicas are promoted when recovering from a primary instance failure. For more information, see Fault Tolerance for an Aurora DB Cluster (p. 380) .
Granularity	Only available if Enhanced Monitoring is set to Enable enhanced monitoring . Set the interval, in seconds, between when metrics are collected for your DB cluster.
Initial database name	Type a name for your default database of up to 64 alpha-numeric characters. If you don't provide a name, Amazon RDS doesn't create a database on the DB cluster you are creating. To create additional databases, connect to the DB cluster and use the SQL command <code>CREATE DATABASE</code> . For more information about connecting to the DB cluster, see Connecting to an Amazon Aurora DB Cluster (p. 166) .
Log exports	Choose the types of MySQL or PostgreSQL database log files to generate. For more information, see MySQL Database Log Files (p. 567) and PostgreSQL Database Log Files (p. 572) .
Maintenance window	Choose Select window and specify the weekly time range during which system maintenance can occur. Or choose No preference for Amazon RDS to assign a period randomly.
Master key	Only available if Encryption is set to Enable encryption . Choose the master key to use for encrypting this DB cluster. For more information, see Encrypting Amazon Aurora Resources (p. 175) .
Option group	Aurora has a default option group.
Master password	Enter a password that contains from 8 to 41 printable ASCII characters (excluding /, ", and @) for your master user password.
Master username	Enter a name using alphanumeric characters to use as the master user name to log on to your DB cluster.

For This Option	Do This
Monitoring Role	Only available if Enhanced Monitoring is set to Enable enhanced monitoring . Choose the IAM role that you created to permit Amazon RDS to communicate with Amazon CloudWatch Logs for you, or choose Default to have RDS create a role for you named <code>rds-monitoring-role</code> . For more information, see Enhanced Monitoring (p. 469) .
Multi-AZ deployment	Applies only to the provisioned capacity type. Determine if you want to create Aurora Replicas in other Availability Zones for failover support. If you choose Create Replica in Different Zone , then Amazon RDS creates an Aurora Replica for you in your DB cluster in a different Availability Zone than the primary instance for your DB cluster. For more information about multiple Availability Zones, see Choosing the Regions and Availability Zones (p. 80) .
Option group	Aurora has a default option group.
Performance Insights	Doesn't apply to MySQL 5.6. Choose Enable Performance Insights if you want to use Amazon RDS Performance Insights to monitor your Amazon Aurora DB cluster load. For more information about Performance Insights, see Using Amazon RDS Performance Insights (p. 476) .
Publicly accessible	Choose Yes to give the DB cluster a public IP address; otherwise, choose No . The instances in your DB cluster can be a mix of both public and private DB instances. For more information about hiding instances from public access, see Hiding a DB Instance in a VPC from the Internet (p. 251) .
Retention period	Choose the length of time, from 1 to 35 days, that Aurora retains backup copies of the database. Backup copies can be used for point-in-time restores (PITR) of your database down to the second.
Subnet group	Choose the DB subnet group to use for the DB cluster. For more information, see DB Cluster Prerequisites (p. 100) .
Select the log types to publish to Amazon CloudWatch Logs	Applies only to Aurora MySQL. In the Log exports section, choose the logs that you want to start publishing to Amazon CloudWatch Logs. For more about publishing to CloudWatch Logs, see Publishing Amazon Aurora MySQL Logs to Amazon CloudWatch Logs (p. 759) .
Virtual Private Cloud (VPC)	Choose the VPC to host the DB cluster. Choose Create a New VPC to have Amazon RDS create a VPC for you. For more information, see DB Cluster Prerequisites (p. 100) .

For This Option	Do This
VPC security group	<p>Choose Create new to have Amazon RDS create a VPC security group for you. Or choose Choose existing and specify one or more VPC security groups to secure network access to the DB cluster.</p> <p>When you choose Create new in the RDS console, a new security group is created with an inbound rule that allows access to the DB instance from the IP address detected in your browser.</p> <p>For more information, see DB Cluster Prerequisites (p. 100).</p>

Using Amazon Aurora Serverless

Amazon Aurora Serverless is an on-demand, autoscaling configuration for Amazon Aurora. An *Aurora Serverless DB cluster* is a DB cluster that automatically starts up, shuts down, and scales up or down its compute capacity based on your application's needs. Aurora Serverless provides a relatively simple, cost-effective option for infrequent, intermittent, or unpredictable workloads. It can provide this because it automatically starts up, scales compute capacity to match your application's usage, and shuts down when it's not in use.

Note

A non-Serverless DB cluster for Aurora is called a *provisioned DB cluster*. Aurora Serverless clusters and provisioned clusters both have the same kind of high-capacity, distributed, and highly available storage volume.

Topics

- [Advantages of Aurora Serverless \(p. 116\)](#)
- [Use Cases for Aurora Serverless \(p. 116\)](#)
- [Limitations of Aurora Serverless \(p. 117\)](#)
- [Using TLS/SSL with Aurora Serverless \(p. 118\)](#)
- [How Aurora Serverless Works \(p. 119\)](#)
- [Creating an Aurora Serverless DB Cluster \(p. 124\)](#)
- [Restoring an Aurora Serverless DB Cluster \(p. 129\)](#)
- [Modifying an Aurora Serverless DB Cluster \(p. 132\)](#)
- [Setting the Capacity of an Aurora Serverless DB Cluster \(p. 133\)](#)
- [Viewing Aurora Serverless DB Clusters \(p. 135\)](#)
- [Using the Data API for Aurora Serverless \(p. 139\)](#)
- [Logging Data API Calls with AWS CloudTrail \(p. 159\)](#)
- [Using the Query Editor for Aurora Serverless \(p. 161\)](#)

Advantages of Aurora Serverless

Aurora Serverless provides the following advantages:

Simpler

Aurora Serverless removes much of the complexity of managing DB instances and capacity.

Scalable

Aurora Serverless seamlessly scales compute and memory capacity as needed, with no disruption to client connections.

Cost-effective

When you use Aurora Serverless, you pay for only the database resources that you consume, on a per-second basis.

Highly available storage

Aurora Serverless uses the same fault-tolerant, distributed storage system with six-way replication as Aurora to protect against data loss.

Use Cases for Aurora Serverless

Aurora Serverless 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, you pay for only the database resources that you consume on a per-second basis.

New applications

You are deploying a new application and are unsure about which instance size you need. With Aurora Serverless, 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, you no longer need to provision to either peak or average capacity.

Unpredictable workloads

You're running workloads where there is database usage throughout the day, but also peaks of activity that are hard to predict. An example is a traffic site that sees a surge of activity when it starts raining. With Aurora Serverless, 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, your database automatically shuts down when it's not in use.

Multi-tenant applications

With Aurora Serverless, you don't have to individually manage database capacity for each application in your fleet. Aurora Serverless manages individual database capacity for you.

Limitations of Aurora Serverless

The following limitations apply to Aurora Serverless:

- Aurora Serverless is only available for the following:
 - Aurora with MySQL version 5.6 compatibility
 - Aurora with PostgreSQL version 10.7 compatibility
- The port number for connections must be:
 - 3306 for Aurora MySQL
 - 5432 for Aurora PostgreSQL
- You can't give an Aurora Serverless DB cluster a public IP address. You can access an Aurora Serverless DB cluster only from within a virtual private cloud (VPC) based on the Amazon VPC service.
- Each Aurora Serverless DB cluster requires two AWS PrivateLink endpoints. If you reach the limit for PrivateLink endpoints within your VPC, you can't create any more Aurora Serverless clusters in that VPC. For information about checking and changing the limits on endpoints within a VPC, see [Amazon VPC Limits](#).
- A DB subnet group used by Aurora Serverless can't have more than one subnet in the same Availability Zone.
- Changes to a subnet group used by an Aurora Serverless DB cluster are not applied to the cluster.
- Aurora Serverless doesn't support the following features:
 - [Loading data from an Amazon S3 bucket \(p. 739\)](#)
 - [Saving data to an Amazon S3 bucket \(p. 747\)](#)

- [Invoking an AWS Lambda function with an Aurora MySQL native function \(p. 753\)](#)
- [Aurora Replicas \(p. 672\)](#)
- [Backtrack \(p. 625\)](#)
- [Multi-master clusters \(p. 702\)](#)
- [Database cloning \(p. 335\)](#)
- [IAM database authentication \(p. 207\)](#)
- [Restoring a snapshot from a MySQL DB instance \(p. 607\)](#)
- [Amazon RDS Performance Insights \(p. 476\)](#)

Note

You can access an Aurora Serverless DB cluster from AWS Lambda. 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

You can connect to Aurora Serverless clusters using the Transport Layer Security/Secure Sockets Layer (TLS/SSL) protocol. To do so, you use the same general procedure as described in [Connecting to an Amazon Aurora DB Cluster \(p. 166\)](#). You use certificates from the AWS Certificate Manager (ACM). For more information, see the [AWS Certificate Manager User Guide](#).

Aurora Serverless can ensure that your session uses TLS between your client and the Aurora Serverless VPC endpoint. To have Aurora Serverless do so, specify the requirement on the client side with the `--ssl-mode` parameter. SSL session variables are not set for SSL connections to an Aurora Serverless DB cluster.

Aurora Serverless supports TLS protocol version 1.0, 1.1, and 1.2. However, you don't need to configure an Aurora Serverless database for TLS. In particular, don't use the `REQUIRE` clause on your database user privileges for SSL. Doing so prevents that user from connecting.

By default, client programs establish an encrypted connection with Aurora Serverless, with further control available through the `--ssl-mode` option. From the client side, Aurora Serverless supports all SSL modes.

Note

TLS support for Aurora Serverless clusters currently isn't available in the China (Beijing) AWS Region.

For the `mysql` and `psql` 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 `mysql` or `psql` client with `--ssl-mode VERIFY_CA` or `VERIFY_IDENTITY`, specify the `--ssl-ca` option pointing to a CA in .pem format. For a .pem file that you can use, download the [Amazon Root CA 1 trust store](#) from Amazon Trust Services.

Aurora Serverless uses wildcard certificates. If you use the `mysql` client to connect with SSL mode `VERIFY_IDENTITY`, currently you must use the MySQL 8.0-compatible `mysql` command.

How Aurora Serverless Works

When you work with Amazon Aurora without Aurora Serverless (provisioned DB clusters), you can choose your DB instance class size and create Aurora Replicas to increase read throughput. If your workload changes, you can modify the DB instance class size and change the number of Aurora Replicas. This model works well when the database workload is predictable, because you can adjust capacity manually based on the expected workload.

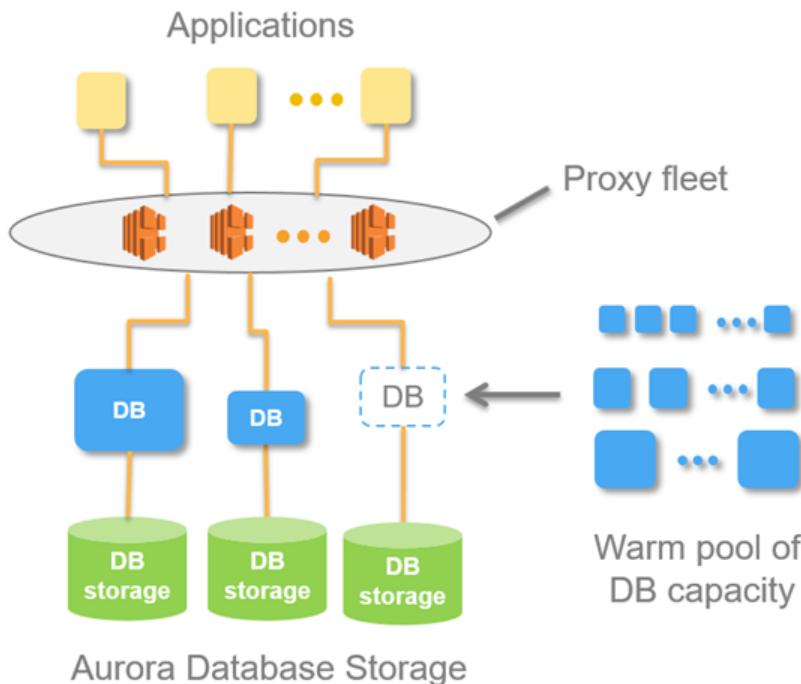
However, in some environments, workloads can be intermittent and unpredictable. There can be periods of heavy workloads that might last only a few minutes or hours, and also long periods of light activity, or even no activity. Some examples are retail websites with intermittent sales events, reporting databases that produce reports when needed, development and testing environments, and new applications with uncertain requirements. In these cases and many others, it can be difficult to configure the correct capacity at the right times. It can also result in higher costs when you pay for capacity that isn't used.

With Aurora Serverless, you can create a database endpoint without specifying the DB instance class size. You set the minimum and maximum capacity. With Aurora Serverless, the database endpoint connects to a *proxy fleet* that routes the workload to a fleet of resources that are automatically scaled. Because of the proxy fleet, connections are continuous as Aurora Serverless scales the resources automatically based on the minimum and maximum capacity specifications. Database client applications don't need to change to use the proxy fleet. Aurora Serverless manages the connections automatically. Scaling is rapid because it uses a pool of "warm" resources that are always ready to service requests. Storage and processing are separate, so you can scale down to zero processing and pay only for storage.

Aurora Serverless introduces a new `serverless` DB engine mode for Aurora DB clusters. Non-Serverless DB clusters use the provisioned DB engine mode.

Aurora Serverless Architecture

The following image provides an overview of the Aurora Serverless architecture.



Instead of provisioning and managing database servers, you specify Aurora capacity units (ACUs). Each ACU is a combination of processing and memory capacity. Database storage automatically scales from 10 GiB to 64 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 automatically creates scaling rules for thresholds for CPU utilization, connections, and available memory.

Aurora Serverless manages the warm pool of resources in an AWS Region to minimize scaling time. When Aurora Serverless adds new resources to the Aurora DB cluster, it uses the proxy fleet to switch active client connections to the new resources. At any specific time, you are only charged for the ACUs that are being actively used in your Aurora DB cluster.

Autoscaling for Aurora Serverless

The capacity allocated to your Aurora Serverless DB cluster seamlessly scales up and down based on the load (the CPU utilization and number of connections) generated by your client application. It also scales to zero capacity when there are no connections for a 5-minute period.

Aurora Serverless scales up when capacity constraints are seen in CPU, connections, or memory. It also scales up when it detects performance issues that can be resolved by scaling up.

After scaling up, the cooldown period for scaling down is 15 minutes. After scaling down, the cooldown period for scaling down again is 310 seconds.

Note

There is no cooldown period for scaling up. Aurora Serverless can scale up whenever necessary, including immediately after scaling up or scaling down.

A *scaling point* is a point in time at which the database can safely initiate the scaling operation. Under the following conditions, Aurora Serverless might not be able to find a scaling point:

- Long-running queries or transactions are in progress
- Temporary tables or table locks are in use

In these cases, Aurora Serverless continues to try to find a scaling point so that it can initiate the scaling operation. It does this for as long as it determines that the DB cluster should be scaled.

You can see scaling events in the details for a DB cluster in the AWS Management Console. You can also monitor the current capacity allocated to the DB cluster by using the `ServerlessDatabaseCapacity` metric for Amazon CloudWatch.

During autoscaling, Aurora Serverless resets the `EngineUptime` metric. The reset metric value doesn't indicate any issues with seamless scaling and doesn't mean that any connections were dropped. For more information about metrics, see [Monitoring Amazon Aurora DB Cluster Metrics \(p. 457\)](#).

Automatic Pause and Resume for Aurora Serverless

You can choose to pause your Aurora Serverless 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. The default is five minutes. You can also disable pausing the DB cluster.

When the DB cluster is paused, no compute or memory activity occurs, and you are charged only for storage. If database connections are requested when an Aurora Serverless DB cluster is paused, the DB cluster automatically resumes and services the connection requests.

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, the DB cluster is restored when there is a request to connect to it.

Timeout Action for Capacity Changes

You can change the capacity of an Aurora Serverless DB cluster. When you change the capacity, Aurora Serverless tries to find a scaling point for the change. If Aurora Serverless can't find a scaling point, it times out. You can specify one of the following actions to take when a capacity change times out:

- **Force the capacity change** – Set the capacity to the specified value as soon as possible.
- **Roll back the capacity change** – Cancel the capacity change.

Important

If you force the capacity change, connections that prevent Aurora Serverless from finding a scaling point might be dropped.

For information about changing the capacity, see [Modifying an Aurora Serverless DB Cluster \(p. 132\)](#).

Aurora Serverless and Parameter Groups

Parameter groups work differently for Serverless DB clusters than for provisioned DB clusters. In particular, the DB instances in an Aurora Serverless cluster only have associated DB cluster parameter groups, not DB parameter groups. Serverless clusters rely on DB cluster parameter groups because DB instances are not permanently associated with Aurora Serverless clusters. Aurora adds and removes DB instances automatically as needed.

To customize configuration settings for an Aurora Serverless cluster, you can define your own DB cluster parameter group and modify the parameters it contains. You can modify both cluster-level parameters, and parameters that apply at the instance level in other kinds of Aurora clusters. However,

when you modify a DB cluster parameter group that's associated with an Aurora Serverless DB cluster, modifications apply differently than for other DB cluster parameter groups.

When you save changes to a DB cluster parameter group for an Aurora Serverless DB cluster, changes are applied immediately. In doing so, Aurora Serverless ignores the following settings:

- When modifying the DB cluster as a whole, Aurora Serverless ignores the following:
 - The **Apply Immediately** setting in the AWS Management Console
 - The `--apply-immediately|--no-apply-immediately` option in the AWS CLI command `modify-db-cluster`
 - The `ApplyImmediately` parameter in the RDS API operation `ModifyDBCluster`
- When modifying parameters, Aurora Serverless ignores the `ApplyMethod` value in the parameter list in the AWS CLI commands `modify-db-cluster-parameter-group` and `reset-db-cluster-parameter-group`.
- When modifying parameters, Aurora Serverless ignores the `ApplyMethod` value in the parameter list in the RDS API operations `ModifyDBClusterParameterGroup` and `ResetDBClusterParameterGroup`.

To apply a change to a DB cluster parameter group, Aurora Serverless starts a seamless scale with the current capacity if the DB cluster is active. It resumes the DB cluster if it's paused.

Important

Aurora performs the seamless scaling operation for a parameter group change with the `force-apply-capacity-change` option. If a scaling point can't be found, connections that prevent Aurora Serverless from finding a scaling point might be dropped.

To view the supported engine mode for cluster-level parameters, run the `describe-engine-default-cluster-parameters` command or the RDS API operation `DescribeEngineDefaultClusterParameters`. For example, the following Linux command extracts the names of parameters that you can set for Aurora MySQL Serverless clusters, from an `aurora5.6` default DB cluster parameter group.

```
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
```

the following Linux command extracts the names of parameters that you can set for Aurora PostgreSQL Serverless clusters, from a `aurora-postgresql10` default DB cluster parameter group.

```
aws rds describe-engine-default-cluster-parameters \
--db-parameter-group-family aurora-postgresql10 \
--query 'EngineDefaults.Parameters[*].{ParameterName:ParameterName, \
SupportedEngineModes:SupportedEngineModes} \
| [?contains(SupportedEngineModes, `serverless`) == `true`] \
| [*].{param:ParameterName}' \
--output text
```

The following Linux command extracts the names of parameters that you can set for Serverless clusters from a DB cluster parameter group that you created.

```
aws rds describe-db-cluster-parameters \
--db-cluster-parameter-group-name my_cluster_param_group_name \
--query 'Parameters[*].{ParameterName:ParameterName, \
SupportedEngineModes:SupportedEngineModes}'
```

```
| [?contains(SupportedEngineModes, `serverless`) == `true`]
| [*].{param:ParameterName}' \
--output text
```

For more information about parameter groups, see [Working with DB Parameter Groups and DB Cluster Parameter Groups \(p. 286\)](#).

With an Aurora MySQL Serverless DB cluster, you can modify only the following parameters. For all other configuration parameters, Aurora MySQL Serverless clusters use the default values.

- `character_set_server`.
- `collation_server`.
- `general_log`. This setting was formerly only in the DB instance parameter group.
- `innodb_file_format`. This setting was formerly only in the DB instance parameter group.
- `innodb_file_per_table`.
- `innodb_large_prefix`. This setting was formerly only in the DB instance parameter group.
- `innodb_lock_wait_timeout`. This setting was formerly only in the DB instance parameter group.
- `innodb_monitor_disable`. This setting was formerly only in the DB instance parameter group.
- `innodb_monitor_enable`. This setting was formerly only in the DB instance parameter group.
- `innodb_monitor_reset`. This setting was formerly only in the DB instance parameter group.
- `innodb_monitor_reset_all`. This setting was formerly only in the DB instance parameter group.
- `innodb_print_all_deadlocks`. This setting was formerly only in the DB instance parameter group.
- `lc_time_names`.
- `log_output`. This setting was formerly only in the DB instance parameter group. This setting has a default value of `FILE`. You can't change this value.
- `log_queries_not_using_indexes`. This setting was formerly only in the DB instance parameter group.
- `log_warnings`. This setting was formerly only in the DB instance parameter group.
- `long_query_time`. This setting was formerly only in the DB instance parameter group.
- `lower_case_table_names`.
- `net_read_timeout`. This setting was formerly only in the DB instance parameter group.
- `net_retry_count`. This setting was formerly only in the DB instance parameter group.
- `net_write_timeout`. This setting was formerly only in the DB instance parameter group.
- `server_audit_logging`.
- `server_audit_events`.
- `server_audit_excl_users`.
- `server_audit_incl_users`.
- `slow_query_log`. This setting was formerly only in the DB instance parameter group.
- `sql_mode`. This setting was formerly only in the DB instance parameter group.
- `time_zone`.
- `tx_isolation`. This setting was formerly only in the DB instance parameter group.

Aurora Serverless and Maintenance

Aurora Serverless performs regular maintenance so that your DB cluster has the latest features, fixes, and security updates. Aurora Serverless performs maintenance in a non-disruptive manner whenever

possible. If long-running transactions are in progress, or temporary tables or table locks are in use, maintenance might cause actively used connections to drop momentarily.

To apply maintenance, Aurora Serverless must find a scaling point. For more information about scaling points, see [Autoscaling for Aurora Serverless \(p. 120\)](#).

If there is maintenance required for an Aurora Serverless DB cluster, the DB cluster attempts to find a scaling point to apply the maintenance for one day. If after a day no scaling point can be found, Aurora Serverless creates a scaling event to notify you that it must scale to apply maintenance. The notification includes the amount of time you have to force the scaling of your DB cluster. After you get this notification, you can control the time of the scaling, and the associated maintenance is applied at that time. Aurora Serverless tries to apply maintenance seamlessly for the amount of time specified in the event. After that time period has elapsed, Aurora Serverless drops connections to apply maintenance.

Note

Maintenance windows don't apply to Aurora Serverless.

Aurora Serverless and Failover

The DB instance for an Aurora Serverless DB cluster currently is created in a single Availability Zone (AZ). If the DB instance or the AZ becomes unavailable, Aurora recreates the DB instance in a different AZ. We refer to this capability as automatic multi-AZ failover.

This failover mechanism takes longer than for an Aurora Provisioned cluster. The Aurora Serverless failover time is currently undefined because it depends on demand and capacity availability in other AZs within the given AWS Region.

Because Aurora separates computation capacity and storage, the storage volume for the cluster is spread across multiple AZs. Your data remains available even if outages affect the DB instance or the associated AZ.

Aurora Serverless and Snapshots

The cluster volume for an Aurora Serverless cluster is always encrypted. You can choose the encryption key, but not turn off encryption. To copy or share a snapshot of an Aurora Serverless cluster, you encrypt the snapshot using your own KMS key.

Creating an Aurora Serverless DB Cluster

When you create an Aurora Serverless DB cluster, you can set the minimum and maximum capacity for the 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:

- **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 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 \(p. 121\)](#).
- **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 create an Aurora Serverless DB cluster with the AWS Management Console, the AWS CLI, or the RDS API.

For general information about creating a DB cluster, see [Creating an Amazon Aurora DB Cluster \(p. 100\)](#).

Note

Currently, Aurora Serverless isn't available in all AWS Regions. For more information on Aurora Serverless, see [Pricing](#).

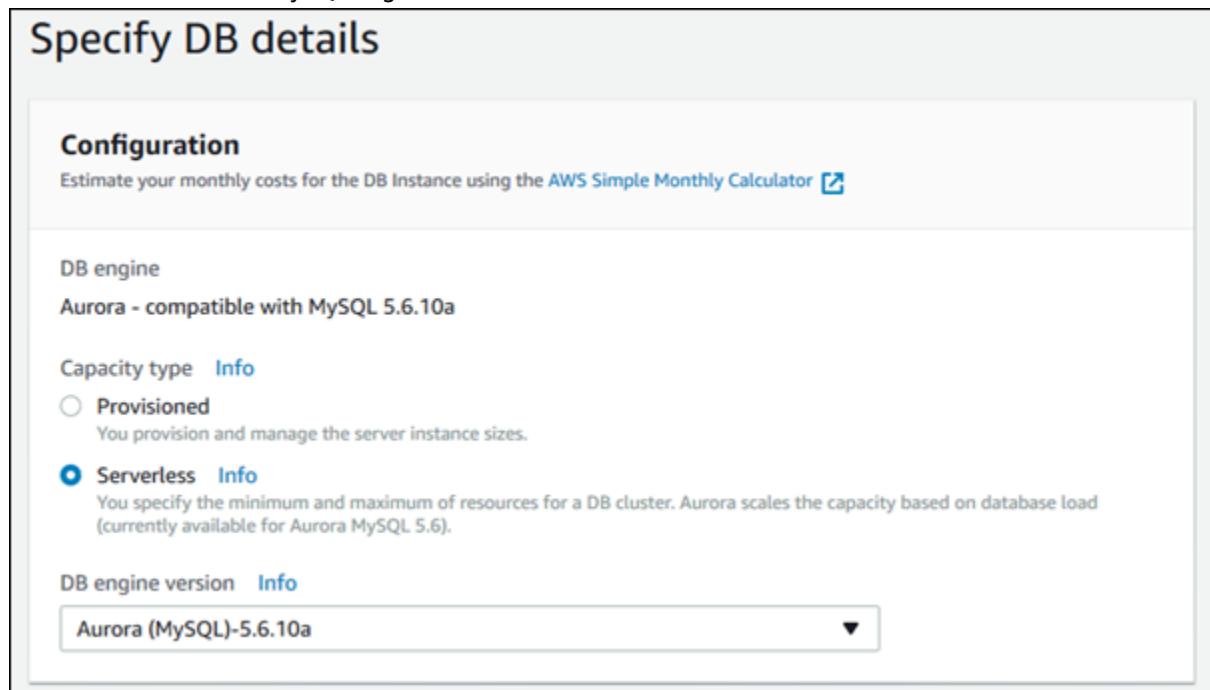
The cluster volume for an Aurora Serverless cluster is always encrypted. You can choose the encryption key, but not turn off encryption. Therefore, you can't perform operations that aren't allowed for encrypted snapshots. For example, you can't copy snapshots of Aurora Serverless clusters to a different AWS Region.

Console

To create a new Aurora Serverless DB cluster with the AWS Management Console, specify **Serverless** for **Capacity type** on the **Specify DB details** page.

Example for Aurora MySQL

The following image shows the **Specify DB details** page with **Serverless** chosen during DB cluster creation for the Aurora MySQL engine.



You can configure the scaling configuration of the Aurora Serverless DB cluster by adjusting values in the **Capacity settings** section on the **Configure advanced settings** page.

The following image shows the **Capacity settings** you can adjust for an Aurora MySQL Serverless DB cluster.

Capacity settings

Billing estimate is based on published prices. [Learn more](#)

Minimum Aurora capacity unit Info	Maximum Aurora capacity unit Info
<input type="text" value="1"/> 2GB RAM	<input type="text" value="64"/> 122GB RAM

▼ Additional scaling configuration

Force scaling the capacity to the specified values when the timeout is reached [Info](#)
Enable to force capacity scaling as soon as possible. Disable to cancel the capacity changes when a timeout is reached

Pause compute capacity after consecutive minutes of inactivity [Info](#)
You are only charged for database storage while the compute capacity is paused

00 hours 05 minutes 00 seconds

Max: 24 hours

You can also enable the Data API for your Aurora MySQL Serverless DB cluster. For more information, see [Using the Data API for Aurora Serverless \(p. 139\)](#).

Example for Aurora PostgreSQL

For the Aurora PostgreSQL engine, first choose the **DB engine version** for Aurora PostgreSQL that is compatible with PostgreSQL version 10.7. Then choose **Serverless** for the **Capacity type**.

Configuration

Estimate your monthly costs for the DB Instance using the [AWS Simple Monthly Calculator](#)

DB engine
Aurora PostgreSQL

Capacity type [Info](#)

Provisioned
You provision and manage the server instance sizes.

Provisioned with Aurora parallel query enabled [Info](#)
You provision and manage the server instance sizes, and Aurora improves the performance of analytic queries by pushing processing down to the Aurora storage layer (currently available for Aurora MySQL 5.6)

Serverless [Info](#)
You specify the minimum and maximum of resources for a DB cluster. Aurora scales the capacity based on database load.

DB engine version [Info](#)

Aurora PostgreSQL (compatible with PostgreSQL 10.7)

You can configure the scaling configuration of the Aurora Serverless DB cluster by adjusting values in the **Capacity settings** section on the [Configure advanced settings](#) page.

The following image shows the **Capacity settings** you can adjust for an Aurora PostgreSQL Serverless DB cluster.

The screenshot shows the 'Capacity settings' configuration page. It includes fields for 'Minimum Aurora capacity unit' (set to 8, 16GB RAM) and 'Maximum Aurora capacity unit' (set to 384, 768GB RAM). A section for 'Additional scaling configuration' is expanded, showing two options: 'Force scaling the capacity to the specified values when the timeout is reached' (unchecked) and 'Pause compute capacity after consecutive minutes of inactivity' (checked, set to 00 hours, 05 minutes, 00 seconds). A note at the bottom states 'Max: 24 hours'.

For more information on creating an Aurora DB cluster using the AWS Management Console, see [Creating an Amazon Aurora DB Cluster \(p. 100\)](#).

To connect to an Aurora Serverless DB cluster, use the database endpoint. For details, see the instructions in [Connecting to an Amazon Aurora DB Cluster \(p. 166\)](#).

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 \(p. 235\)](#).

AWS CLI

To create a new Aurora Serverless 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 command creates a new MySQL 5.6-compatible Serverless DB cluster. Valid capacity values for Aurora MySQL are 1, 2, 4, 8, 16, 32, 64, 128, and 256.

For Linux, OS X, or Unix:

```
aws rds create-db-cluster --db-cluster-identifier sample-cluster --engine aurora --engine-version 5.6.10a \
--engine-mode serverless --scaling-configuration
MinCapacity=4,MaxCapacity=32,SecondsUntilAutoPause=1000,AutoPause=true \
--master-username username --master-user-password password
```

For Windows:

```
aws rds create-db-cluster --db-cluster-identifier sample-cluster --engine aurora --engine-version 5.6.10a ^
--engine-mode serverless --scaling-configuration
MinCapacity=4,MaxCapacity=32,SecondsUntilAutoPause=1000,AutoPause=true ^
--master-username username --master-user-password password
```

Example for Aurora PostgreSQL

The following command creates a new PostgreSQL 10.7–compatible Serverless DB cluster. Valid capacity values for Aurora PostgreSQL are 2, 4, 8, 16, 32, 64, 192, and 384.

For Linux, OS X, or Unix:

```
aws rds create-db-cluster --db-cluster-identifier sample-cluster --engine aurora-postgresql
--engine-version 10.7 \
--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 10.7 ^
--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 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 DB Cluster

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 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 \(p. 121\)](#).
- **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 \(p. 385\)](#).

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.

Capacity settings

Billing estimate is based on published prices. [Learn more](#)

Minimum Aurora capacity unit [Info](#)

1

2GB RAM

Maximum Aurora capacity unit [Info](#)

64

122GB RAM

▼ Additional scaling configuration

Force scaling the capacity to the specified values when the timeout is reached [Info](#)

Enable to force capacity scaling as soon as possible. Disable to cancel the capacity changes when a timeout is reached

Pause compute capacity after consecutive minutes of inactivity [Info](#)

You are only charged for database storage while the compute capacity is paused

00

hours

05

minutes

00

seconds

Max: 24 hours

8. Choose **Restore DB Cluster**.

To connect to an Aurora Serverless DB cluster, use the database endpoint. For details, see the instructions in [Connecting to an Amazon Aurora DB Cluster \(p. 166\)](#).

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 \(p. 235\)](#).

AWS CLI

To configure an Aurora Serverless DB cluster when you restore from a DB cluster using the AWS CLI, run the [restore-db-cluster-from-snapshot](#) CLI 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. 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`. You restore to a new DB cluster named `mynewdbcluster`. To restore the DB cluster as an Aurora Serverless DB cluster, set the `--engine-mode` option to `serverless`. The example also specifies values for the `--scaling-configuration` option.

For Linux, OS X, 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 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 DB Cluster

After you configure an Aurora Serverless DB cluster, you can modify its scaling configuration with the AWS Management Console, the AWS CLI, or the RDS API.

You can set the minimum and maximum capacity for the DB cluster. Each capacity unit is equivalent to a specific compute and memory configuration. Aurora Serverless 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:

- **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 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 \(p. 121\)](#).
- **Pause after inactivity** – The amount of time with no database traffic to scale to zero processing capacity. When database traffic resumes, Aurora automatically resumes processing capacity and scales to handle the traffic.

Console

You can modify the scaling configuration of an Aurora DB cluster with the AWS Management Console.

To modify an Aurora Serverless 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 DB cluster that you want to modify.
4. For **Actions**, choose **Modify cluster**.
5. In the **Capacity settings** section, modify the scaling configuration.

Capacity settings
Billing estimate is based on published prices. [Learn more](#)

Minimum Aurora capacity unit Info <input type="text" value="1"/> 2GB RAM	Maximum Aurora capacity unit Info <input type="text" value="64"/> 122GB RAM
---	--

▼ Additional scaling configuration

Force scaling the capacity to the specified values when the timeout is reached [Info](#)
Enable to force capacity scaling as soon as possible. Disable to cancel the capacity changes when a timeout is reached

Pause compute capacity after consecutive minutes of inactivity [Info](#)
You are only charged for database storage while the compute capacity is paused
 hours minutes seconds
Max: 24 hours

6. Choose **Continue**.
7. Choose **Modify cluster**.

The change is applied immediately.

AWS CLI

To modify the scaling configuration of an Aurora Serverless 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 DB cluster named `sample-cluster`.

For Linux, OS X, or Unix:

```
aws rds modify-db-cluster --db-cluster-identifier sample-cluster \
--scaling-configuration
MinCapacity=8,MaxCapacity=64,SecondsUntilAutoPause=500,TimeoutAction='ForceApplyCapacityChange',AutoPa
```

For Windows:

```
aws rds modify-db-cluster --db-cluster-identifier sample-cluster ^
--scaling-configuration
MinCapacity=8,MaxCapacity=64,SecondsUntilAutoPause=500,TimeoutAction='ForceApplyCapacityChange',AutoPa
```

RDS API

You can modify the scaling configuration of an Aurora DB cluster with the `ModifyDBCluster` API operation. Specify the `ScalingConfiguration` parameter to configure the minimum capacity, maximum capacity, and automatic pause when there are no connections. Valid capacity values include the following:

- Aurora MySQL: 1, 2, 4, 8, 16, 32, 64, 128, and 256.
- Aurora PostgreSQL: 2, 4, 8, 16, 32, 64, 192, and 384.

Setting the Capacity of an Aurora Serverless DB Cluster

Set the capacity of an Aurora Serverless DB cluster to a specific value with the AWS Management Console, the AWS CLI, or the RDS API.

Aurora Serverless scales seamlessly based on the workload on the DB cluster. In some cases, the capacity might not scale fast enough to meet a sudden change in workload, such as a large number of new

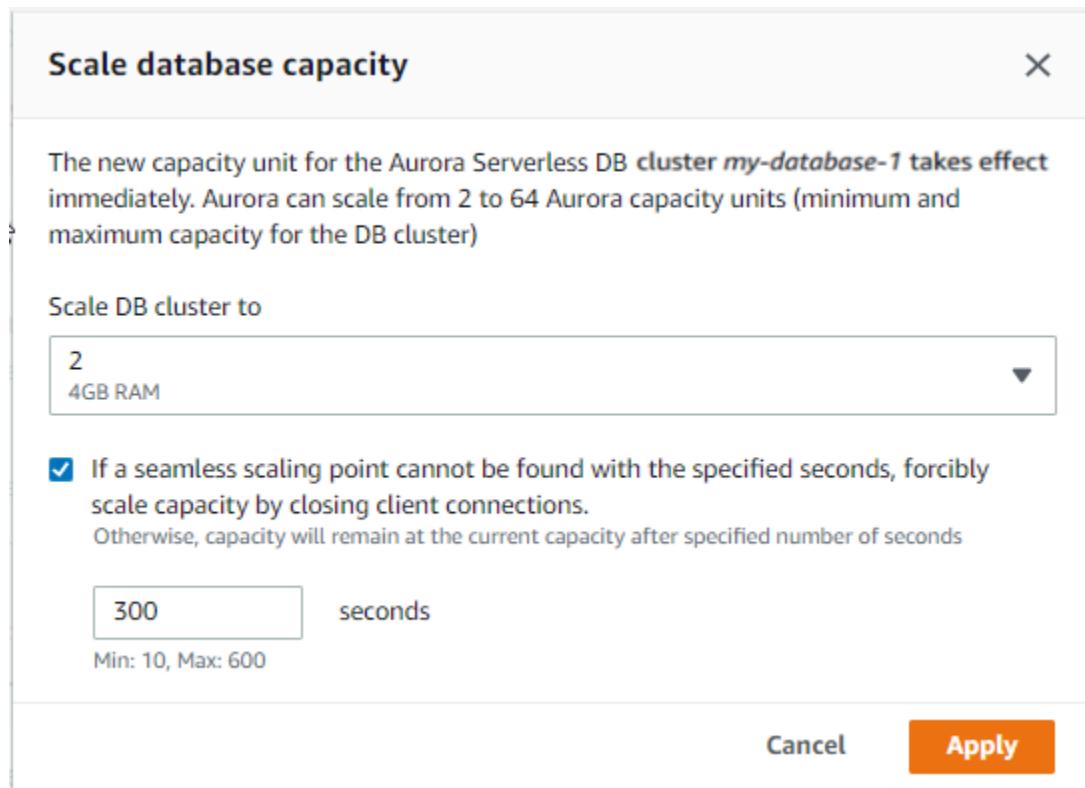
transactions. In these cases, you can set the capacity explicitly. After you set the capacity explicitly, Aurora Serverless can automatically scale the DB cluster. It does so based on the cooldown period for scaling down.

Console

You can set the capacity of an Aurora DB cluster with the AWS Management Console.

To modify an Aurora Serverless 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 DB cluster that you want to modify.
4. For **Actions**, choose **Set capacity**.
5. In the **Scale database capacity** window, set the capacity, the capacity that the DB cluster should scale to.



6. Enable or disable the option to forcibly scale capacity. If you enable it, specify the amount of time to find a scaling point. Aurora Serverless attempts to find a scaling point before timing out. If the timeout is reached, Aurora Serverless performs one of the following actions:
 - If checked, Aurora Serverless will force the scaling capacity change to proceed after the timeout.
 - If unchecked, Aurora Serverless will not perform the scaling capacity change after the timeout.

Important

When the option is enabled, connections that prevent Aurora Serverless from finding a scaling point might be dropped. For more information about scaling points and cooldown periods, see [Autoscaling for Aurora Serverless \(p. 120\)](#).

7. Choose **Apply**.

AWS CLI

To set the capacity of an Aurora Serverless 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 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 DB Clusters

After you create one or more Aurora Serverless 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 DB cluster is using. Each ACU is a combination of processing and memory capacity.

To view your Aurora Serverless 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 DB clusters.
3. In the navigation pane, choose **Databases**.

For each DB cluster, the DB cluster type is shown under **Role**. The Aurora Serverless DB clusters show **Serverless** for the type. You can view an Aurora Serverless 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	✓ /
my-database-1	Serverless	Aurora PostgreSQL	us-east-1	0 capacity units	✓ /
mysqls3	Instance	MySQL	us-east-1c	db.t3.medium	✓ /
tests3aurora	Regional	Aurora MySQL	us-east-1	2 instances	✓ /
tests3aurora	Writer	Aurora MySQL	us-east-1a	db.t3.medium	✓ /
tests3aurora-us-east-1b	Reader	Aurora MySQL	us-east-1b	db.t3.medium	✓ /

4. Choose the name of an Aurora Serverless DB cluster to display its details.

On the **Connectivity & security** tab, note the database endpoint, because it's required to connect to the DB cluster.

database-1

Summary

DB cluster id	CPU
database-1	
Role	Current activity
Serverless	

Connectivity & security Monitoring Logs & events Configuration

Connectivity & security

Endpoint & port	Network
Endpoint	VPC
database-1. [REDACTED].us-east-1.rds.amazonaws.com	vpc-6
Port	Subnet
3306	default
	Subnet
	subnet

A red oval highlights the "Endpoint" row.

Choose the Configuration tab to view the capacity settings.

The screenshot shows the 'Configuration' tab selected in the top navigation bar. Under the 'Database' heading, there's a 'Configuration' section with the following details:

- Resource id:** cluster-[REDACTED]
- ARN:** arn:aws:rds:us-east-1:[REDACTED]:cluster:harsht-database-10
- DB cluster parameter group:** default.aurora5.6
- Deletion protection:** Disabled

On the right, under 'Capacity settings', the following configuration is shown:

- Minimum Aurora capacity unit:** 2 capacity units
- Maximum Aurora capacity unit:** 16 capacity units
- Pause compute capacity after consecutive minutes of inactivity:** 5 minutes
- Force scaling the capacity to the specified values when the timeout is reached:** Enabled

A *scaling event* is generated when the DB cluster scales up, scales down, pauses, or resumes. Choose the **Logs & events** tab to see recent events. The following image shows examples of these events.

The screenshot shows the 'Logs & events' tab selected in the top navigation bar. Under the 'Recent events (2)' heading, two entries are listed:

Time	System notes
Mon Aug 06 17:04:15 GMT-700 2018	The DB cluster has scaled from 8 capacity units to 4 capacity units.
Mon Aug 06 17:04:09 GMT-700 2018	Scaling DB cluster from 8 capacity units to 4 capacity units for this.

You can also monitor your Aurora Serverless DB cluster in CloudWatch. In particular, you can 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. For details about how to monitor Aurora clusters through CloudWatch, see [Monitoring Log Events in Amazon CloudWatch \(p. 761\)](#).

You can have Aurora publish some or all database logs to CloudWatch. You select which logs to publish by enabling the configuration parameters such as `general_log` and `slow_query_log` in the DB cluster parameter group associated with the Serverless cluster. Unlike provisioned clusters, Serverless clusters don't require you to specify in the DB cluster settings which log types to upload to CloudWatch. Serverless 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 connect to an Aurora Serverless DB cluster, use the database endpoint. Follow the instructions in [Connecting to an Amazon Aurora DB Cluster \(p. 166\)](#) to connect to your Aurora Serverless DB cluster.

Note

With Aurora Serverless, you can't connect directly to specific DB instances in the DB cluster.

Using the Data API for Aurora Serverless

You can access your Aurora Serverless DB cluster using the built-in Data API. Using this API, you can access Aurora Serverless with web services-based applications, including AWS Lambda, AWS AppSync, and AWS Cloud9. For more information on these applications, see details at [AWS Lambda](#), [AWS AppSync](#), and [AWS Cloud9](#).

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 and is terminated in 45 seconds if it's not finished processing. You can use the `continueAfterTimeout` parameter to continue running the SQL statement after the call times out.

The API uses database credentials stored in AWS Secrets Manager, so you don't need to pass credentials in the API calls. The API also 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 about AWS Secrets Manager, see [What Is AWS Secrets Manager?](#) in the *AWS Secrets Manager User Guide*.

Note

When you enable the Data API, you can also use the query editor for Aurora Serverless. For more information, see [Using the Query Editor for Aurora Serverless \(p. 161\)](#).

Availability of the Data API

The Data API is only available for the following Aurora Serverless DB clusters:

- Aurora with MySQL version 5.6 compatibility
- Aurora with PostgreSQL version 10.7 compatibility

The following table shows the AWS Regions where the Data API is currently available for Aurora Serverless. Use the HTTPS protocol to access the Data API in these AWS Regions.

Region	Link
US East (N. Virginia)	rds-data.us-east-1.amazonaws.com
US East (Ohio)	rds-data.us-east-2.amazonaws.com
US West (Oregon)	rds-data.us-west-2.amazonaws.com
Europe (Ireland)	rds-data.eu-west-1.amazonaws.com
Asia Pacific (Tokyo)	rds-data.ap-northeast-1.amazonaws.com

Authorizing Access to the Data API

A user must be authorized to access the Data API. You can authorize a user to access the Data API by adding the `AmazonRDSDataFullAccess` policy, a predefined AWS Identity and Access Management (IAM) policy, to that user.

You can also create an IAM policy that grants access to the Data API. After you create the policy, add it to each user that requires access to the Data API.

The following policy provides the minimum required permissions for a user to access the Data API.

```
{  
    "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": [  
                "secretsmanager>CreateSecret",  
                "secretsmanager>ListSecrets",  
                "secretsmanager:GetRandomPassword",  
                "tag:GetResources",  
                "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 *IAM User Guide*.

For information about adding an IAM policy to a user, see [Adding and Removing IAM Identity Permissions](#) in the *IAM User Guide*.

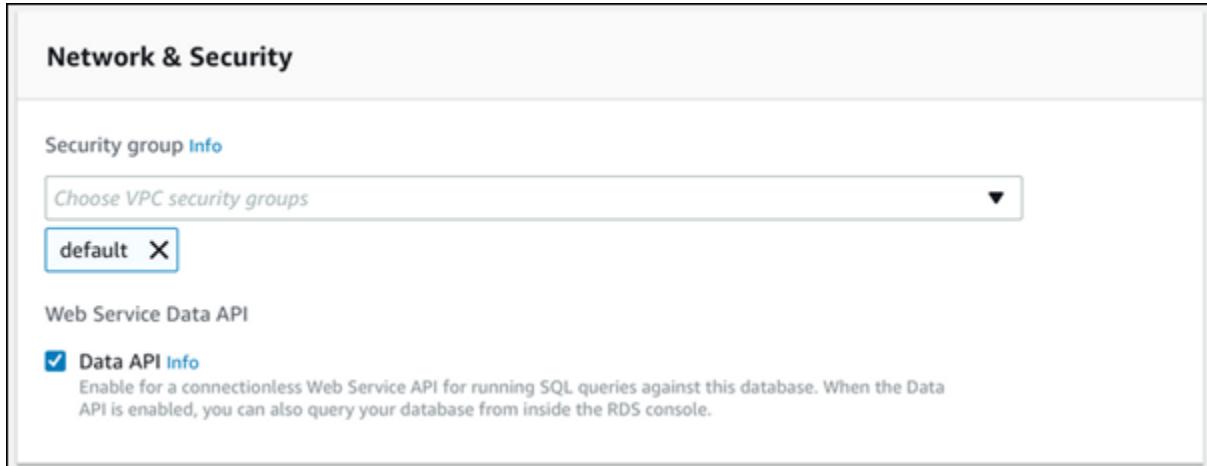
Enabling the Data API

To use the Data API, enable it for your Aurora Serverless DB cluster. You can enable the Data API when you create or modify the DB cluster.

Console

You can enable the Data API by using the RDS console when you create or modify an Aurora Serverless DB cluster. When you create an Aurora Serverless DB cluster, you do so by enabling the Data API in the RDS console's **Connectivity** section. When you modify an Aurora Serverless 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 DB cluster.



For instructions, see [Creating an Aurora Serverless DB Cluster \(p. 124\)](#) and [Modifying an Aurora Serverless DB Cluster \(p. 132\)](#).

AWS CLI

When you create or modify an Aurora Serverless DB cluster using AWS CLI commands, the Data API is enabled when you specify `--enable-http-endpoint`.

You can specify the `--enable-http-endpoint` using the following AWS CLI commands:

- [create-db-cluster](#)
- [modify-db-cluster](#)

The following example modifies `sample-cluster` to enable the Data API.

For Linux, OS X, or Unix:

```
aws rds modify-db-cluster \
--db-cluster-identifier sample-cluster \
--enable-http-endpoint
```

For Windows:

```
aws rds modify-db-cluster ^
--db-cluster-identifier sample-cluster ^
--enable-http-endpoint
```

RDS API

When you create or modify an Aurora Serverless DB cluster using Amazon RDS API operations, you enable the Data API by setting the `EnableHttpEndpoint` value to `true`.

You can set the `EnableHttpEndpoint` value using the following API operations:

- [CreateDBCluster](#)

- [ModifyDBCluster](#)

Storing Database Credentials in AWS Secrets Manager

When you call the Data API, you can pass credentials for the Aurora Serverless DB cluster by using a secret in AWS 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 AWS Secrets Manager to create a secret that contains credentials for the Aurora DB cluster.
For instructions, see [Creating a Basic Secret](#) in the *AWS Secrets Manager User Guide*.
2. Use the AWS 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.

Calling the Data API

After you enable the Data API for an Aurora Serverless DB cluster, you can call the Data API or the AWS CLI to run SQL statements on the DB cluster. The Data API supports the programming languages supported by the AWS SDK. For more information, see [Tools to Build on AWS](#).

The Data API provides the following operations to execute SQL statements.

Data API Operation	AWS CLI Command	Description
<code>ExecuteStatement</code>	<code>rds-data execute-statement</code>	Runs a SQL statement against 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 DML statement with array of parameter sets. A batch SQL statement can provide a significant performance improvement over individual insert and update statements.

You can run both operations for executing a SQL statement independently, or you can run them in a transaction. The Data API provides the following operations to support transactions.

Data API Operation	AWS CLI Command	Description
<code>BeginTransaction</code>	<code>rds-data begin-transaction</code>	Starts a SQL transaction.
<code>CommitTransaction</code>	<code>rds-data commit-transaction</code>	Ends a SQL transaction and commits the changes.
<code>RollbackTransaction</code>	<code>rds-data rollback-transaction</code>	Performs a rollback of a transaction.

The operations for executing 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
resourceArn	--resource-arn	Yes	The Amazon Resource Name (ARN) of the Aurora Serverless DB cluster.
secretArn	--secret-arn	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 you specify in Data API calls.

JDBC Data Type	Data API Data Type
INTEGER, TINYINT, SMALLINT, BIGINT	LONG
FLOAT, REAL, DOUBLE	DOUBLE
DECIMAL	STRING
BOOLEAN, BIT	BOOLEAN
BLOB, BINARY, LONGVARBINARY, VARBINARY	BLOB
CLOB	STRING
Other types (including types related to date and time)	STRING

For some specific types, such as `DECIMAL` or `TIME`, a hint might be required to instruct the Data API that the `String` value should be passed to the database as a different type. You can do this by including values in `typeHint` in the `SqlParameter` data type. The possible values for `typeHint` are the following:

- `DECIMAL` – The corresponding `String` parameter value is sent as an object of `DECIMAL` type to the database.
- `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]`.
- `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]`.
- `DATE` – The corresponding `String` parameter value is sent as an object of `DATE` type to the database. The accepted format is `YYYY-MM-DD`.

Examples

- [Calling the Data API with the AWS CLI \(p. 144\)](#)
- [Calling the Data API from a Python Application \(p. 151\)](#)

- [Calling the Data API from a Java Application \(p. 154\)](#)

Note

These examples are not exhaustive.

Calling the Data API with the AWS CLI

You can call the Data API using the AWS Command Line Interface (AWS CLI).

The following examples use the AWS CLI for the Data API. For more information, see [AWS Command Line Interface Reference for the Data API](#).

In each example, replace the DB cluster ARN with the ARN for your Aurora Serverless DB cluster. Also, replace the secret ARN with the ARN of the secret in AWS Secrets Manager that allows access to the DB cluster.

Note

The AWS CLI can format responses in JSON.

Topics

- [Starting a SQL Transaction \(p. 144\)](#)
- [Running a SQL Statement \(p. 145\)](#)
- [Running a Batch SQL Statement Over an Array of Data \(p. 148\)](#)
- [Committing a SQL Transaction \(p. 149\)](#)
- [Rolling Back a SQL Transaction \(p. 150\)](#)

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.

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 following option:

- `--database` (optional) – The name of the database.

For example, the following CLI command starts a SQL transaction.

For Linux, OS X, or Unix:

```
aws rds-data begin-transaction --resource-arn "arn:aws:rds:us-east-1:123456789012:cluster:mydbcluster" \
--database "mydb" --secret-arn "arn:aws:secretsmanager:us-east-1:123456789012:secret:mysecret"
```

For Windows:

```
aws rds-data begin-transaction --resource-arn "arn:aws:rds:us-  
east-1:123456789012:cluster:mydbcluster" ^  
--database "mydb" --secret-arn "arn:aws:secretsmanager:us-  
east-1:123456789012:secret:mysecret"
```

The following is an example of the response.

```
{  
    "transactionId": "ABC1234567890xyz"  
}
```

Running a SQL Statement

You can run a SQL statement using the `aws rds-data execute-statement` CLI command.

You can run the SQL statement in a transaction by specifying the transaction identifier with the `--transaction-id` option. You can start a transaction using the `aws rds-data begin-transaction` CLI command. You can end and commit a transaction using the `aws rds-data commit-transaction` CLI command.

Important

If you don't specify the `--transaction-id` option, changes that result from the call are committed automatically.

In addition to the common options, specify the following options:

- `--sql` (required) – A SQL statement to run on the DB cluster.
- `--transaction-id` (optional) – The identifier of a transaction that was started using the `begin-transaction` CLI command. Specify the transaction ID of the transaction that you want to include the SQL statement in.
- `--parameters` (optional) – The parameters for the SQL statement.
- `--include-result-metadata` | `--no-include-result-metadata` (optional) – A value that indicates whether to include metadata in the results. The default is `--no-include-result-metadata`.
- `--database` (optional) – The name of the database.
- `--continue-after-timeout` | `--no-continue-after-timeout` (optional) – A value that indicates whether to continue running the statement after the call times out. The default is `--no-continue-after-timeout`.

For data definition language (DDL) statements, we recommend continuing to run the statement after the call times out to avoid errors and the possibility of corrupted data structures.

The DB cluster returns a response for the call.

Note

The response size limit is 1 MB or 1,000 records. If the call returns more than 1 MB of response data or over 1,000 records, the call is terminated.

For example, the following CLI command runs a single SQL statement and omits the metadata in the results (the default).

For Linux, OS X, 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, OS X, 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, OS X, or Unix:

```
aws rds-data execute-statement --resource-arn "arn:aws:rds:us-east-1:123456789012:cluster:mydbcluster" \
--database "mydb" --secret-arn "arn:aws:secretsmanager:us-east-1:123456789012:secret:mysecret" \
--sql "insert into mytable values (:id, :val)" --parameters "[{"name": "\id", "value": "\longValue": 1}, {"name": "\val", "value": {"stringValue": "\value1"}}]"
```

For Windows:

```
aws rds-data execute-statement --resource-arn "arn:aws:rds:us-east-1:123456789012:cluster:mydbcluster" ^
--database "mydb" --secret-arn "arn:aws:secretsmanager:us-east-1:123456789012:secret:mysecret" ^
--sql "insert into mytable values (:id, :val)" --parameters "[{"name": "\id", "value": "\longValue": 1}, {"name": "\val", "value": {"stringValue": "\value1"}}]"
```

The following is an example of the response.

```
{
    "numberOfRecordsUpdated": 1
}
```

The following CLI command runs a data definition language (DDL) SQL statement. The DDL statement renames column job to column role.

Important

For DDL statements, we recommend continuing to run the statement after the call times out. When a DDL statement terminates before it is finished running, it can result in errors and

possibly corrupted data structures. To continue running a statement after a call times out, specify the `--continue-after-timeout` option.

For Linux, OS X, 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.
- `--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

The response size limit is 1 MB or 1,000 records. If the call returns more than 1 MB of response data or over 1,000 records, the call is terminated.

For example, the following CLI command runs a batch SQL statement over an array of data with a parameter set.

For Linux, OS X, 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, OS X, 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, OS X, or Unix:

```
aws rds-data rollback-transaction --resource-arn "arn:aws:rds:us-east-1:123456789012:cluster:mydbcluster" \
--secret-arn "arn:aws:secretsmanager:us-east-1:123456789012:secret:mysecret" \
--transaction-id "ABC1234567890xyz"
```

For Windows:

```
aws rds-data rollback-transaction --resource-arn "arn:aws:rds:us-east-1:123456789012:cluster:mydbcluster" ^
--secret-arn "arn:aws:secretsmanager:us-east-1:123456789012:secret:mysecret" ^
--transaction-id "ABC1234567890xyz"
```

The following is an example of the response.

```
{  
    "transactionStatus": "Rollback Complete"  
}
```

Calling the Data API from a Python Application

You can call the Data API from a Python application.

The following examples use the AWS SDK for Python (Boto). For more information about Boto, see the [AWS SDK for Python \(Boto 3\) Documentation](#).

In each example, replace the DB cluster's Amazon Resource Name (ARN) with the ARN for your Aurora Serverless DB cluster. Also, replace the secret ARN with the ARN of the secret in AWS Secrets Manager that allows access to the DB cluster.

Topics

- [Running a SQL Query \(p. 151\)](#)
- [Running a DML SQL Statement \(p. 152\)](#)
- [Running a SQL Transaction \(p. 153\)](#)

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': 'ROSALEZ'  
        }  
    ]  
]
```

```
{  
    'stringValue': 'ALEJANDRO'  
},  
{  
    'stringValue': '2016-02-15 04:34:33.0'  
}  
],  
[  
    {  
        'longValue': 1  
    },  
    {  
        'stringValue': 'DOE'  
    },  
    {  
        'stringValue': 'JANE'  
    },  
    {  
        'stringValue': '2014-05-09 04:34:33.0'  
    }  
],  
[  
    {  
        'longValue': 1  
    },  
    {  
        'stringValue': 'STILES'  
    },  
    {  
        'stringValue': 'JOHN'  
    },  
    {  
        'stringValue': '2017-09-20 04:34:33.0'  
    }  
]  
]
```

Running a DML SQL Statement

You can run a data manipulation language (DML) statement to insert, update, or delete data in your database. You can also use parameters in DML statements.

Important

If a call isn't part of a transaction because it doesn't include the `transactionID` parameter, changes that result from the call are committed automatically.

The following example runs an insert SQL statement and uses parameters.

```
import boto3  
  
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'  
  
response2 = rdsData.execute_statement(  
    resourceArn = cluster_arn,  
    secretArn = secret_arn,  
    database = 'mydb',  
    sql = 'insert into employees(first_name, last_name)  
VALUES(:firstname, :lastname)')  
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',
    parameters = paramSet,
    sql = 'insert into employees(first_name, last_name)
VALUES(:firstname, :lastname)')
'numberOfRecordsUpdated': 1

response2['numberOfRecordsUpdated']
1
```

Running a SQL Transaction

You can start a SQL transaction, run one or more SQL statements, and then commit the changes with a Python application.

Important

A transaction times out if there are no calls that use its transaction ID in three minutes. If a transaction times out before it's committed, it's rolled back automatically.

If you don't specify a transaction ID, changes that result from the call are committed automatically.

The following example runs a SQL transaction that inserts a row in a table.

```
import boto3

rdsData = boto3.client('rds-data')

cluster_arn = 'arn:aws:rds:us-east-1:123456789012:cluster:mydbcluster'
secret_arn = 'arn:aws:secretsmanager:us-east-1:123456789012:secret:mysecret'

tr = rdsData.begin_transaction(
    resourceArn = cluster_arn,
    secretArn = secret_arn,
    database = 'mydb')

response3 = rdsData.execute_statement(
    resourceArn = cluster_arn,
    secretArn = secret_arn,
    database = 'mydb',
    sql = 'insert into employees(first_name, last_name) values('XIULAN', 'WANG')',
    transactionId = tr['transactionId'])

cr = rdsData.commit_transaction(
    resourceArn = cluster_arn,
    secretArn = secret_arn,
    transactionId = tr['transactionId'])

cr['transactionStatus']
'Transaction Committed'

response3['numberOfRecordsUpdated']
1
```

Note

If you run a data definition language (DDL) statement, we recommend continuing to run the statement after the call times out. When a DDL statement terminates before it is finished

running, it can result in errors and possibly corrupted data structures. To continue running a statement after a call times out, set the `continueAfterTimeout` parameter to `true`.

Calling the Data API from a Java Application

You can call the Data API from a Java application.

The following examples use the AWS SDK for Java. For more information, see the [AWS SDK for Java Developer Guide](#).

In each example, replace the DB cluster's Amazon Resource Name (ARN) with the ARN for your Aurora Serverless DB cluster. Also, replace the secret ARN with the ARN of the secret in AWS Secrets Manager that allows access to the DB cluster.

Topics

- [Running a SQL Query \(p. 154\)](#)
- [Running a SQL Transaction \(p. 155\)](#)
- [Running a Batch SQL Operation \(p. 156\)](#)

Running a SQL Query

You can run a `SELECT` statement and fetch the results with a Java application.

The following example runs a SQL query.

```
package com.amazonaws.rdsdata.examples;

import com.amazonaws.services.rdsdata.AWSRDSData;
import com.amazonaws.services.rdsdata.AWSRDSDataClient;
import com.amazonaws.services.rdsdata.model.ExecuteStatementRequest;
import com.amazonaws.services.rdsdata.model.ExecuteStatementResult;
import com.amazonaws.services.rdsdata.model.Field;

import java.util.List;

public class FetchResultsExample {
    public static final String RESOURCE_ARN = "arn:aws:rds:us-
east-1:123456789012:cluster:mydbcluster";
    public static final String SECRET_ARN = "arn:aws:secretsmanager:us-
east-1:123456789012:secret:mysecret";

    public static void main(String[] args) {
        AWSRDSData rdsData = AWSRDSDataClient.builder().build();

        ExecuteStatementRequest request = new ExecuteStatementRequest()
            .withResourceArn(RESOURCE_ARN)
            .withSecretArn(SECRET_ARN)
            .withDatabase("mydb")
            .withSql("select * from mytable");

        ExecuteStatementResult result = rdsData.executeStatement(request);

        for (List<Field> fields: result.getRecords()) {
            String stringValue = fields.get(0).getStringValue();
            long numberValue = fields.get(1).getLongValue();

            System.out.println(String.format("Fetched row: string = %s, number = %d",
                stringValue, numberValue));
        }
    }
}
```

```
}
```

Running a SQL Transaction

You can start a SQL transaction, run one or more SQL statements, and then commit the changes with a Java application.

Important

A transaction times out if there are no calls that use its transaction ID in three minutes. If a transaction times out before it's committed, it's rolled back automatically.

If you don't specify a transaction ID, changes that result from the call are committed automatically.

The following example runs a SQL transaction.

```
package com.amazonaws.rdsdata.examples;

import com.amazonaws.services.rdsdata.AWSRDSData;
import com.amazonaws.services.rdsdata.AWSRDSDataClient;
import com.amazonaws.services.rdsdata.model.BeginTransactionRequest;
import com.amazonaws.services.rdsdata.model.BeginTransactionResult;
import com.amazonaws.services.rdsdata.model.CommitTransactionRequest;
import com.amazonaws.services.rdsdata.model.ExecuteStatementRequest;

public class TransactionExample {
    public static final String RESOURCE_ARN = "arn:aws:rds:us-
east-1:123456789012:cluster:mydbcluster";
    public static final String SECRET_ARN = "arn:aws:secretsmanager:us-
east-1:123456789012:secret:mysecret";

    public static void main(String[] args) {
        AWSRDSData rdsData = AWSRDSDataClient.builder().build();

        BeginTransactionRequest beginTransactionRequest = new BeginTransactionRequest()
            .withResourceArn(RESOURCE_ARN)
            .withSecretArn(SECRET_ARN)
            .withDatabase("mydb");
        BeginTransactionResult beginTransactionResult =
rdsData.beginTransaction(beginTransactionRequest);
        String transactionId = beginTransactionResult.getTransactionId();

        ExecuteStatementRequest executeStatementRequest = new ExecuteStatementRequest()
            .withTransactionId(transactionId)
            .withResourceArn(RESOURCE_ARN)
            .withSecretArn(SECRET_ARN)
            .withSql("INSERT INTO test_table VALUES ('hello world!')");
        rdsData.executeStatement(executeStatementRequest);

        CommitTransactionRequest commitTransactionRequest = new CommitTransactionRequest()
            .withTransactionId(transactionId)
            .withResourceArn(RESOURCE_ARN)
            .withSecretArn(SECRET_ARN);
        rdsData.commitTransaction(commitTransactionRequest);
    }
}
```

Note

If you run a data definition language (DDL) statement, we recommend continuing to run the statement after the call times out. When a DDL statement terminates before it is finished

running, it can result in errors and possibly corrupted data structures. To continue running a statement after a call times out, set the `continueAfterTimeout` parameter to `true`.

Running a Batch SQL Operation

You can run bulk insert and update operations over an array of data with a Java application. You can run a DML statement with array of parameter sets.

Important

If you don't specify a transaction ID, changes that result from the call are committed automatically.

The following example runs a batch insert operation.

```
package com.amazonaws.rdsdata.examples;

import com.amazonaws.services.rdsdata.AWSRDSData;
import com.amazonaws.services.rdsdata.AWSRDSDataClient;
import com.amazonaws.services.rdsdata.model.BatchExecuteStatementRequest;
import com.amazonaws.services.rdsdata.model.Field;
import com.amazonaws.services.rdsdata.model.SqlParameter;

import java.util.Arrays;

public class BatchExecuteExample {
    public static final String RESOURCE_ARN = "arn:aws:rds:us-
east-1:123456789012:cluster:mydbcluster";
    public static final String SECRET_ARN = "arn:aws:secretsmanager:us-
east-1:123456789012:secret:mysecret";

    public static void main(String[] args) {
        AWSRDSData rdsData = AWSRDSDataClient.builder().build();

        BatchExecuteStatementRequest request = new BatchExecuteStatementRequest()
            .withDatabase("test")
            .withResourceArn(RESOURCE_ARN)
            .withSecretArn(SECRET_ARN)
            .withSql("INSERT INTO test_table2 VALUES (:string, :number)")
            .withParameterSets(Arrays.asList(
                Arrays.asList(
                    new SqlParameter().withName("string").WithValue(new
Field().withStringValue("Hello")),
                    new SqlParameter().withName("number").WithValue(new
Field().withLongValue(1L))
                ),
                Arrays.asList(
                    new SqlParameter().withName("string").WithValue(new
Field().withStringValue("World")),
                    new SqlParameter().withName("number").WithValue(new
Field().withLongValue(2L))
                )
            ));
        rdsData.batchExecuteStatement(request);
    }
}
```

Using the Java Client Library for Data API (Preview)

This is prerelease documentation for a service in preview release. It is subject to change.

You can download and use a Java client library for the Data API. The 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. You can build the library manually from the source files, but the best practice is to consume the library using Apache Maven dependency management. To do this, use the following procedure.

To use the Data API in your application as a dependency

1. Download and install Apache Maven. For more information, see [Downloading Apache Maven](#) and [Installing Apache Maven](#) in the Maven documentation.
2. Add the Data API Maven repository to your application's Project Object Model (POM) file as shown following.

```
<!--Dependency:-->
<dependencies>
    <dependency>
        <groupId>com.amazonaws</groupId>
        <artifactId>rdsdata-client-library</artifactId>
        <version>1.0</version>
    </dependency>
</dependencies>
<!--Custom repository:-->
<repositories>
    <repository>
        <id>rdsdata-client-repository</id>
        <name>RDS Data Client Library Release Repository</name>
        <url>https://rds-data-client-library-java.s3.amazonaws.com/maven/release</url>
    </repository>
</repositories>
```

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 `balance`. You also have the following data transfer object (DTO).

```
public class Account {
    String accountId;
    double balance;
    // 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("A-1", 1.1);
var account2 = new Account("B-2", 100);
client.forSql("INSERT INTO accounts(accountId, balance) VALUES(:accountId, :balance)")
    .withParams(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, balance) VALUES(:accountId, :balance)")  
    .withParam("accountId", "A-1")  
    .withParam("balance", 12.2)  
    .execute();
```

The following is another example that works with simple values as input parameters.

```
client.forSql("INSERT INTO accounts(accountId, balance) VALUES(?, ?, "A-1", 12.2)  
    .execute();
```

The client library provides automatic mapping to DTOs when an execution result is returned. The following examples show how the execution 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);
```

Troubleshooting Data API Issues

Use the following sections, titled with common error messages, to help troubleshoot problems that you have with the Data API.

Note

If you have questions or comments related to the Data API, send email to rds-data-api-feedback@amazon.com.

Topics

- [Transaction <transaction_ID> Is Not Found \(p. 158\)](#)
- [Packet for Query Is Too Large \(p. 159\)](#)
- [Query Response Exceeded Limit of Number of Records \(p. 159\)](#)
- [Database Response Exceeded Size Limit \(p. 159\)](#)
- [HttpEndpoint Is Not Enabled for Cluster <cluster_ID> \(p. 159\)](#)

[Transaction <transaction_ID> Is Not Found](#)

In this case, the transaction ID specified in a Data API call wasn't found. The cause for this issue is almost always one of the following:

- The specified transaction ID wasn't created by a [BeginTransaction](#) call.

- The specified transaction ID has expired.

A transaction expires if no call uses the transaction ID within three minutes.

To solve the issue, make sure that your call has a valid transaction ID. Also make sure that each transaction call runs within three minutes of the last one.

For information about running transactions, see [Calling the Data API \(p. 142\)](#).

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.

Query Response Exceeded Limit of Number of Records

In this case, the number of rows in the result set returned was too large. The Data API limit is 1,000 rows in the result set returned by the database.

To solve this issue, make sure that calls to the Data API return 1,000 rows or less. If you need to return more than 1,000 rows, 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.

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 MB in the result set returned by the database.

To solve this issue, make sure that calls to the Data API return 1 MB of data or less. If you need to return more than 1 MB, you can use multiple `ExecuteStatement` calls with the `LIMIT` clause in your query.

For more information about the `LIMIT` clause, see [SELECT Syntax](#) in the MySQL documentation.

HttpEndpoint Is Not Enabled for Cluster <cluster_ID>

The cause for this issue is almost always one of the following:

- The Data API isn't enabled for the Aurora Serverless DB cluster. To use the Data API with an Aurora Serverless DB cluster, the Data API must be enabled for the DB cluster.
- The DB cluster was renamed after the Data API was enabled for it.

If the Data API has not been enabled for the DB cluster, enable it.

If the DB cluster was renamed after the Data API was enabled for the DB cluster, disable the Data API and then enable it again.

For information about enabling the Data API, see [Enabling the Data API \(p. 140\)](#).

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"
}
```

Using the Query Editor for Aurora Serverless

With the query editor for Aurora Serverless, you can run SQL queries in the RDS console. You can run any valid SQL statement on the Aurora Serverless DB cluster, including data manipulation and data definition statements.

The query editor requires an Aurora Serverless DB cluster with the Data API enabled. For information about creating an Aurora Serverless DB cluster with the Data API enabled, see [Using the Data API for Aurora Serverless \(p. 139\)](#).

Authorizing Access to the Query Editor

A user must be authorized to run queries in the query editor. You can authorize a user to run queries in the query editor by adding the `AmazonRDSDataFullAccess` policy, a predefined AWS Identity and Access Management (IAM) policy, to that user.

You can also create an IAM policy that grants access to the query editor. After you create the policy, add it to each user that requires access to the query editor.

The following policy provides the minimum required permissions for a user to access the query editor.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "QueryEditor0",
```

```
        "Effect": "Allow",
        "Action": [
            "secretsmanager:GetSecretValue",
            "secretsmanager:PutResourcePolicy",
            "secretsmanager:PutSecretValue",
            "secretsmanager>DeleteSecret",
            "secretsmanager:DescribeSecret",
            "secretsmanager:TagResource"
        ],
        "Resource": "arn:aws:secretsmanager:***:secret:rds-db-credentials/*"
    },
    {
        "Sid": "QueryEditor1",
        "Effect": "Allow",
        "Action": [
            "secretsmanager:GetRandomPassword",
            "tag:GetResources",
            "secretsmanager>CreateSecret",
            "secretsmanager>ListSecrets",
            "dbqms>CreateFavoriteQuery",
            "dbqms:DescribeFavoriteQueries",
            "dbqms:UpdateFavoriteQuery",
            "dbqms:DeleteFavoriteQueries",
            "dbqms:GetQueryString",
            "dbqms>CreateQueryHistory",
            "dbqms:UpdateQueryHistory",
            "dbqms:DeleteQueryHistory",
            "dbqms:DescribeQueryHistory",
            "rds-data:BatchExecuteStatement",
            "rds-data:BeginTransaction",
            "rds-data:CommitTransaction",
            "rds-data:ExecuteStatement",
            "rds-data:RollbackTransaction"
        ],
        "Resource": "*"
    }
]
```

For information about creating an IAM policy, see [Creating IAM Policies](#) in the *AWS Identity and Access Management User Guide*.

For information about adding an IAM policy to a user, see [Adding and Removing IAM Identity Permissions](#) in the *AWS Identity and Access Management User Guide*.

Running Queries in the Query Editor

You can run SQL statements on an Aurora Serverless DB cluster in the query editor.

To run a query in the query editor

1. Sign in to the AWS Management Console and open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. In the upper-right corner of the AWS Management Console, choose the AWS Region in which you created the Aurora Serverless DB clusters that you want to query.
3. In the navigation pane, choose **Databases**.
4. Choose the Aurora Serverless DB cluster that you want to run SQL queries against.
5. For **Actions**, choose **Query**. If you haven't connected to the database before, the **Connect to database** page opens.

Connect to database

You need to choose a database and enter the database credentials to use the query editor. We will be storing your credentials and the connection in the AWS Secrets Manager service. [Learn more](#)

Database instance or cluster

database-1

Database username

Add new database credentials

Enter database username

Enter database password

Enter the name of the database or schema (optional)

Enter the name for schemas collection

Enter database or schema name

Cancel **Connect to database**

6. Enter the following information:
 - a. For **Database instance or cluster**, choose the Aurora Serverless DB cluster that you want to run SQL queries on.
 - b. For **Database username**, choose the user name of the database user to connect with, or choose **Add new database credentials**. If you choose **Add new database credentials**, enter the user name for the new database credentials in **Enter database username**.
 - c. For **Enter database password**, enter the password for the database user that you chose.
 - d. In the last box, enter the name of the database or schema that you want to use for the Aurora DB cluster.
 - e. Choose **Connect to database**.

Note

If your connection is successful, your connection and authentication information are stored in AWS Secrets Manager. You don't need to enter the connection information again.

7. In the query editor, enter the SQL query that you want to run on the database.

A screenshot of the Amazon Aurora Query Editor interface. At the top, there are three tabs: 'Editor' (which is selected), 'Recent', and 'Saved queries'. Below the tabs, a code editor window contains the following SQL script:

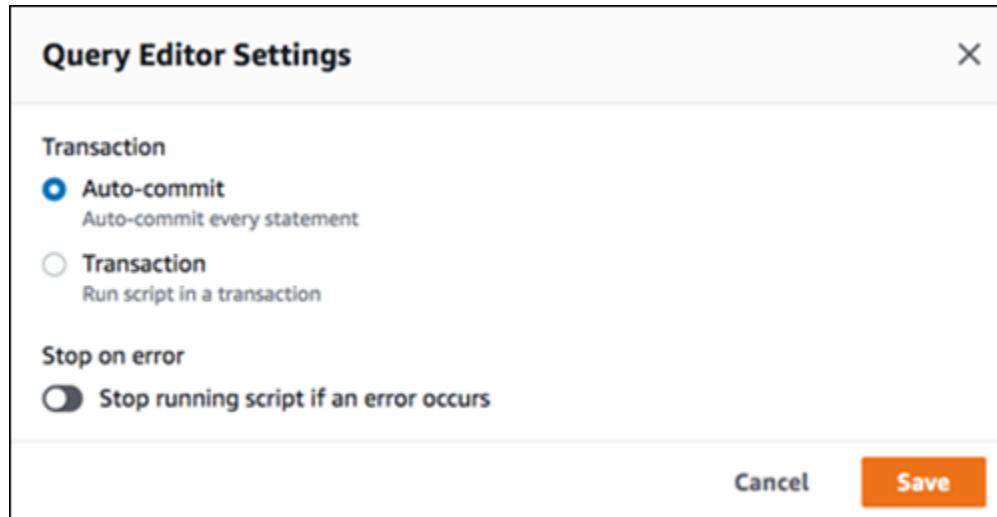
```
1 select * from information_schema.tables;
2 # Press run and see the current database tables below
```

Below the code editor are four buttons: 'Run' (orange), 'Save', 'Clear', and 'Change database'. Underneath these buttons is a section labeled 'Output' with a search bar containing 'Search rows' and an 'Export to csv' button. A message at the bottom of the output section says 'Empty results' and 'You haven't finished running any queries yet.' To the right of the output section is a vertical scroll bar.

Each SQL statement can commit automatically, or you can run SQL statements in a script as part of a transaction. To control this behavior, choose the gear icon above the query window.

A screenshot of the Amazon Aurora Query Editor interface, similar to the one above but with a red circle highlighting the gear icon located in the top right corner of the main window area.

The **Query Editor Settings** window appears.



If you choose **Auto-commit**, every SQL statement commits automatically. If you choose **Transaction**, you can run a group of statements in a script, and they don't commit automatically. If **Transaction** is set, the statements in the group are committed when choose **Run**. 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**.

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.

For specialized tasks such as troubleshooting, you can use the host and port information from a specific DB instance in your Aurora DB cluster.

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 [User Account Management](#) in the MySQL documentation. For more information on using IAM database authentication, see [IAM Database Authentication \(p. 207\)](#).

When you have a connection to your Amazon Aurora DB cluster with MySQL 5.6 compatibility, you can execute 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](#).

When you have a connection to your Amazon Aurora DB cluster with MySQL 5.7 compatibility, you can execute 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 [Comparison of Aurora MySQL 5.7 and MySQL 5.7 \(p. 580\)](#).

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.

Note

If the cluster is an Aurora Serverless DB cluster, you can only connect to its database endpoint. For more information, see [Using Amazon Aurora Serverless \(p. 116\)](#).

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 MySQL database cluster named "aurora-cl-mysql".

Related Databases:

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

Endpoints (2):

Endpoint name	Status	Type	Port
aurora-cl-mysql.cluster-ro...us-east-1.rds.amazonaws.com	Available	Reader	3306
aurora-cl-mysql.cluster...us-east-1.rds.amazonaws.com	Available	Writer	3306

Connection Utilities for Aurora MySQL

Some connection utilities you can use are the following:

- **Command line** – You can connect to an Amazon Aurora DB cluster by using tools like the MySQL command line utility. For more information on using the MySQL utility, see [mysql - The MySQL Command Line Tool](#) in the MySQL documentation.
- **GUI** – You can use the MySQL Workbench utility to connect by using a UI interface. For more information, see the [Download MySQL Workbench](#) page.
- **Applications** – You can use the MariaDB Connector/J utility to connect your applications to your Aurora DB cluster. For more information, see the [MariaDB Connector/J download](#) page.

Note

If you use the MariaDB Connector/J utility with an Aurora Serverless cluster, use the prefix `jdbc:mariadb:aurora://` in your connection string. The `mariadb:aurora` parameter avoids the automatic DNS scan for failover targets. That scanning is not needed with Aurora Serverless clusters and causes a delay in establishing the connection.

You can use SSL encryption on connections to an Amazon Aurora DB instance. For information, see [Using SSL with a MySQL DB Instance](#).

Note

Because you can create Amazon Aurora DB cluster only in an Amazon Virtual Private Cloud (VPC), connections to an Amazon Aurora DB cluster from AWS instances that are not in a VPC have been required to use the public endpoint address of the Amazon Aurora DB cluster. However, you can now communicate with an Amazon EC2 instance that is not in a VPC and an Amazon Aurora DB cluster using ClassicLink. For more information, see [A DB Instance in a VPC Accessed by an EC2 Instance Not in a VPC \(p. 248\)](#).

Connecting with SSL for Aurora MySQL

To connect using SSL, use the MySQL utility as described in the following procedure. If you are using IAM database authentication, you must use an SSL connection. For information, see [IAM Database Authentication \(p. 207\)](#).

Note

In order 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 \(p. 4\)](#).

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

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 --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 --ssl-ca=[full path]rds-combined-ca-bundle.pem --ssl-verify-server-cert
```

You should see output similar to the following.

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

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

mysql>
```

For general instructions on constructing Amazon RDS 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 version 9.6.3.

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 Aurora PostgreSQL DB cluster you can find the cluster endpoint. You use this endpoint in your PostgreSQL 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:5432`, 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.

RDS > Databases > aurora-cl-postgresql

aurora-cl-postgresql

Related

DB identifier

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

Connectivity & security | Monitoring | Logs & events | Configuration | Maintenance & backups | Tags

Endpoints (2)

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

Manage IAM roles

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.

Note

Because you can create an Amazon Aurora PostgreSQL DB cluster only in an Amazon Virtual Private Cloud (VPC), connections to an Aurora PostgreSQL DB cluster from AWS instances that are not in a VPC have been required to use the public endpoint address of the Aurora PostgreSQL DB cluster. However, you can now communicate with an Amazon EC2 instance that

is not in a VPC and an Aurora PostgreSQL DB cluster using ClassicLink. For more information, see [A DB Instance in a VPC Accessed by an EC2 Instance Not in a VPC \(p. 248\)](#).

Troubleshooting Aurora Connection Failures

Note

For a helpful and detailed guide on connecting to an Amazon Aurora MySQL DB cluster, you can see the [Aurora Connection Management](#) handbook.

Common causes of connection failures to a new Aurora DB cluster are as follows:

- The DB cluster was created using a VPC that does not allow connections from your device. To fix this failure, modify the VPC to allow connections from your device, or create a new VPC for your DB cluster that allows connections from your device. For an example, see [Create a VPC and Subnets \(p. 239\)](#).
- The DB cluster was created using the default port, and your company has firewall rules blocking connections to that port from devices in your company network. To fix this failure, recreate the instance with a different port.
- If you are using IAM database authentication, you might need to configure IAM database authentication. For information, see [IAM Database Authentication \(p. 207\)](#).

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 Amazon RDS 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 \(p. 587\)](#).

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 \(p. 887\)](#).

Security in Amazon Aurora

Cloud security at AWS is the highest priority. As an AWS customer, you benefit from a data center and network architecture that are built to meet the requirements of the most security-sensitive organizations.

Security is a shared responsibility between AWS and you. The [shared responsibility model](#) describes this as security *of* the cloud and security *in* the cloud:

- **Security of the cloud** – AWS is responsible for protecting the infrastructure that runs AWS services in the AWS Cloud. AWS also provides you with services that you can use securely. Third-party auditors regularly test and verify the effectiveness of our security as part of the [AWS compliance programs](#). To learn about the compliance programs that apply to Amazon Aurora (Aurora), see [AWS Services in Scope by Compliance Program](#).
- **Security in the cloud** – Your responsibility is determined by the AWS service that you use. You are also responsible for other factors including the sensitivity of your data, your organization's requirements, and applicable laws and regulations.

This documentation helps you understand how to apply the shared responsibility model when using Amazon Aurora. The following topics show you how to configure Amazon Aurora to meet your security and compliance objectives. You also learn how to use other AWS services that help you monitor and secure your Amazon Aurora resources.

You can manage access to your Amazon Aurora resources and your databases on a DB cluster. The method you use to manage access depends on what type of task the user needs to perform with Amazon Aurora:

- Run your DB cluster in a virtual private cloud (VPC) based on the Amazon VPC service for the greatest possible network access control. For more information about creating a DB cluster in a VPC, see [Amazon Virtual Private Cloud VPCs and Amazon Aurora \(p. 239\)](#).
- 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 \(p. 49\)](#).

- 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 \(p. 177\)](#).
- 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 \(p. 175\)](#).
- 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 \(p. 581\)](#).

For information about security with Aurora PostgreSQL, see [Security with Amazon Aurora PostgreSQL \(p. 879\)](#).

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

Aurora includes a high-performance storage subsystem. Its MySQL- and PostgreSQL-compatible database engines are customized to take advantage of that fast distributed storage. Aurora also automates and standardizes database clustering and replication, which are typically among the most challenging aspects of database configuration and administration.

For both Amazon RDS and Aurora, you can access the RDS API programmatically, and you can use the AWS CLI to access the RDS API interactively. Some RDS API operations and AWS CLI commands apply to both Amazon RDS and Aurora, while others apply to either Amazon RDS or Aurora. For information about RDS API operations, see [Amazon RDS API Reference](#). For more information about the AWS CLI, see [AWS Command Line Interface Reference for Amazon RDS](#).

Note

You only have to configure security for your use cases. You don't have to configure security access for processes that Amazon Aurora manages. These include creating backups, replicating data between a master and a Read Replica, and other processes.

For more information on managing access to Amazon Aurora resources and your databases on a DB cluster, see the following topics.

Topics

- [Data Protection in Amazon Aurora \(p. 174\)](#)
- [Identity and Access Management in Amazon Aurora \(p. 191\)](#)
- [Logging and Monitoring in Amazon Aurora \(p. 225\)](#)
- [Compliance Validation for Amazon Aurora \(p. 227\)](#)
- [Resilience in Amazon Aurora \(p. 228\)](#)
- [Infrastructure Security in Amazon Aurora \(p. 230\)](#)
- [Security Best Practices for Amazon Aurora \(p. 230\)](#)
- [Controlling Access with Security Groups \(p. 231\)](#)
- [Master User Account Privileges \(p. 234\)](#)
- [Using Service-Linked Roles for Amazon Aurora \(p. 235\)](#)
- [Amazon Virtual Private Cloud VPCs and Amazon Aurora \(p. 239\)](#)

Data Protection in Amazon Aurora

Amazon Aurora conforms to the AWS [shared responsibility model](#), which includes regulations and guidelines for data protection. AWS is responsible for protecting the global infrastructure that runs all the AWS services. AWS maintains control over data hosted on this infrastructure, including the security configuration controls for handling customer content and personal data. AWS customers and APN partners, acting either as data controllers or data processors, are responsible for any personal data that they put in the AWS Cloud.

For data protection, we recommend that you protect AWS account credentials and set up principals with AWS Identity and Access Management (IAM). Doing this means that 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.

- 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.
- For Aurora PostgreSQL, use database activity streams to monitor and audit database activity to provide safeguards for your database and meet compliance and regulatory requirements.

We strongly recommend that you never put sensitive identifying information, such as your customers' account numbers, into free-form fields such as a **Name** field. This recommendation includes when you work with Amazon Aurora or other AWS services using the console, API, AWS CLI, or AWS SDKs. Any data that you enter into Amazon Aurora or other services might get picked up for inclusion in diagnostic logs. When you provide a URL to an external server, don't include credentials information in the URL to validate your request to that server.

For more information about data protection, see the [AWS Shared Responsibility Model](#) and [GDPR](#) blog post on the [AWS Security Blog](#).

Topics

- [Protecting Data Using Encryption \(p. 175\)](#)
- [Internetwork Traffic Privacy \(p. 190\)](#)

Protecting Data Using Encryption

You can enable encryption for database resources. You can also encrypt connections to DB clusters.

Topics

- [Encrypting Amazon Aurora Resources \(p. 175\)](#)
- [Key Management \(p. 177\)](#)
- [Using SSL/TLS to Encrypt a Connection to a DB Cluster \(p. 177\)](#)
- [Rotating Your SSL/TLS Certificate \(p. 179\)](#)

Encrypting Amazon Aurora Resources

You can encrypt your Amazon Aurora DB clusters and snapshots at rest by enabling the encryption option for 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 \(p. 176\)](#)
- [Enabling Encryption for an Amazon Aurora DB Cluster \(p. 176\)](#)
- [Availability of Amazon Aurora Encryption \(p. 176\)](#)

- [Limitations of Amazon Aurora Encrypted DB Clusters \(p. 177\)](#)

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 data-at-rest encryption.

To manage the 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 encryption keys and define the policies that control how these keys can be used. AWS KMS supports CloudTrail, so you can audit key usage to verify that keys are being used appropriately. You can use your AWS KMS keys with Amazon Aurora and supported AWS services such as and supported AWS services such as Amazon S3, Amazon EBS, and Amazon Redshift. For a list of services that support AWS KMS, see [Supported Services](#) in the *AWS Key Management Service Developer Guide*.

For an Amazon Aurora encrypted DB cluster, all logs, backups, and snapshots are encrypted. You can also encrypt a Read Replica of an Amazon Aurora encrypted cluster. The encryption for the Read Replica is protected by the AWS Region's KMS master key.

Enabling Encryption for an Amazon Aurora DB Cluster

To enable encryption for a new DB cluster, choose **Enable encryption** on the console. For information on creating a DB cluster, see [Creating an Amazon Aurora DB Cluster \(p. 100\)](#).

If you use the `create-db-cluster` AWS CLI command to create an encrypted DB cluster, set the `--storage-encrypted` parameter to true. If you use the `CreateDBCluster` API operation, set the `StorageEncrypted` parameter to true.

When you create an encrypted DB cluster, you can also supply the AWS KMS key identifier for your encryption key. If you don't specify an AWS KMS key identifier, then Amazon Aurora uses your default encryption key for your new DB cluster. AWS KMS creates your default encryption key for Amazon Aurora for your AWS account. Your AWS account has a different default encryption key for each AWS Region.

Once you have created an encrypted DB cluster, you can't change the type of encryption key used by that DB cluster. Therefore, be sure to determine your encryption 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, set the `--kms-key-id` parameter to the Amazon Resource Name (ARN) for the KMS key for the DB cluster. If you use the RDS API `CreateDBCluster` action, set the `KmsKeyId` parameter to the ARN for your KMS key for the DB cluster.

You can use the ARN of a key from another account to encrypt a DB cluster. Or you might create a DB cluster with the same AWS account that owns the KMS encryption key used to encrypt that new DB cluster. In this case, the KMS key ID that you pass can be the KMS key alias instead of the key's ARN.

Important

In some cases, Amazon Aurora can lose access to the encryption key for a DB cluster. For example, Aurora loses access when RDS access to a key is revoked. In these cases, the encrypted DB cluster goes into a terminal state and you can only restore the DB cluster from a backup. We strongly recommend that you always enable backups for encrypted DB clusters to guard against the loss of encrypted data in your databases.

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:

- DB clusters that are encrypted can't be modified to disable encryption.
- You can't convert an unencrypted DB cluster to an encrypted one. However, you can restore an unencrypted Aurora DB cluster snapshot to an encrypted Aurora DB cluster. To do this, specify a KMS encryption key when you restore from the unencrypted DB cluster 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 identifier of the destination AWS Region. This is because KMS encryption keys are specific to the AWS Region that they are created in.

The source snapshot remains encrypted throughout the copy process. AWS Key Management Service uses envelope encryption to protect data during the copy process. For more information about envelope encryption, see [Envelope Encryption](#).

Key Management

You can manage keys used for Amazon Aurora encrypted DB clusters using the [AWS Key Management Service \(AWS KMS\)](#) in the IAM console. If you want full control over a key, then you must create a customer-managed key.

You can't delete, revoke, or rotate default keys provisioned by AWS KMS. You can't share a snapshot that has been encrypted using the default AWS KMS encryption key of the AWS account that shared the snapshot.

You can view audit logs of every action taken with a customer-managed key by using [AWS CloudTrail](#).

Important

When Aurora encounters a DB cluster encrypted by a key that Aurora doesn't have access to, Aurora puts the DB cluster into a terminal state. 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 encryption key for Aurora, and then restore the DB cluster from a backup.

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. 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 \(p. 581\)](#)
- [Security with Amazon Aurora PostgreSQL \(p. 879\)](#)

Important

For information about rotating your certificate, see [Rotating Your SSL/TLS Certificate \(p. 179\)](#).

Note

All certificates are only available for download using SSL/TLS connections.

To get a root certificate that works for all AWS Regions, download from one of these locations:

- <https://s3.amazonaws.com/rds-downloads/rds-ca-2019-root.pem> (CA-2019)
- <https://s3.amazonaws.com/rds-downloads/rds-ca-2015-root.pem> (CA-2015)

This root certificate is a trusted root entity and should work in most cases but might fail if your application doesn't accept certificate chains. If your application doesn't accept certificate chains, download the AWS Region-specific certificate from the list of intermediate certificates found later in this section.

To get a certificate bundle that contains both the intermediate and root certificates, download from <https://s3.amazonaws.com/rds-downloads/rds-combined-ca-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://s3.amazonaws.com/rds-downloads/rds-combined-ca-bundle.p7b>.

Intermediate Certificates

You might need to use an intermediate certificate to connect to your AWS Region. For example, you must use an intermediate certificate to connect to the AWS GovCloud (US-West) Region using SSL/TLS. If you need an intermediate certificate for a particular AWS Region, download the certificate from the following table.

AWS Region	CA-2019	CA-2015
Asia Pacific (Hong Kong)	rds-ca-2019-ap-east-1.pem	No certificate available
Asia Pacific (Mumbai)	rds-ca-2019-ap-south-1.pem	rds-ca-2015-ap-south-1.pem
Asia Pacific (Tokyo)	rds-ca-2019-ap-northeast-1.pem	rds-ca-2015-ap-northeast-1.pem
Asia Pacific (Seoul)	rds-ca-2019-ap-northeast-2.pem	rds-ca-2015-ap-northeast-2.pem
Asia Pacific (Osaka-Local)	rds-ca-2019-ap-northeast-3.pem	rds-ca-2015-ap-northeast-3.pem
Asia Pacific (Singapore)	rds-ca-2019-ap-southeast-1.pem	rds-ca-2015-ap-southeast-1.pem
Asia Pacific (Sydney)	rds-ca-2019-ap-southeast-2.pem	rds-ca-2015-ap-southeast-2.pem
Canada (Central)	rds-ca-2019-ca-central-1.pem	rds-ca-2015-ca-central-1.pem
Europe (Frankfurt)	rds-ca-2019-eu-central-1.pem	rds-ca-2015-eu-central-1.pem
Europe (Ireland)	rds-ca-2019-eu-west-1.pem	rds-ca-2015-eu-west-1.pem
Europe (London)	rds-ca-2019-eu-west-2.pem	rds-ca-2015-eu-west-2.pem
Europe (Paris)	rds-ca-2019-eu-west-3.pem	rds-ca-2015-eu-west-3.pem
Europe (Stockholm)	rds-ca-2019-eu-north-1.pem	rds-ca-2015-eu-north-1.pem
Middle East (Bahrain)	rds-ca-2019-me-south-1.pem	No certificate available

AWS Region	CA-2019	CA-2015
South America (São Paulo)	rds-ca-2019-sa-east-1.pem	rds-ca-2015-sa-east-1.pem
US East (N. Virginia)	rds-ca-2019-us-east-1.pem	rds-ca-2015-us-east-1.pem
US East (Ohio)	rds-ca-2019-us-east-2.pem	rds-ca-2015-us-east-2.pem
US West (N. California)	rds-ca-2019-us-west-1.pem	rds-ca-2015-us-west-1.pem
US West (Oregon)	rds-ca-2019-us-west-2.pem	rds-ca-2015-us-west-2.pem

AWS GovCloud (US) Certificates

You can download a root certificate for the AWS GovCloud (US-West) Region at <https://s3-us-gov-west-1.amazonaws.com/rds-downloads/rds-GovCloud-Root-CA-2017.pem>.

To get a certificate bundle that contains both the intermediate and root certificates for the AWS GovCloud (US) Regions, download from <https://s3-us-gov-west-1.amazonaws.com/rds-downloads/rds-combined-ca-us-gov-bundle.pem>.

You can download the intermediate certificate for an AWS GovCloud (US) Region from the following list:

[AWS GovCloud \(US-East\) \(CA-2017\)](#)

[AWS GovCloud \(US-West\) \(CA-2017\)](#)

[AWS GovCloud \(US-West\) \(CA-2012\)](#)

Rotating Your SSL/TLS Certificate

Note

If your application connects to an RDS DB instance using Secure Socket Layer (SSL) or Transport Layer Security (TLS), you must take the following steps *before March 5, 2020*. Doing this means you can avoid interruption of connectivity between your applications and your RDS DB instances.

As of September 19, 2019, Amazon RDS has published new Certificate Authority (CA) certificates for connecting to your RDS DB instances using SSL/TLS. We provide these 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 \(p. 177\)](#).

The current CA certificates expire on March 5, 2020. Therefore, we strongly recommend completing this change as soon as possible (and no later than February 5, 2020), to avoid disruption on the expiration date. If the change is not completed, your applications will fail to connect to your RDS DB instances using SSL/TLS after March 5, 2020.

We encourage you to test the steps listed following in a development or staging environment before taking them for your production environments.

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.

Any new RDS DB instances created after January 14, 2020 will use the new certificates by default. If you want to temporarily modify new DB instances manually to use the old (rds-ca-2015) certificates, you can do so using the AWS Management Console or the AWS CLI. Any DB instances created prior to January 14, 2020 use the rds-ca-2015 certificates until you update them to the rds-ca-2019 certificates.

Note

If you are using 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 \(p. 118\)](#).

Topics

- [Updating Your CA Certificate by Modifying Your DB Instance \(p. 180\)](#)
- [Updating Your CA Certificate by Applying DB Instance Maintenance \(p. 182\)](#)
- [Reverting an Update of a CA Certificate \(p. 188\)](#)
- [Sample Script for Importing Certificates Into Your Trust Store \(p. 189\)](#)

Updating Your CA Certificate by Modifying Your DB Instance

Complete the following steps to update your CA certificate by modifying your DB instance.

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 \(p. 177\)](#).
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 \(p. 583\)](#)
- [Updating Applications to Connect to Aurora PostgreSQL DB Clusters Using New SSL/TLS Certificates \(p. 883\)](#)

For a sample script that updates a trust store for a Linux operating system, see [Sample Script for Importing Certificates Into Your Trust Store \(p. 189\)](#).

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. Modify the DB instance to change the CA from **rds-ca-2015** to **rds-ca-2019**.

Important

This operation reboots your DB instance. By default, this operation is scheduled to run during your next maintenance window. Or you can choose to run it immediately.

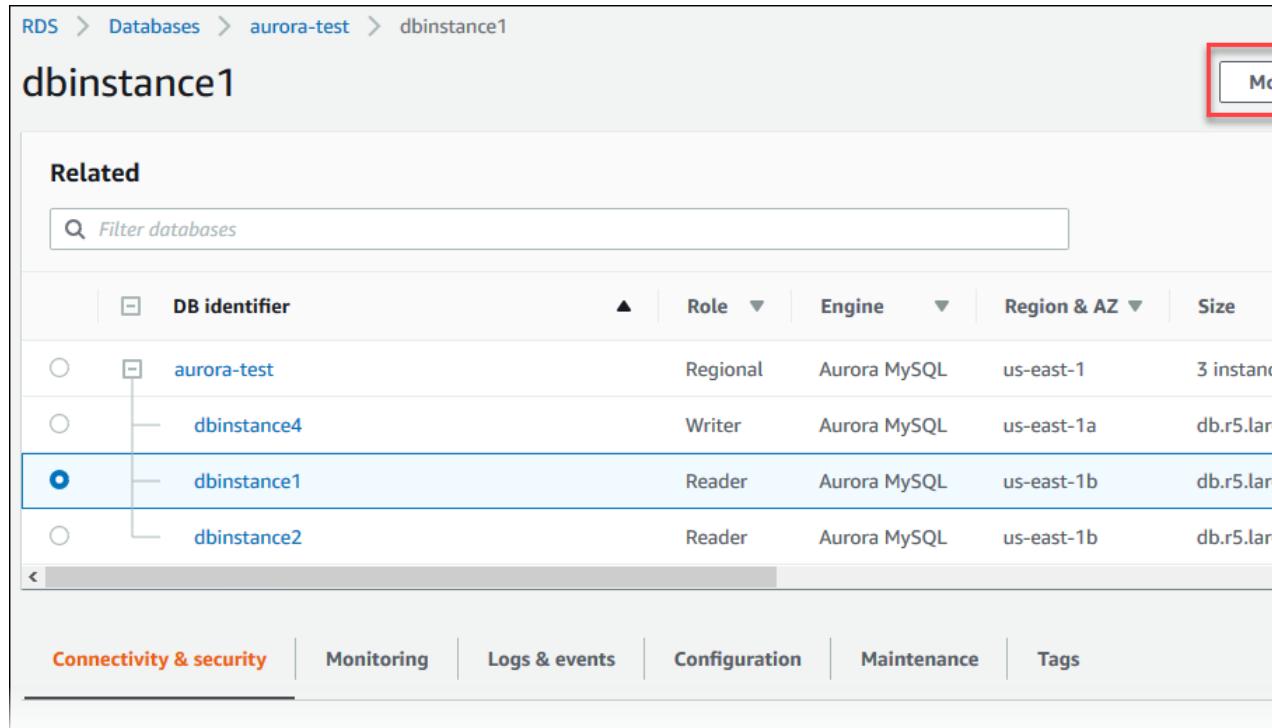
You can use the AWS Management Console or the AWS CLI to change the CA certificate from **rds-ca-2015** to **rds-ca-2019** for a DB instance.

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/>.

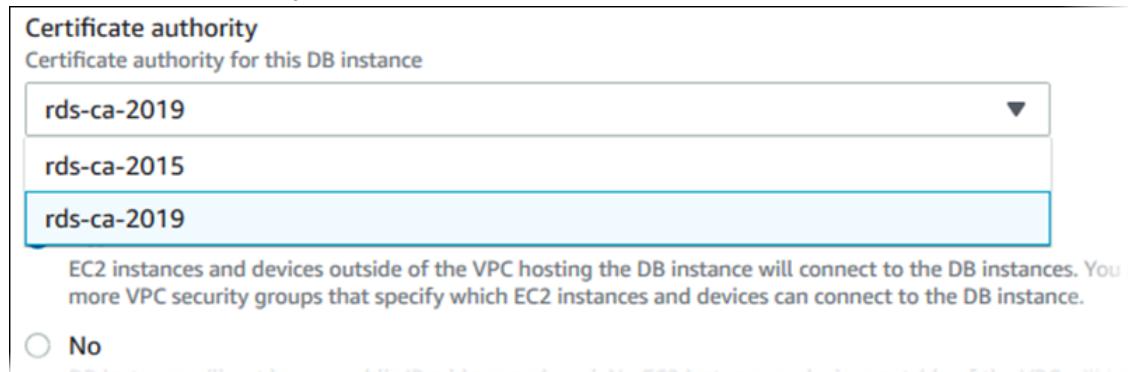
2. In the navigation pane, choose **Databases**, and then choose the DB instance that you want to modify.
3. Choose **Modify**.



The screenshot shows the AWS RDS Modify DB Instance page for a database named 'aurora-test'. The 'dbinstance1' tab is selected. The 'Related' section lists other databases under 'aurora-test': 'dbinstance4' (Writer), 'dbinstance1' (Reader, currently selected), and 'dbinstance2' (Reader). The 'Connectivity & security' tab is active, showing a dropdown menu for 'Certificate authority' with options: 'rds-ca-2019' (selected), 'rds-ca-2015', and 'rds-ca-2019'. Below the dropdown, a note states: '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.' There are also 'No' and 'Yes' radio buttons for this setting. Other tabs include 'Monitoring', 'Logs & events', 'Configuration', 'Maintenance', and 'Tags'.

The **Modify DB Instance** page appears.

4. In the **Network & Security** section, choose **rds-ca-2019**.



The screenshot shows the 'Network & Security' section of the Modify DB Instance page. It displays a dropdown menu for 'Certificate authority' with three options: 'rds-ca-2019' (selected), 'rds-ca-2015', and 'rds-ca-2019'. A note below the dropdown states: '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.' There are also 'No' and 'Yes' radio buttons for this setting.

5. Choose **Continue** and check the summary of modifications.
6. To apply the changes immediately, choose **Apply immediately**.

Important

Choosing this option causes an outage.

7. On the confirmation page, review your changes. If they are correct, choose **Modify DB Instance** to save your changes.

Important

When you schedule this operation, make sure that you have updated your client-side trust store beforehand.

Or choose **Back** to edit your changes or **Cancel** to cancel your changes.

AWS CLI

To use the AWS CLI to change the CA from `rds-ca-2015` to `rds-ca-2019` for a DB instance, call the [modify-db-instance](#) command. Specify the DB instance identifier and the `--ca-certificate-identifier` option.

Important

When you schedule this operation, make sure that you have updated your client-side trust store beforehand.

Example

The following code modifies `mydbinstance` by setting the CA certificate to `rds-ca-2019`. The changes are applied during the next maintenance window by using `--no-apply-immediately`. Use `--apply-immediately` to apply the changes immediately.

Important

Using the `--apply-immediately` option causes an outage.

For Linux, OS X, 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 \(p. 177\)](#).
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 \(p. 583\)](#)
- [Updating Applications to Connect to Aurora PostgreSQL DB Clusters Using New SSL/TLS Certificates \(p. 883\)](#)

For a sample script that updates a trust store for a Linux operating system, see [Sample Script for Importing Certificates Into Your Trust Store \(p. 189\)](#).

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

This operation reboots your DB instance. By default, this operation is scheduled to run during your next maintenance window. Or you can choose to run it immediately.

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 a single DB instance or for multiple DB instances.

Topics

- [Updating Your CA Certificate by Applying Maintenance to a Single DB Instance \(p. 183\)](#)
- [Updating Your CA Certificate by Applying Maintenance to Multiple DB Instances \(p. 184\)](#)

[Updating Your CA Certificate by Applying Maintenance to a Single DB Instance](#)

Use the AWS Management Console to change the CA certificate for a single DB instance.

To change the CA from rds-ca-2015 to rds-ca-2019 for a single 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 name of the DB instance that you want to modify.
3. Choose **Maintenance & backups**.

If your DB instance is using the old CA certificate, the **Pending maintenance** section shows an action with the description **Rotation of CA certificate**. This pending maintenance action is scheduled by default for your maintenance window and before February 5, 2020. However, you can apply the rotation immediately by choosing the pending maintenance action and choosing **Apply now**.

Important

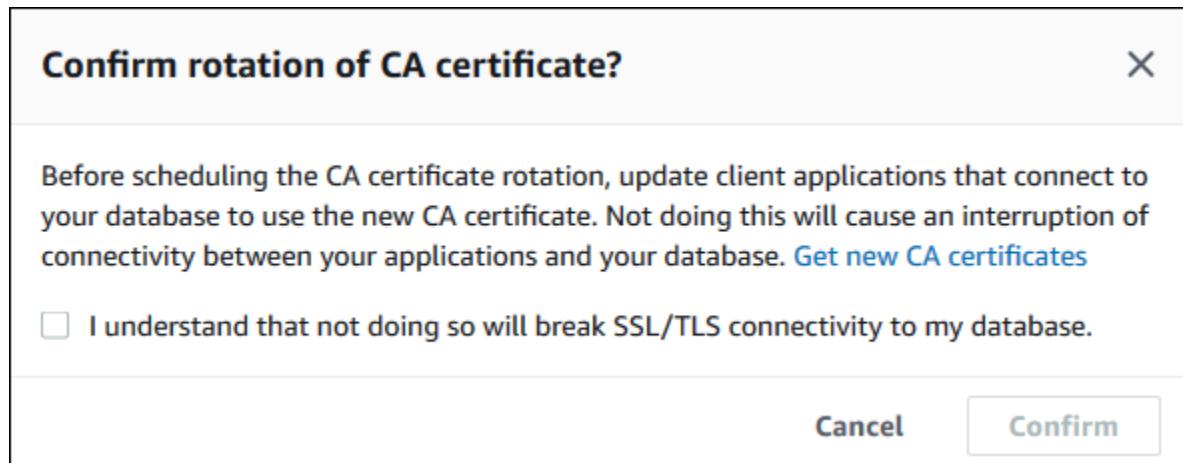
When your CA certificate is rotated, the operation reboots your DB instance.

The screenshot shows the AWS Management Console interface for Amazon Aurora. The top navigation bar includes tabs for Connectivity & security, Monitoring, Logs & events, Configuration, Maintenance & backups (which is highlighted in orange), and Tags. Below the navigation, a section titled 'Maintenance' displays the status of 'Auto minor version upgrade' as 'Enabled'. Under 'Pending maintenance', there is one item: 'Rotation of CA certificate' (Type: ca-certificate-rotation, Status: next window, Apply date: February 4th 2020, 1:54:04 pm UTC-8 (local)). At the bottom of this section are buttons for 'Apply now' (circled in red) and 'Apply at next maintenance window'.

4. If you choose **Apply now** or **Apply at next maintenance window**, 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 \(p. 182\)](#). After you have updated these client applications, you can confirm the CA certificate rotation.



To continue, choose the check box, and then choose **Confirm**.

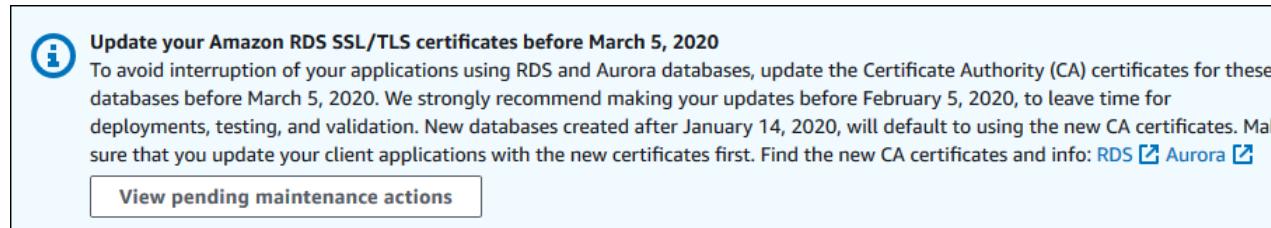
[Updating Your CA Certificate by Applying Maintenance to Multiple DB Instances](#)

Use the AWS Management Console to change the CA certificate for multiple DB instances.

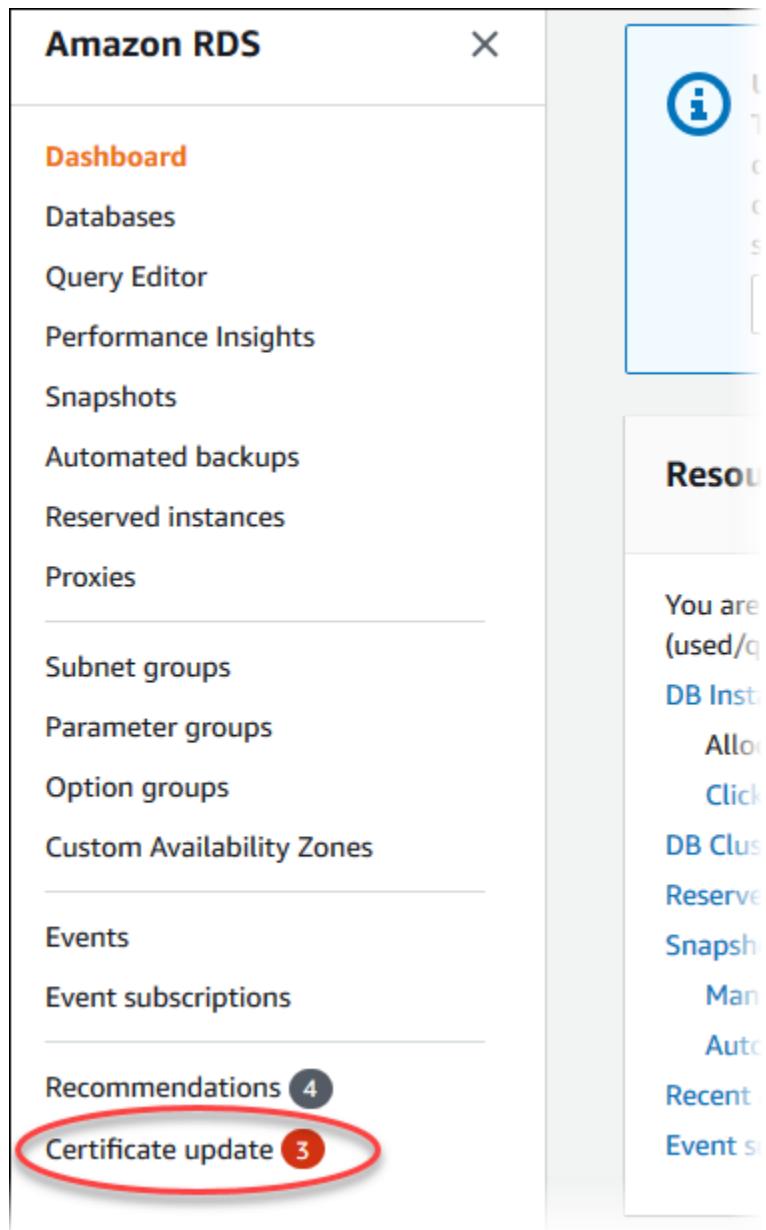
To change the CA from rds-ca-2015 to rds-ca-2019 for multiple DB instances

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**.

If you have at least one DB instance that is using the old CA certificate, the following banner appears at the top of the page.



In the navigation pane, there is also a **Certificate update** option that shows the total number of affected DB instances.



Either choose **View pending maintenance actions** in the banner, or 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**. Or you can apply the rotation immediately by choosing **Update now**.

Important

When your CA certificate is rotated, the operation reboots your DB instance.

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 \(p. 182\)](#). 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.

Reverting an Update of a CA Certificate

You can use the AWS Management Console or the AWS CLI to revert to a previous CA certificate for a DB instance.

Console

To revert to a previous CA certificate 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 'dbinstance1' page in the AWS RDS console. At the top, the navigation path is RDS > Databases > aurora-test > dbinstance1. Below the path, the DB identifier 'dbinstance1' is displayed. On the right, there is a red box highlighting the 'Modify' button. The main area shows 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 for Connectivity & security, Monitoring, Logs & events, Configuration, Maintenance, and Tags. The 'Connectivity & security' tab is selected.

The **Modify DB Instance** page appears.

4. In the **Network & Security** section, choose **rds-ca-2015**.

The screenshot shows the 'Certificate authority' section of the 'Modify DB Instance' page. The heading is 'Certificate authority for this DB instance'. A dropdown menu is open, showing three options: 'rds-ca-2015' (which is highlighted with a blue border), 'rds-ca-2015' (disabled, shown in gray), and 'rds-ca-2019'. Below the dropdown, there is a note: 'EC2 instances and devices outside of the VPC hosting the DB instance will connect to the DB instances. You must add more VPC security groups that specify which EC2 instances and devices can connect to the DB instance.' There is also a radio button labeled 'No'.

5. Choose **Continue** and check the summary of modifications.
6. To apply the changes immediately, choose **Apply immediately**.

Important

Choosing this option causes an outage.

7. On the confirmation page, review your changes. If they are correct, choose **Modify DB Instance** to save your changes.

Important

When you schedule this operation, make sure that you have updated your client-side trust store beforehand.

Or choose **Back** to edit your changes or **Cancel** to cancel your changes.

AWS CLI

To revert to a previous CA certificate 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 example modifies `mydbinstance` by setting the CA certificate to `rds-ca-2015`. The changes are applied during the next maintenance window by using `--no-apply-immediately`. Use `--apply-immediately` to apply the changes immediately.

Important

Using the `--apply-immediately` option causes an outage.

For Linux, OS X, or Unix:

```
aws rds modify-db-instance \
--db-instance-identifier mydbinstance \
--ca-certificate-identifier rds-ca-2015 \
--no-apply-immediately
```

For Windows:

```
aws rds modify-db-instance ^
--db-instance-identifier mydbinstance ^
--ca-certificate-identifier rds-ca-2015 ^
--no-apply-immediately
```

Sample Script for Importing Certificates Into Your Trust Store

The following is a sample shell script that imports the certificate bundle into a trust store on a Linux operating system.

```
mydir=/tmp/certs
truststore=${mydir}/rds-truststore.jks
storepassword=changeit

curl -sS "https://s3.amazonaws.com/rds-downloads/rds-combined-ca-bundle.pem" > ${mydir}/
rds-combined-ca-bundle.pem
awk 'split_after == 1 {n++;split_after=0} /-----END CERTIFICATE-----/ {split_after=1}{print
> "rds-ca-" n ".pem"}' < ${mydir}/rds-combined-ca-bundle.pem

for CERT in rds-ca-*; do
```

```
alias=$(openssl x509 -noout -text -in $CERT | perl -ne 'next unless /Subject:/; s/.*CN=//; print')
echo "Importing $alias"
keytool -import -file ${CERT} -alias "${alias}" -storepass ${storepassword} -keystore ${truststore} -noprompt
rm $CERT
done

rm ${mydir}/rds-combined-ca-bundle.pem

echo "Trust store content is:"

keytool -list -v -keystore "$truststore" -storepass ${storepassword} | grep Alias | cut -d
" " -f3- | while read alias
do
    expiry=`keytool -list -v -keystore "$truststore" -storepass ${storepassword} -alias
"${alias}" | grep Valid | perl -ne 'if(/until: (.*)\n/) { print "$1\n"; }'` 
    echo " Certificate ${alias} expires in '$expiry'"
done
```

Internetwork Traffic Privacy

Connections are protected both between Amazon Aurora and on-premises applications and between Amazon Aurora and other AWS resources within the same AWS Region.

Traffic Between Service and On-Premises Clients and Applications

You have two connectivity options between your private network and AWS:

- An AWS Site-to-Site VPN connection. For more information, see [What is AWS Site-to-Site VPN?](#)
- An AWS Direct Connect connection. For more information, see [What is AWS Direct Connect?](#)

You get access to Amazon Aurora through the network by using AWS-published API operations. Clients must support Transport Layer Security (TLS) 1.0. We recommend TLS 1.2. Clients must also support cipher suites with Perfect Forward Secrecy (PFS), such as Ephemeral Diffie-Hellman (DHE) or Elliptic Curve Diffie-Hellman Ephemeral (ECDHE). Most modern systems such as Java 7 and later support these modes. Additionally, you must sign requests using an access key ID 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.

Traffic Between AWS Resources in the Same Region

An Amazon Virtual Private Cloud (Amazon VPC) endpoint for Amazon Aurora is a logical entity within a VPC that allows connectivity only to Amazon Aurora. The Amazon VPC routes requests to Amazon Aurora and routes responses back to the VPC. For more information, see [VPC Endpoints](#) in the *Amazon VPC User Guide*. For sample policies that you can use to control DB cluster access from VPC endpoints, see [Creating and Using an IAM Policy for IAM Database Access \(p. 209\)](#).

Identity and Access Management in Amazon Aurora

AWS Identity and Access Management (IAM) is an AWS service that helps an administrator securely control access to AWS resources. IAM administrators control who can be *authenticated* (signed in) and *authorized* (have permissions) to use Aurora resources. IAM is an AWS service that you can use with no additional charge.

Topics

- [Audience \(p. 191\)](#)
- [Authenticating With Identities \(p. 191\)](#)
- [Managing Access Using Policies \(p. 193\)](#)
- [How Amazon Aurora Works with IAM \(p. 195\)](#)
- [Amazon Aurora Identity-Based Policy Examples \(p. 197\)](#)
- [IAM Database Authentication \(p. 207\)](#)
- [Troubleshooting Amazon Aurora Identity and Access \(p. 223\)](#)

Audience

How you use AWS Identity and Access Management (IAM) differs, depending on the work you do in Aurora.

Service user – If you use the Aurora service to do your job, then your administrator provides you with the credentials and permissions that you need. As you use more Aurora features to do your work, you might need additional permissions. Understanding how access is managed can help you request the right permissions from your administrator. If you cannot access a feature in Aurora, see [Troubleshooting Amazon Aurora Identity and Access \(p. 223\)](#).

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 \(p. 195\)](#).

IAM administrator – If you're an IAM administrator, you might want to learn details about how you can write policies to manage access to Aurora. To view example Aurora identity-based policies that you can use in IAM, see [Amazon Aurora Identity-Based Policy Examples \(p. 197\)](#).

Authenticating With Identities

Authentication is how you sign in to AWS using your identity credentials. For more information about signing in using the AWS Management Console, see [The IAM Console and Sign-in Page](#) in the *IAM User Guide*.

You must be *authenticated* (signed in to AWS) as the AWS account root user, an IAM user, or by assuming an IAM role. You can also use your company's single sign-on authentication, or even sign in using Google or Facebook. In these cases, your administrator previously set up identity federation using IAM roles. When you access AWS using credentials from another company, you are assuming a role indirectly.

To sign in directly to the [AWS Management Console](#), use your password with your root user email or your IAM user name. You can access AWS programmatically using your root user or IAM user access keys. AWS

provides SDK and command line tools to cryptographically sign your request using your credentials. If you don't use AWS tools, you must sign the request yourself. Do this using *Signature Version 4*, a protocol for authenticating inbound API requests. For more information about authenticating requests, see [Signature Version 4 Signing Process](#) in the *AWS General Reference*.

Regardless of the authentication method that you use, you might also be required to provide additional security information. For example, AWS recommends that you use multi-factor authentication (MFA) to increase the security of your account. To learn more, see [Using Multi-Factor Authentication \(MFA\) in AWS](#) in the *IAM User Guide*.

AWS Account Root User

When you first create an AWS account, you begin with a single sign-in identity that has complete access to all AWS services and resources in the account. This identity is called the AWS account *root user* and is accessed by signing in with the email address and password that you used to create the account. We strongly recommend that you do not use the root user for your everyday tasks, even the administrative ones. Instead, adhere to the [best practice of using the root user only to create your first IAM user](#). Then securely lock away the root user credentials and use them to perform only a few account and service management tasks.

IAM Users and Groups

An *IAM user* is an identity within your AWS account that has specific permissions for a single person or application. An IAM user can have long-term credentials such as a user name and password or a set of access keys. To learn how to generate access keys, see [Managing Access Keys for IAM Users](#) in the *IAM User Guide*. When you generate access keys for an IAM user, make sure you view and securely save the key pair. You cannot recover the secret access key in the future. Instead, you must generate a new access key pair.

An *IAM group* is an identity that specifies a collection of IAM users. You can't sign in as a group. You can use groups to specify permissions for multiple users at a time. Groups make permissions easier to manage for large sets of users. For example, you could have a group named *IAMAdmins* and give that group permissions to administer IAM resources.

Users are different from roles. A user is uniquely associated with one person or application, but a role is intended to be assumable by anyone who needs it. Users have permanent long-term credentials, but roles provide temporary credentials. To learn more, see [When to Create an IAM User \(Instead of a Role\)](#) in the *IAM User Guide*.

You can authenticate to your DB cluster using IAM database authentication.

IAM database authentication works with Aurora. For more information about authenticating to your DB cluster using IAM, see [IAM Database Authentication \(p. 207\)](#).

IAM Roles

An *IAM role* is an identity within your AWS account that has specific permissions. It is similar to an IAM user, but is not associated with a specific person. You can temporarily assume an IAM role in the AWS Management Console by [switching roles](#). You can assume a role by calling an AWS CLI or AWS API operation or by using a custom URL. For more information about methods for using roles, see [Using IAM Roles](#) in the *IAM User Guide*.

IAM roles with temporary credentials are useful in the following situations:

- **Temporary IAM user permissions** – An IAM user can assume an IAM role to temporarily take on different permissions for a specific task.
- **Federated user access** – Instead of creating an IAM user, you can use existing identities from AWS Directory Service, your enterprise user directory, or a web identity provider. These are known as

federated users. AWS assigns a role to a federated user when access is requested through an [identity provider](#). For more information about federated users, see [Federated Users and Roles](#) in the *IAM User Guide*.

- **Cross-account access** – You can use an IAM role to allow someone (a trusted principal) in a different account to access resources in your account. Roles are the primary way to grant cross-account access. However, with some AWS services, you can attach a policy directly to a resource (instead of using a role as a proxy). To learn the difference between roles and resource-based policies for cross-account access, see [How IAM Roles Differ from Resource-based Policies](#) in the *IAM User Guide*.
- **AWS service access** – A service role is an IAM role that a service assumes to perform actions in your account on your behalf. When you set up some AWS service environments, you must define a role for the service to assume. This service role must include all the permissions that are required for the service to access the AWS resources that it needs. Service roles vary from service to service, but many allow you to choose your permissions as long as you meet the documented requirements for that service. Service roles provide access only within your account and cannot be used to grant access to services in other accounts. You can create, modify, and delete a service role from within IAM. For example, you can create a role that allows Amazon Redshift to access an Amazon S3 bucket on your behalf and then load data from that bucket into an Amazon Redshift cluster. For more information, see [Creating a Role to Delegate Permissions to an AWS Service](#) in the *IAM User Guide*.
- **Applications running on Amazon EC2** – You can use an IAM role to manage temporary credentials for applications that are running on an EC2 instance and making AWS CLI or AWS API requests. This is preferable to storing access keys within the EC2 instance. To assign an AWS role to an EC2 instance and make it available to all of its applications, you create an instance profile that is attached to the instance. An instance profile contains the role and enables programs that are running on the EC2 instance to get temporary credentials. For more information, see [Using an IAM Role to Grant Permissions to Applications Running on Amazon EC2 Instances](#) in the *IAM User Guide*.

To learn whether to use IAM roles, see [When to Create an IAM Role \(Instead of a User\)](#) in the *IAM User Guide*.

Managing Access Using Policies

You control access in AWS by creating policies and attaching them to IAM identities or AWS resources. A policy is an object in AWS that, when associated with an identity or resource, defines their permissions. AWS evaluates these policies when an entity (root user, IAM user, or IAM role) makes a request. Permissions in the policies determine whether the request is allowed or denied. Most policies are stored in AWS as JSON documents. For more information about the structure and contents of JSON policy documents, see [Overview of JSON Policies](#) in the *IAM User Guide*.

An IAM administrator can use policies to specify who has access to AWS resources, and what actions they can perform on those resources. Every IAM entity (user or role) starts with no permissions. In other words, by default, users can do nothing, not even change their own password. To give a user permission to do something, an administrator must attach a permissions policy to a user. Or the administrator can add the user to a group that has the intended permissions. When an administrator gives permissions to a group, all users in that group are granted those permissions.

IAM policies define permissions for an action regardless of the method that you use to perform the operation. For example, suppose that you have a policy that allows the `iam:GetRole` action. A user with that policy can get role information from the AWS Management Console, the AWS CLI, or the AWS API.

Identity-Based Policies

Identity-based policies are JSON permissions policy documents that you can attach to an identity, such as an IAM user, role, or group. These policies control what actions that identity can perform, on which resources, and under what conditions. To learn how to create an identity-based policy, see [Creating IAM Policies](#) in the *IAM User Guide*.

Identity-based policies can be further categorized as *inline policies* or *managed policies*. Inline policies are embedded directly into a single user, group, or role. Managed policies are standalone policies that you can attach to multiple users, groups, and roles in your AWS account. Managed policies include AWS managed policies and customer managed policies. To learn how to choose between a managed policy or an inline policy, see [Choosing Between Managed Policies and Inline Policies](#) in the *IAM User Guide*.

The following AWS managed policies, which you can attach to users in your account, are specific to Amazon Aurora:

- **AmazonRDSReadOnlyAccess** – Grants read-only access to all Amazon Aurora resources for the AWS account specified.
- **AmazonRDSFullAccess** – Grants full access to all Amazon Aurora resources for the AWS account specified.

Other Policy Types

AWS supports additional, less-common policy types. These policy types can set the maximum permissions granted to you by the more common policy types.

- **Permissions boundaries** – A permissions boundary is an advanced feature in which you set the maximum permissions that an identity-based policy can grant to an IAM entity (IAM user or role). You can set a permissions boundary for an entity. The resulting permissions are the intersection of entity's identity-based policies and its permissions boundaries. Resource-based policies that specify the user or role in the `Principal` field are not limited by the permissions boundary. An explicit deny in any of these policies overrides the allow. For more information about permissions boundaries, see [Permissions Boundaries for IAM Entities](#) in the *IAM User Guide*.
- **Service control policies (SCPs)** – SCPs are JSON policies that specify the maximum permissions for an organization or organizational unit (OU) in AWS Organizations. AWS Organizations is a service for grouping and centrally managing multiple AWS accounts that your business owns. If you enable all features in an organization, then you can apply service control policies (SCPs) to any or all of your accounts. The SCP limits permissions for entities in member accounts, including each AWS account root user. For more information about Organizations and SCPs, see [How SCPs Work](#) in the *AWS Organizations User Guide*.
- **Session policies** – Session policies are advanced policies that you pass as a parameter when you programmatically create a temporary session for a role or federated user. The resulting session's permissions are the intersection of the user or role's identity-based policies and the session policies. Permissions can also come from a resource-based policy. An explicit deny in any of these policies overrides the allow. For more information, see [Session Policies](#) in the *IAM User Guide*.

Multiple Policy Types

When multiple types of policies apply to a request, the resulting permissions are more complicated to understand. To learn how AWS determines whether to allow a request when multiple policy types are involved, see [Policy Evaluation Logic](#) in the *IAM User Guide*.

For more information about identity and access management for Aurora, continue to the following pages:

- [How Amazon Aurora Works with IAM \(p. 195\)](#)
- [Troubleshooting Amazon Aurora Identity and Access \(p. 223\)](#)

How Amazon Aurora Works with IAM

Before you use IAM to manage access to Aurora, you should understand what IAM features are available to use with Aurora. To get a high-level view of how Aurora and other AWS services work with IAM, see [AWS Services That Work with IAM](#) in the *IAM User Guide*.

Topics

- [Aurora Identity-Based Policies \(p. 195\)](#)
- [Aurora Resource-Based Policies \(p. 197\)](#)
- [Authorization Based on Aurora Tags \(p. 197\)](#)
- [Aurora IAM Roles \(p. 197\)](#)

Aurora Identity-Based Policies

With IAM identity-based policies, you can specify allowed or denied actions and resources as well as the conditions under which actions are allowed or denied. Aurora supports specific actions, resources, and condition keys. To learn about all of the elements that you use in a JSON policy, see [IAM JSON Policy Elements Reference](#) in the *IAM User Guide*.

Actions

The `Action` element of an IAM identity-based policy describes the specific action or actions that will be allowed or denied by the policy. Policy actions usually have the same name as the associated AWS API operation. The action is used in a policy to grant permissions to perform the associated operation.

Policy actions in Aurora use the following prefix before the action: `rds:`. For example, to grant someone permission to describe DB instances with the Amazon RDS `DescribeDBInstances` API operation, you include the `rds:DescribeDBInstances` action in their policy. Policy statements must include either an `Action` or `NotAction` element. Aurora defines its own set of actions that describe tasks that you can perform with this service.

To specify multiple actions in a single statement, separate them with commas as follows:

```
"Action": [  
    "rds:action1",  
    "rds:action2"]
```

You can specify multiple actions using wildcards (*). For example, to specify all actions that begin with the word `Describe`, include the following action:

```
"Action": "rds:Describe*"
```

To see a list of Aurora actions, see [Actions Defined by Amazon RDS](#) in the *IAM User Guide*.

Resources

The `Resource` element specifies the object or objects to which the action applies. Statements must include either a `Resource` or a `NotResource` element. You specify a resource using an ARN or using the wildcard (*) to indicate that the statement applies to all resources.

The DB instance resource has the following ARN:

```
arn:${Partition}:rds:${Region}:${Account}:{ ResourceType }/${Resource}
```

For more information about the format of ARNs, see [Amazon Resource Names \(ARNs\) and AWS Service Namespaces](#).

For example, to specify the dbtest DB instance in your statement, use the following ARN:

```
"Resource": "arn:aws:rds:us-west-2:123456789012:db:dbtest"
```

To specify all DB instances that belong to a specific account, use the wildcard (*):

```
"Resource": "arn:aws:ec2:us-east-1:123456789012:db:/*"
```

Some RDS API operations, such as those for creating resources, cannot be performed on a specific resource. In those cases, you must use the wildcard (*).

```
"Resource": "*"
```

Many Amazon RDS API operations involve multiple resources. For example, `CreateDBInstance` creates a DB instance. You can specify that an IAM user must use a specific security group and parameter group when creating a DB instance. To specify multiple resources in a single statement, separate the ARNs with commas.

```
"Resource": [  
    "resource1",  
    "resource2"]
```

To see a list of Aurora resource types and their ARNs, see [Resources Defined by Amazon RDS](#) in the *IAM User Guide*. To learn with which actions you can specify the ARN of each resource, see [Actions Defined by Amazon RDS](#).

Condition Keys

The Condition element (or Condition block) lets you specify conditions in which a statement is in effect. The Condition element is optional. You can build conditional expressions that use [condition operators](#), such as equals or less than, to match the condition in the policy with values in the request.

If you specify multiple Condition elements in a statement, or multiple keys in a single Condition element, AWS evaluates them using a logical AND operation. If you specify multiple values for a single condition key, AWS evaluates the condition using a logical OR operation. All of the conditions must be met before the statement's permissions are granted.

You can also use placeholder variables when you specify conditions. For example, you can grant an IAM user permission to access a resource only if it is tagged with their IAM user name. For more information, see [IAM Policy Elements: Variables and Tags](#) in the *IAM User Guide*.

Aurora defines its own set of condition keys and also supports using some global condition keys. To see all AWS global condition keys, see [AWS Global Condition Context Keys](#) in the *IAM User Guide*.

All RDS API operations support the `aws:RequestedRegion` condition key.

To see a list of Aurora condition keys, see [Condition Keys for Amazon RDS](#) in the *IAM User Guide*. To learn with which actions and resources you can use a condition key, see [Actions Defined by Amazon RDS](#).

Examples

To view examples of Aurora identity-based policies, see [Amazon Aurora Identity-Based Policy Examples \(p. 197\)](#).

Aurora Resource-Based Policies

Aurora does not support resource-based policies.

Authorization Based on Aurora Tags

You can attach tags to Aurora resources or pass tags in a request to Aurora. To control access based on tags, you provide tag information in the [condition element](#) of a policy using the `rds:ResourceTag/key-name`, `aws:RequestTag/key-name`, or `aws:TagKeys` condition keys. For more information about tagging Aurora resources, see [Specifying Conditions: Using Custom Tags \(p. 204\)](#).

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 \(p. 201\)](#).

Aurora IAM Roles

An [IAM role](#) is an entity within your AWS account that has specific permissions.

Using Temporary Credentials with Aurora

You can use temporary credentials to sign in with federation, assume an IAM role, or to assume a cross-account role. You obtain temporary security credentials by calling AWS STS API operations such as [AssumeRole](#) or [GetFederationToken](#).

Aurora supports using temporary credentials.

Service-Linked Roles

[Service-linked roles](#) allow AWS services to access resources in other services to complete an action on your behalf. Service-linked roles appear in your IAM account and are owned by the service. An IAM administrator can view but not edit the permissions for service-linked roles.

Aurora supports service-linked roles. For details about creating or managing Aurora service-linked roles, see [Using Service-Linked Roles for Amazon Aurora \(p. 235\)](#).

Service Roles

This feature allows a service to assume a [service role](#) on your behalf. This role allows the service to access resources in other services to complete an action on your behalf. Service roles appear in your IAM account and are owned by the account. This means that an IAM administrator can change the permissions for this role. However, doing so might break the functionality of the service.

Aurora supports service roles.

Amazon Aurora Identity-Based Policy Examples

By default, IAM users and roles don't have permission to create or modify Aurora resources. They also can't perform tasks using the AWS Management Console, AWS CLI, or AWS API. An IAM administrator must create IAM policies that grant users and roles permission to perform specific API operations on the specified resources they need. The administrator must then attach those policies to the IAM users or groups that require those permissions.

To learn how to create an IAM identity-based policy using these example JSON policy documents, see [Creating Policies on the JSON Tab](#) in the [IAM User Guide](#).

Topics

- [Policy Best Practices \(p. 198\)](#)
- [Using the Aurora Console \(p. 198\)](#)
- [Allow Users to View Their Own Permissions \(p. 199\)](#)
- [Allow a User to Create DB Instances in an AWS account \(p. 199\)](#)
- [Permissions Required to Use the Console \(p. 200\)](#)
- [Allow a User to Perform Any Describe Action on Any RDS Resource \(p. 201\)](#)
- [Allow a User to Create a DB Instance That Uses the Specified DB Parameter and Security Groups \(p. 201\)](#)
- [Grant Permission for Actions on a Resource with a Specific Tag with Two Different Values \(p. 201\)](#)
- [Prevent a User from Deleting a DB Instance \(p. 202\)](#)
- [Example Policies: Using Condition Keys \(p. 202\)](#)
- [Specifying Conditions: Using Custom Tags \(p. 204\)](#)

Policy Best Practices

Identity-based policies are very powerful. They determine whether someone can create, access, or delete Aurora resources in your account. These actions can incur costs for your AWS account. When you create or edit identity-based policies, follow these guidelines and recommendations:

- **Get Started Using AWS Managed Policies** – To start using Aurora quickly, use AWS managed policies to give your employees the permissions they need. These policies are already available in your account and are maintained and updated by AWS. For more information, see [Get Started Using Permissions With AWS Managed Policies](#) in the *IAM User Guide*.
- **Grant Least Privilege** – When you create custom policies, grant only the permissions required to perform a task. Start with a minimum set of permissions and grant additional permissions as necessary. Doing so is more secure than starting with permissions that are too lenient and then trying to tighten them later. For more information, see [Grant Least Privilege](#) in the *IAM User Guide*.
- **Enable MFA for Sensitive Operations** – For extra security, require IAM users to use multi-factor authentication (MFA) to access sensitive resources or API operations. For more information, see [Using Multi-Factor Authentication \(MFA\) in AWS](#) in the *IAM User Guide*.
- **Use Policy Conditions for Extra Security** – To the extent that it's practical, define the conditions under which your identity-based policies allow access to a resource. For example, you can write conditions to specify a range of allowable IP addresses that a request must come from. You can also write conditions to allow requests only within a specified date or time range, or to require the use of SSL or MFA. For more information, see [IAM JSON Policy Elements: Condition](#) in the *IAM User Guide*.

Using the Aurora Console

To access the Amazon Aurora console, you must have a minimum set of permissions. These permissions must enable you to list and view details about the Aurora resources in your AWS account. You can create an identity-based policy that is more restrictive than the minimum required permissions. However, if you do, the console doesn't function as intended for entities (IAM users or roles) with that policy.

To ensure that those entities can still use the Aurora console, also attach the following AWS managed policy to the entities. For more information, see [Adding Permissions to a User](#) in the *IAM User Guide*.

AmazonRDSReadOnlyAccess

You don't need to allow minimum console permissions for users that are making calls only to the AWS CLI or the AWS API. Instead, allow access to only the actions that match the API operation that you're trying to perform.

Allow Users to View Their Own Permissions

This example shows how you might create a policy that allows IAM users to view the inline and managed policies that are attached to their user identity. This policy includes permissions to complete this action on the console or programmatically using the AWS CLI or AWS API.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ViewOwnUserInfo",
            "Effect": "Allow",
            "Action": [
                "iam:GetUserPolicy",
                "iam>ListGroupsForUser",
                "iam>ListAttachedUserPolicies",
                "iam>ListUserPolicies",
                "iam GetUser"
            ],
            "Resource": [
                "arn:aws:iam::*:user/${aws:username}"
            ]
        },
        {
            "Sid": "NavigateInConsole",
            "Effect": "Allow",
            "Action": [
                "iam:GetGroupPolicy",
                "iam:GetPolicyVersion",
                "iam GetPolicy",
                "iam>ListAttachedGroupPolicies",
                "iam>ListGroupPolicies",
                "iam>ListPolicyVersions",
                "iam>ListPolicies",
                "iam>ListUsers"
            ],
            "Resource": "*"
        }
    ]
}
```

Allow a User to Create to 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 \(p. 425\)](#).

The `Resource` element in the example specifies the following policy constraints on resources for the user:

- The DB instance identifier for the new DB instance must begin with `test` (for example, `testCustomerData1`, `test-region2-data`).
- The option group for the new DB instance must begin with `default`.
- The DB parameter group for the new DB instance must begin with `default`.
- The subnet group for the new DB instance must be the `default` subnet group.
- The `Condition` element specifies that the DB engine must be MySQL and the DB instance class must be db.t2.micro. The `Condition` element specifies the conditions when a policy should take effect. You can add additional permissions or restrictions by using the `Condition` element. For more information about specifying conditions, see [Condition Keys \(p. 196\)](#). 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 [Hardware Specifications for All Available DB Instance Classes for Aurora \(p. 77\)](#).

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 *IAM User Guide*.

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

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 and Security Groups

The following permissions policy grants permissions to allow a user to only create a DB instance that must use the `mysql-production` DB parameter group and the `db-production` DB security group.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowMySQLProductionCreate",
            "Effect": "Allow",
            "Action": "rds>CreateDBInstance",
            "Resource": [
                "arn:aws:rds:us-west-2:123456789012:pg:mysql-production",
                "arn:aws:rds:us-west-2:123456789012:secgrp:db-production"
            ]
        }
    ]
}
```

Grant Permission for Actions on a Resource with a Specific Tag with Two Different Values

You can use conditions in your identity-based policy to control access to Aurora resources based on tags. The following policy allows permission to perform the `ModifyDBInstance` and `CreateDBSnapshot` APIs on DB instances with either the `stage` tag set to `development` or `test`.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowDevTestCreate",
            "Effect": "Allow",
            "Action": [
                "rds:ModifyDBInstance",
                "rds>CreateDBSnapshot"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "rds:db-tag/stage": [
                        "development",
                        "test"
                    ]
                }
            }
        }
    ]
}
```

Prevent a User from Deleting a DB Instance

The following permissions policy grants permissions to prevent a user from deleting a specific DB instance. For example, you might want to deny the ability to delete your production DB instances to any user that is not an administrator.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "DenyDelete1",
            "Effect": "Deny",
            "Action": "rds>DeleteDBInstance",
            "Resource": "arn:aws:rds:us-west-2:123456789012:db:my-mysql-instance"
        }
    ]
}
```

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>CreateDBSnapshot"
        }
    ]
}
```

```

    "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 to 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": {
                "NumericNotEquals": {
                    "rds:Piops": "0"
                }
            }
        }
    ]
}

```

Example 3: Limit the Set of Tag Keys and Values That Can Be Used to Tag a Resource

The following policy uses an RDS condition key and allows the addition of a tag with the key `stage` to be added to a resource with the values `test`, `qa`, and `production`.

```
{
  {
    "Version" : "2012-10-17",
    "Statement" : [
      {
        "Effect" : "Allow",
        "Action" : [ "rds:AddTagsToResource", "rds:RemoveTagsFromResource" ],
        "Resource" : "*",
        "Condition" : { "streq" : { "rds:req-tag/stage" : [ "test", "qa", "production" ] } }
      }
    ]
  }
}
```

Specifying Conditions: Using Custom Tags

Amazon Aurora supports specifying conditions in an IAM policy using custom tags.

For example, suppose that you add a tag named `environment` to your DB instances with values such as `beta`, `staging`, `production`, and so on. If you do, you can create a policy that restricts certain users to DB instances based on the `environment` tag value.

Note

Custom tag identifiers are case-sensitive.

The following table lists the RDS tag identifiers that you can use in a `Condition` element.

RDS Tag Identifier	Applies To
<code>db-tag</code>	DB instances, including Read Replicas
<code>snapshot-tag</code>	DB snapshots
<code>ri-tag</code>	Reserved DB instances
<code>secgrp-tag</code>	DB security groups
<code>og-tag</code>	DB option groups
<code>pg-tag</code>	DB parameter groups
<code>subgrp-tag</code>	DB subnet groups
<code>es-tag</code>	Event subscriptions
<code>cluster-tag</code>	DB clusters
<code>cluster-pg-tag</code>	DB cluster parameter groups
<code>cluster-snapshot-tag</code>	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 \(p. 420\)](#).

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 *IAM User Guide*.

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 \(p. 425\)](#).

Note

All examples use the us-west-2 region and contain fictitious account IDs.

Example 1: Grant Permission for Actions on a Resource with a Specific Tag with Two Different Values

The following policy allows permission to perform the `ModifyDBInstance` and `CreateDBSnapshot` APIs on DB instances with either the `stage` tag set to `development` or `test`.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "AllowDevTestCreate",  
            "Effect": "Allow",  
            "Action": [  
                "rds:ModifyDBInstance",  
                "rds>CreateDBSnapshot"  
            ],  
            "Resource": "*",  
            "Condition": {  
                "StringEquals": {  
                    "rds:db-tag/stage": [  
                        "development",  
                        "test"  
                    ]  
                }  
            }  
        }  
    ]  
}
```

```
    ]
}
```

Example 2: Explicitly Deny Permission to Create a DB Instance that Uses Specified DB Parameter Groups

The following policy explicitly denies permission to create a DB instance that uses DB parameter groups with specific tag values. You might apply this policy if you require that a specific customer-created DB parameter group always be used when creating DB instances. Policies that use Deny are most often used to restrict access that was granted by a broader policy.

Explicitly denying permission supersedes any other permissions granted. This ensures that identities do 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 \(p. 425\)](#).

```
{
  "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"
        }
      }
    }
  ]
}
```

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.

IAM database authentication provides the following benefits:

- Network traffic to and from the database is encrypted using Secure Sockets Layer (SSL).
- You can use IAM to centrally manage access to your database resources, instead of managing access individually on each DB cluster.
- For applications running on Amazon EC2, you can use profile credentials specific to your EC2 instance to access your database instead of a password, for greater security.

Topics

- [Availability for IAM Database Authentication \(p. 207\)](#)
- [MySQL Limitations for IAM Database Authentication \(p. 207\)](#)
- [PostgreSQL Limitations for IAM Database Authentication \(p. 208\)](#)
- [Enabling and Disabling IAM Database Authentication \(p. 208\)](#)
- [Creating and Using an IAM Policy for IAM Database Access \(p. 209\)](#)
- [Creating a Database Account Using IAM Authentication \(p. 212\)](#)
- [Connecting to Your DB Cluster Using IAM Authentication \(p. 213\)](#)

Availability for IAM Database Authentication

IAM database authentication is available for the following database engines and DB instance classes:

- Aurora with MySQL compatibility version 1.10 or higher. All DB instance classes are supported, except for db.t2.small and db.t3.small.
- Aurora with PostgreSQL compatibility, PostgreSQL versions 9.6.9 and 10.4 or higher.

MySQL Limitations for IAM Database Authentication

When using IAM database authentication with Aurora MySQL, you are limited to a maximum of 200 new connections per second.

The database engines that work with Amazon Aurora don't impose any limits on authentication attempts per second. However, when you use IAM database authentication, your application must generate an authentication token. Your application then uses that token to connect to the DB cluster. If you exceed the limit of maximum new connections per second, then the extra overhead of IAM database authentication can cause connection throttling. The extra overhead can cause even existing connections to drop. For information about the maximum total connections for Aurora MySQL, see [Maximum Connections to an Aurora MySQL DB Instance \(p. 624\)](#).

Currently, Aurora MySQL parallel query doesn't support IAM database authentication.

We recommend the following when using the MySQL engine:

- Use IAM database authentication as a mechanism for temporary, personal access to databases.
- Use IAM database authentication only for workloads that can be easily retried.
- Don't use IAM database authentication if your application requires more than 200 new connections per second.

PostgreSQL Limitations for IAM Database Authentication

When using IAM database authentication with PostgreSQL, note the following limitation:

- The maximum number of connections per second for your database cluster may be limited depending on the cluster type and your workload.

Enabling and Disabling IAM Database Authentication

By default, IAM database authentication is disabled on DB clusters. You can enable IAM database authentication (or disable it again) using the AWS Management Console, AWS CLI, or the API.

Console

To create a new DB cluster with IAM authentication by using the console, see [Creating an Amazon Aurora DB Cluster \(p. 100\)](#).

Each creation workflow has a **Configure Advanced Settings** page, where you can enable IAM DB authentication. In that page's **Database Options** section, choose **Yes** for **Enable IAM DB Authentication**.

To enable or disable IAM 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

Make sure that all affected DB instances are compatible with IAM authentication. Check the compatibility requirements in [Availability for IAM Database Authentication \(p. 207\)](#). For an Aurora DB cluster, you can only enable IAM authentication if all DB instances in the cluster are compatible with IAM.

4. Choose **Modify**.
5. In the **Database options** section, for **IAM DB authentication** choose **Enable IAM DB authentication** or **Disable**, and then choose **Continue**.
6. To apply the changes immediately, choose **Apply immediately**.
7. Choose **Modify cluster**.

To restore a DB cluster

1. Open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. In the navigation pane, choose **Snapshots**.
3. Choose the snapshot that you want to restore, and then choose **Restore Snapshot for Actions**.
4. In the **Settings** section, enter an identifier for the DB instance for **DB Instance Identifier**.
5. In the **Database options** section, for **IAM DB authentication**, choose **Enable IAM DB authentication** or **Disable**.
6. Choose **Restore DB Instance**.

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

Make sure that all affected DB instances are compatible with IAM authentication. Check the compatibility requirements in [Availability for IAM Database Authentication \(p. 207\)](#). For an Aurora DB cluster, you can only enable IAM authentication if all DB instances in the cluster are compatible with IAM.

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

Make sure that all affected DB instances are compatible with IAM authentication. Check the compatibility requirements in [Availability for IAM Database Authentication \(p. 207\)](#). For an Aurora DB cluster, you can only enable IAM authentication if all DB instances in the cluster are compatible with IAM.

If you are restoring a DB cluster, use one of the following API operations:

- [RestoreDBClusterToPointInTime](#)
- [RestoreDBClusterFromSnapshot](#)

The IAM database authentication setting defaults to that of the source snapshot. To change this setting, set the `EnableIAMDatabaseAuthentication` parameter to `true` to enable IAM authentication, or `false` to disable it.

Creating and Using an IAM Policy for IAM Database Access

To allow an IAM user or role to connect to your DB cluster, you must create an IAM policy. After that, you attach the policy to an IAM user or role.

Note

To learn more about IAM policies, see [Identity and Access Management in Amazon Aurora \(p. 191\)](#).

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-ABCDEFGHIJKL01234/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 \(p. 49\)](#) creates an IAM administrator user. If you want to restrict administrator access to DB clusters, you can create an IAM role with the appropriate, lesser privileged permissions and assign it to the administrator.

Note

Don't confuse the `rds-db:` prefix with other RDS API operation prefixes that begin with `rds:`. You use the `rds-db:` prefix and the `rds-db:connect` action only for IAM database authentication. They aren't valid in any other context.

Currently, the IAM console displays an error for policies with the `rds-db:connect` action. You can ignore this error.

The example policy includes a single statement with the following elements:

- **Effect** – Specify `Allow` to grant access to the DB cluster. If you don't explicitly allow access, then access is denied by default.
- **Action** – Specify `rds-db:connect` to allow connections to the DB cluster.
- **Resource** – Specify an Amazon Resource Name (ARN) that describes one database account in one DB cluster. The ARN format is as follows.

```
arn:aws:rds-db:region:account-id:dbuser:DbClusterResourceId/db-user-name
```

In this format, replace the following:

- `region` is the AWS Region for the DB cluster. In the example policy, the AWS Region is `us-east-2`.
- `account-id` is the AWS account number for the DB cluster. In the example policy, the account number is `1234567890`.
- `DbClusterResourceId` is the identifier for the DB cluster. This identifier is unique to an AWS Region and never changes. In the example policy, the identifier is `cluster-ABCDEFGHIJKL01234`.

To find a DB cluster resource ID in the AWS Management Console for Amazon Aurora, choose the DB cluster to see its details. Then choose the **Configuration** tab. The **Resource ID** is shown in the **Configuration** section.

Alternatively, you can use the AWS CLI command to list the identifiers and resource IDs for all of your DB cluster in the current AWS Region, as shown following.

```
aws rds describe-db-clusters --query "DBClusters[*].  
[DBClusterIdentifier,DbClusterResourceId]"
```

- **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-2:1234567890:dbuser:jane_doe"  
            ]  
        }  
    ]  
}
```

```
        "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.

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.

Using IAM Authentication with PostgreSQL

To use IAM authentication with PostgreSQL, connect to the DB cluster, create database users, and then grant them the `rds_iam` role as shown in the following example.

```
CREATE USER db_userx WITH LOGIN;
GRANT rds_iam TO db_userx;
```

Using IAM Authentication with MySQL

With 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 MySQL to use the `AWSAuthenticationPlugin` to authenticate the database account (`jane_doe`). The `AS 'RDS'` clause refers to the authentication method, and the specified database account should have the same name as the IAM user or role. In this example, both the database account and the IAM user or role are named `jane_doe`.

Note

If you see the following message, it means that the AWS-provided plugin is not available for the current DB cluster.

`ERROR 1524 (HY000): Plugin 'AWSAuthenticationPlugin' is not loaded`
To troubleshoot this error, verify that you are using a supported configuration and that you have enabled IAM database authentication on your DB cluster. For more information, see [Availability for IAM Database Authentication \(p. 207\)](#) and [Enabling and Disabling IAM Database Authentication \(p. 208\)](#).

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.

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 the AWS SDK for Java 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 the AWS SDK for Java 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 the AWS SDK for Java.

For more information, see [Use IAM authentication to connect with SQL Workbench/J to Amazon Aurora MySQL or Amazon RDS for MySQL](#).

Topics

- [Connecting to Your DB Cluster Using IAM Authentication from the Command Line: AWS CLI and mysql Client \(p. 213\)](#)
- [Connecting to Your DB Cluster Using IAM Authentication from the Command Line: AWS CLI and psql Client \(p. 215\)](#)
- [Connecting to Your DB Cluster Using IAM Authentication and the AWS SDK for Java \(p. 216\)](#)

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.

Topics

- [Generating an IAM Authentication Token \(p. 214\)](#)
- [Connecting to a DB Cluster \(p. 214\)](#)

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.cdgmuqiadpid.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.cdgmuqiadpid.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]rds-combined-ca-bundle.pem --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 SSL certificate file that contains the public key. For more information, see [Using SSL/TLS to Encrypt a Connection to a DB Cluster \(p. 177\)](#).
- **--enable-cleartext-plugin** – A value that specifies that `AWSAuthenticationPlugin` must be used for this connection.
- **--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.

```
RDSHOST="rdsmysql.cdgmuqiadpid.us-west-2.rds.amazonaws.com"
TOKEN=$(aws rds generate-db-auth-token --hostname $RDSHOST --port 3306 --region us-west-2
--username jane_doe )"
```

```
mysql --host=$RDSHOST --port=3306 --ssl-ca=/sample_dir/rds-combined-ca-bundle.pem --enable-cleartext-plugin --user=jane_doe --password=$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.

Topics

- [Generating an IAM Authentication Token \(p. 215\)](#)
- [Connecting to an Aurora PostgreSQL Cluster \(p. 216\)](#)

Generating an IAM Authentication Token

The authentication token consists of several hundred characters so it can be unwieldy on the command line. One way to work around this is to save the token to an environment variable, and then use that variable when you connect. The following example shows how to use the AWS CLI to get a signed authentication token using the `generate-db-auth-token` command, and store it in a `PGPASSWORD` environment variable.

```
export RDSHOST="mynpgsql-cluster.cluster-cdgmuiadpid.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-cdgmuiqiadpid.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=certificateFile  
dbname=DBName user=userName"
```

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 SSL certificate file that contains the public key. For more information, see [Using SSL with a PostgreSQL DB Instance](#).
- **dbname** – The database that you want to access.
- **user** – The database account that you want to access.

The following example shows using the command to connect. The example uses the environment variables that were set when the token was generated in the previous section.

```
psql "host=$RDSHOST port=5432 sslmode=verify-full sslrootcert=/sample_dir/rds-combined-ca-  
bundle.pem dbname=DBName user=jane_doe"
```

Connecting to Your DB Cluster Using IAM Authentication and the AWS SDK for Java

You can connect from the command line to an Aurora MySQL or Aurora PostgreSQL DB cluster with the AWS SDK for Java as described following.

Topics

- [Generating an IAM Authentication Token \(p. 216\)](#)
- [Manually Constructing an IAM Authentication Token \(p. 217\)](#)
- [Connecting to a DB Cluster \(p. 220\)](#)

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.cdgmuiadpid.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.cdgmuqiadpid.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'HHmmssZ").format(now);
        DefaultAWSCredentialsProviderChain creds = new
DefaultAWSCredentialsProviderChain();
        String awsAccessKey = creds.getCredentials().getAWSAccessKeyId();
        String awsSecretKey = creds.getCredentials().getAWSSecretKey();
        String expiryMinutes = "900";

        System.out.println("Step 1: Create a canonical request:");
        String canonicalString = createCanonicalString(username, awsAccessKey, date,
dateTimeStamp, region, expiryMinutes, instanceName, port);
        System.out.println(canonicalString);
        System.out.println();

        System.out.println("Step 2: Create a string to sign:");
        String stringToSign = createStringToSign(dateTimeStamp, canonicalString,
awsAccessKey, date, region);
        System.out.println(stringToSign);
        System.out.println();

        System.out.println("Step 3: Calculate the signature:");
        String signature = BinaryUtils.toHex(calculateSignature(stringToSign,
new SigningKey(awsSecretKey, date, region, serviceName)));
        System.out.println(signature);
        System.out.println();

        System.out.println("Step 4: Add the signing info to the request");
        System.out.println(appendSignature(signature));
        System.out.println();

    }

    //Step 1: Create a canonical request date should be in format YYYYMMDD and dateTime
    //should be in format YYYYMMDDTHHMMSSZ
    public static String createCanonicalString(String user, String accessKey, String date,
String dateTime, String region, String expiryPeriod, String hostName, String port) throws
Exception {
        canonicalQueryParameters.put("Action", action);
        canonicalQueryParameters.put("DBUser", user);
        canonicalQueryParameters.put("X-Amz-Algorithm", "AWS4-HMAC-SHA256");
        canonicalQueryParameters.put("X-Amz-Credential", accessKey + "%2F" + date + "%2F" +
region + "%2F" + serviceName + "%2Faws4_request");
        canonicalQueryParameters.put("X-Amz-Date", dateTime);
        canonicalQueryParameters.put("X-Amz-Expires", expiryPeriod);
        canonicalQueryParameters.put("X-Amz-SignedHeaders", signedHeader);
    }
}

```

```

        String canonicalQueryString = "";
        while(!canonicalQueryParameters.isEmpty()) {
            String currentQueryParameter = canonicalQueryParameters.firstKey();
            String currentQueryParameterValue =
canonicalQueryParameters.remove(currentQueryParameter);
            canonicalQueryString = canonicalQueryString + currentQueryParameter + "=" +
currentQueryParameterValue;
            if (!currentQueryParameter.equals("X-Amz-SignedHeaders")) {
                canonicalQueryString += "&";
            }
        }
        String canonicalHeaders = "host:" + hostName + ":" + port + '\n';
        requestWithoutSignature = hostName + ":" + port + "/" + canonicalQueryString;

        String hashedPayload = BinaryUtils.toHex(hash(payload));
        return httpMethod + '\n' + canonicalURIParameter + '\n' + canonicalQueryString +
'\n' + canonicalHeaders + '\n' + signedHeader + '\n' + hashedPayload;

    }

    //Step 2: Create a string to sign using sig v4
    public static String createStringToSign(String date, String canonicalRequest,
String accessKey, String date, String region) throws Exception {
    String credentialScope = date + "/" + region + "/" + serviceName + "/aws4_request";
    return algorithm + '\n' + date + '\n' + canonicalRequest + '\n' +
BinaryUtils.toHex(hash(canonicalRequest));

}

//Step 3: Calculate signature
/**
 * Step 3 of the AWS Signature version 4 calculation. It involves deriving
 * the signing key and computing the signature. Refer to
 * http://docs.aws.amazon
 * .com/general/latest/gr/sigv4-calculate-signature.html
 */
public static byte[] calculateSignature(String stringToSign,
                                         byte[] signingKey) {
    return sign(stringToSign.getBytes(Charset.forName("UTF-8")), signingKey,
               SigningAlgorithm.HmacSHA256);
}

public static byte[] sign(byte[] data, byte[] key,
                        SigningAlgorithm algorithm) throwsSdkClientException {
    try {
        Mac mac = algorithm.getMac();
        mac.init(new SecretKeySpec(key, algorithm.toString()));
        return mac.doFinal(data);
    } catch (Exception e) {
        throw new SdkClientException(
            "Unable to calculate a request signature: "
            + e.getMessage(), e);
    }
}

public static byte[] newSigningKey(String secretKey,
                                  String dateStamp, String regionName, String serviceName)
{
    byte[] kSecret = ("AWS4" + secretKey).getBytes(Charset.forName("UTF-8"));
    byte[] kDate = sign(dateStamp, kSecret, SigningAlgorithm.HmacSHA256);
    byte[] kRegion = sign(regionName, kDate, SigningAlgorithm.HmacSHA256);
    byte[] kService = sign(serviceName, kRegion,
                           SigningAlgorithm.HmacSHA256);
    return sign(AWS4_TERMINATOR, kService, SigningAlgorithm.HmacSHA256);
}

```

```

public static byte[] sign(String stringData, byte[] key,
                         SigningAlgorithm algorithm) throws SdkClientException {
    try {
        byte[] data = stringData.getBytes(UTF8);
        return sign(data, key, algorithm);
    } catch (Exception e) {
        throw new SdkClientException(
            "Unable to calculate a request signature: "
            + e.getMessage(), e);
    }
}

//Step 4: append the signature
public static String appendSignature(String signature) {
    return requestWithoutSignature + "&X-Amz-Signature=" + signature;
}

public static byte[] hash(String s) throws Exception {
    try {
        MessageDigest md = MessageDigest.getInstance("SHA-256");
        md.update(s.getBytes(UTF8));
        return md.digest();
    } catch (Exception e) {
        throw new SdkClientException(
            "Unable to compute hash while signing request: "
            + e.getMessage(), e);
    }
}
}

```

Connecting to a DB Cluster

The following code example shows how to generate an authentication token, and then use it to connect to a cluster running 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 \(p. 177\)](#).) 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 [Intermediate Certificates \(p. 178\)](#). 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.cdgmuqiadpid.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-2015-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:
     * btusii123.cmz7kenwo2ye.rds.cn-north-1.amazonaws.com.cn:3306/?Action=connect&DBUser=iamtestuser&X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-
     * Date=20171003T010726Z&X-Amz-SignedHeaders=host&X-Amz-Expires=899&X-Amz-
     * Credential=AKIAFPXHGVDI5RNFO4AQ%2F20171003%2Fcn-north-1%2Frds-db%2Faws4_request&X-Amz-
     * Signature=f9f45ef96c1f770cdad11a53e33ffa4c3730bc03fdee820cfdf1322eed15483b
     * @return
     */
    private static String generateAuthToken() {
        BasicAWSCredentials awsCredentials = new BasicAWSCredentials(AWS_ACCESS_KEY,
        AWS_SECRET_KEY);

        RdsIamAuthTokenGenerator generator = RdsIamAuthTokenGenerator.builder()
            .credentials(new
        AWSStaticCredentialsProvider(awsCredentials)).region(REGION_NAME).build();
        return generator.getAuthToken(GetIamAuthTokenRequest.builder()

        .hostname(RDS_INSTANCE_HOSTNAME).port(RDS_INSTANCE_PORT).userName(DB_USER).build());
    }

    /**
     * This method sets the SSL properties which specify the key store file, its type and
     * password:
     * @throws Exception
     */
    private static void setSslProperties() throws Exception {
        System.setProperty("javax.net.ssl.trustStore", createKeyStoreFile());
        System.setProperty("javax.net.ssl.trustStoreType", KEY_STORE_TYPE);
        System.setProperty("javax.net.ssl.trustStorePassword", DEFAULT_KEY_STORE_PASSWORD);
    }

    /**
     * This method returns the path of the Key Store File needed for the SSL verification
     * during the IAM Database Authentication to
     * the db instance.
     * @return
     */
}

```

```
* @throws Exception
*/
private static String createKeyStoreFile() throws Exception {
    return createKeyStoreFile(createCertificate()).getPath();
}

/**
 * This method generates the SSL certificate
 * @return
 * @throws Exception
 */
private static X509Certificate createCertificate() throws Exception {
    CertificateFactory certFactory = CertificateFactory.getInstance("X.509");
    URL url = new File(SSL_CERTIFICATE).toURI().toURL();
    if (url == null) {
        throw new Exception();
    }
    try (InputStream certInputStream = url.openStream()) {
        return (X509Certificate) certFactory.generateCertificate(certInputStream);
    }
}

/**
 * This method creates the Key Store File
 * @param rootX509Certificate - the SSL certificate to be stored in the KeyStore
 * @return
 * @throws Exception
 */
private static File createKeyStoreFile(X509Certificate rootX509Certificate) throws
Exception {
    File keyStoreFile = File.createTempFile(KEY_STORE_FILE_PREFIX,
KEY_STORE_FILE_SUFFIX);
    try (FileOutputStream fos = new FileOutputStream(keyStoreFile.getPath())) {
        KeyStore ks = KeyStore.getInstance(KEY_STORE_TYPE, KEY_STORE_PROVIDER);
        ks.load(null);
        ks.setCertificateEntry("rootCaCertificate", rootX509Certificate);
        ks.store(fos, DEFAULT_KEY_STORE_PASSWORD.toCharArray());
    }
    return keyStoreFile;
}

/**
 * This method clears the SSL properties.
 * @throws Exception
 */
private static void clearSslProperties() throws Exception {
    System.clearProperty("javax.net.ssl.trustStore");
    System.clearProperty("javax.net.ssl.trustStoreType");
    System.clearProperty("javax.net.ssl.trustStorePassword");
}
```

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 Am Not Authorized to Perform an Action in Aurora \(p. 224\)](#)
- [I Am Not Authorized to Perform iam:PassRole \(p. 224\)](#)
- [I Want to View My Access Keys \(p. 224\)](#)

- [I'm an Administrator and Want to Allow Others to Access Aurora \(p. 225\)](#)
- [I Want to Allow People Outside of My AWS Account to Access My Aurora Resources \(p. 225\)](#)

I Am 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 Am 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 \(p. 195\)](#).
- 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. For more information, see [Monitoring with Amazon CloudWatch \(p. 436\)](#).

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 [Logging Amazon RDS API Calls with AWS CloudTrail \(p. 574\)](#).

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 [Enhanced Monitoring \(p. 469\)](#).

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 [Using Amazon RDS Performance Insights \(p. 476\)](#).

Database Logs

You can view, download, and watch database logs using the AWS Management Console, AWS CLI, or RDS API. For more information, see [Amazon Aurora Database Log Files \(p. 563\)](#).

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 [Using Amazon Aurora Recommendations \(p. 520\)](#).

Amazon Aurora Event Notification

Amazon Aurora uses the Amazon Simple Notification Service (Amazon SNS) to provide notification when an Amazon Aurora event occurs. These notifications can be in any notification form supported by Amazon SNS for an AWS Region, such as an email, a text message, or a call to an HTTP endpoint. For more information, see [Using Amazon RDS Event Notification \(p. 543\)](#).

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

For Aurora PostgreSQL, 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 [Using Database Activity Streams with Aurora PostgreSQL \(p. 525\)](#).

For more information about monitoring Aurora see [Monitoring an Amazon Aurora DB Cluster \(p. 433\)](#).

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 \(p. 379\)](#).

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 \(p. 47\)](#).

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 Aurora \(p. 27\)](#).

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.

You can call these API operations from any network location. However, Amazon Aurora also supports resource-based access policies, which can include restrictions based on the source IP address. In addition, you can use Amazon Aurora policies to control access from specific Amazon VPC endpoints or specific VPCs. Effectively, this isolates network access to a given Amazon Aurora resource from only the specific VPC within the AWS network.

In addition, Aurora offers features to help support infrastructure security.

Security Groups

Security groups control the access that traffic has in and out of a DB instance. By default, network access is turned off to a DB instance. You can specify rules in a security group that allow access from an IP address range, port, or Amazon EC2 security group. After ingress rules are configured, the same rules apply to all DB instances that are associated with that security group.

For more information, see [Controlling Access with Security Groups \(p. 231\)](#).

Public Accessibility

When you launch a DB instance inside a virtual private cloud (VPC) based on the Amazon VPC service, you can turn on or off public accessibility for that instance. To designate whether the DB instance that you create has a DNS name that resolves to a public IP address, you use the *Public accessibility* parameter. By using this parameter, you can designate whether there is public access to the DB instance. You can modify a DB instance to turn on or off public accessibility by modifying the *Public accessibility* parameter.

For more information, see [Hiding a DB Instance in a VPC from the Internet \(p. 251\)](#).

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.

- Assign an individual IAM account to each person who manages Amazon Aurora resources. Don't use AWS root credentials to manage Amazon Aurora resources; you should create an IAM user for everyone, including yourself.
- 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*.

For more information about IAM, see [AWS Identity and Access Management](#). For information on IAM best practices, see [IAM Best Practices](#).

Use the AWS Management Console, the AWS CLI, or the RDS API to change the password for your master user. If you use another tool, such as a SQL client, to change the master user password, it might result in privileges being revoked for the user unintentionally.

Controlling Access with Security Groups

Security groups control the access that traffic has in and out of a DB instance. Two types of security groups are used with Aurora: VPC security groups and Amazon EC2 security groups. In simple terms, these work as follows:

- A VPC security group controls access to DB instances and EC2 instances inside a VPC.
- An EC2 security group controls access to an EC2 instance.

By default, network access is disabled for a DB instance. You can specify rules in a security group that allow access from an IP address range, port, or EC2 security group. Once ingress rules are configured, the same rules apply to all DB instances that are associated with that security group. You can specify up to 20 rules in a security group.

VPC Security Groups

Each VPC security group rule enables a specific source to access a DB instance in a VPC that is associated with that VPC security group. The source can be a range of addresses (for example, 203.0.113.0/24), or another VPC security group. By specifying a VPC security group as the source, you allow incoming traffic from all instances (typically application servers) that use the source VPC security group. VPC security groups can have rules that govern both inbound and outbound traffic, though the outbound traffic rules typically do not apply to DB instances. Outbound traffic rules only apply if the DB instance acts as a client. You must use the [Amazon EC2 API](#) or the **Security Group** option on the VPC Console to create VPC security groups.

When you create rules for your VPC security group that allow access to the instances in your VPC, you must specify a port for each range of addresses that the rule allows access for. For example, if you want to enable SSH access to instances in the VPC, then you create a rule allowing access to TCP port 22 for the specified range of addresses.

You can configure multiple VPC security groups that allow access to different ports for different instances in your VPC. For example, you can create a VPC security group that allows access to TCP port 80 for web servers in your VPC. You can then create another VPC security group that allows access to TCP port 3306 for Aurora MySQL DB instances in your VPC.

Note

In an Aurora DB cluster, the VPC security group associated with the DB cluster is also associated with all of the DB instances in the DB cluster. If you change the VPC security group for the DB cluster or for a DB instance, the change is applied automatically to all of the DB instances in the DB cluster.

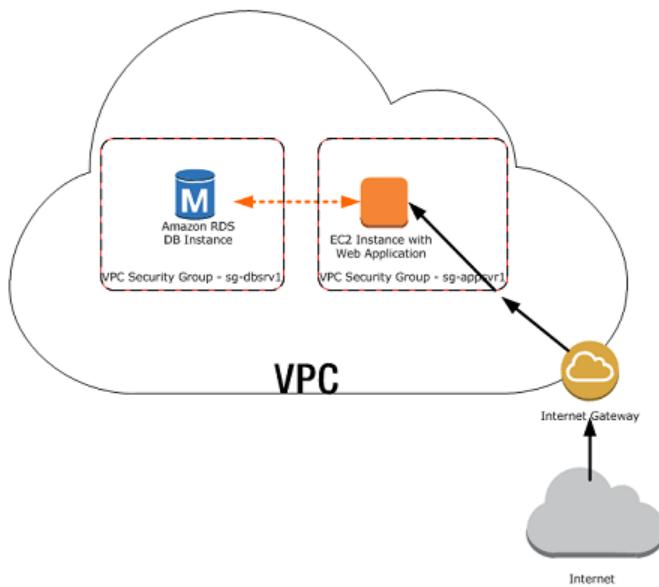
For more information on VPC security groups, see [Security Groups](#) in the *Amazon Virtual Private Cloud User Guide*.

Security Group Scenario

A common use of a DB instance in a VPC is to share data with an application server running in an Amazon EC2 instance in the same VPC, which is accessed by a client application outside the VPC. For this scenario, you use the RDS and VPC pages on the AWS Management Console or the RDS and EC2 API operations to create the necessary instances and security groups:

1. Create a VPC security group (for example, sg-appsrv1) 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-appsrv1) that you created in the previous step. The EC2 instance in the VPC shares the VPC security group with the DB instance.
3. Create a second VPC security group (for example, sg-dbsrv1) and create a new rule by specifying the VPC security group that you created in step 1 (sg-appsrv1) as the source.
4. Create a new DB instance and add the DB instance to the VPC security group (sg-dbsrv1) that you created in the previous step. When you create the DB instance, use the same port number as the one specified for the VPC security group (sg-dbsrv1) rule that you created in step 3.

The following diagram shows this scenario.



For more information about using a VPC, see [Amazon Virtual Private Cloud VPCs and Amazon Aurora \(p. 239\)](#).

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 \(p. 52\)](#) and [Security Groups](#) in the [Amazon Virtual Private Cloud User Guide](#).

Associating a Security Group with a DB Instance

You can associate a security group with a DB instance by using **Modify** on the RDS console, the `ModifyDBInstance` Amazon RDS API, or the `modify-db-instance` AWS CLI command.

For information about modifying a DB instance in a DB cluster, see [Modify a DB Instance in a DB Cluster \(p. 265\)](#). For security group considerations when you restore a DB instance from a DB snapshot, see [Security Group Considerations \(p. 385\)](#).

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 \(p. 264\)](#).

Master User Account Privileges

When you create a new DB cluster, the default master user that you use gets certain privileges for that DB cluster. The following table shows the privileges and database roles the master user gets for each of the database engines.

Important

We strongly recommend that you do not use the master user directly in your applications. Instead, adhere to the best practice of using a database user created with the minimal privileges required for your application.

Note

If you accidentally delete the permissions for the master user, you can restore them by modifying the DB cluster and setting a new master user password. For more information about modifying a DB cluster, see [Modifying an Amazon Aurora DB Cluster \(p. 264\)](#).

Database Engine	System Privilege	Database Role
Amazon Aurora MySQL	CREATE, DROP, GRANT OPTION, REFERENCES, EVENT, ALTER, DELETE, INDEX, INSERT, SELECT, UPDATE, CREATE TEMPORARY TABLES, LOCK TABLES, TRIGGER, CREATE VIEW, SHOW VIEW, LOAD FROM S3, SELECT INTO S3, ALTER ROUTINE, CREATE ROUTINE, EXECUTE, CREATE USER, PROCESS, SHOW DATABASES , RELOAD, REPLICATION CLIENT, REPLICATION SLAVE	—
Amazon Aurora PostgreSQL	LOGIN, NOSUPERUSER, INHERIT, CREATEDB, CREATEROLE, NOREPLICATION, VALID UNTIL 'infinity'	RDS_SUPERUSER

Using Service-Linked Roles for Amazon Aurora

Amazon Aurora uses AWS Identity and Access Management (IAM) [service-linked roles](#). A service-linked role is a unique type of IAM role that is linked directly to Amazon Aurora. Service-linked roles are predefined by Amazon Aurora and include all the permissions that the service requires to call other AWS services on your behalf.

A service-linked role makes using Amazon Aurora easier because you don't have to manually add the necessary permissions. Amazon Aurora defines the permissions of its service-linked roles, and unless defined otherwise, only Amazon Aurora can assume its roles. The defined permissions include the trust policy and the permissions policy, and that permissions policy cannot be attached to any other IAM entity.

You can delete the roles only after first deleting their related resources. This protects your Amazon Aurora resources because you can't inadvertently remove permission to access the resources.

For information about other services that support service-linked roles, see [AWS Services That Work with IAM](#) and look for the services that have **Yes** in the **Service-Linked Role** column. Choose a **Yes** with a link to view the service-linked role documentation for that service.

Service-Linked Role Permissions for Amazon Aurora

Amazon Aurora uses the service-linked role named **AWSServiceRoleForRDS** – to allow Amazon RDS to call AWS services on behalf of your DB clusters.

The **AWSServiceRoleForRDS** service-linked role trusts the following services to assume the role:

- `rds.amazonaws.com`

The role permissions policy allows Amazon Aurora to complete the following actions on the specified resources:

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": [  
                "ec2:AuthorizeSecurityGroupIngress",  
                "ec2>CreateNetworkInterface",  
                "ec2>CreateSecurityGroup",  
                "ec2>DeleteNetworkInterface",  
                "ec2>DeleteSecurityGroup",  
                "ec2:DescribeAvailabilityZones",  
                "ec2:DescribeInternetGateways",  
                "ec2:DescribeSecurityGroups",  
                "ec2:DescribeSubnets",  
                "ec2:DescribeVpcAttribute",  
                "ec2:DescribeVpcs",  
                "ec2:ModifyNetworkInterfaceAttribute",  
                "ec2:ModifyVpcEndpoint",  
                "ec2:RevokeSecurityGroupIngress",  
                "ec2>CreateVpcEndpoint",  
                "ec2:DescribeVpcEndpoints",  
                "ec2>DeleteVpcEndpoints",  
                "ec2:AssignPrivateIpAddresses",  
                "ec2:UnassignPrivateIpAddresses"  
            ],  
            "Resource": "*"  
        }  
    ]  
}
```

```

},
{
    "Effect": "Allow",
    "Action": [
        "sns:Publish"
    ],
    "Resource": "*"
},
{
    "Effect": "Allow",
    "Action": [
        "logs>CreateLogGroup"
    ],
    "Resource": [
        "arn:aws:logs:***:log-group:/aws/rds/*",
        "arn:aws:logs:***:log-group:/aws/docdb/*",
        "arn:aws:logs:***:log-group:/aws/neptune/*"
    ]
},
{
    "Effect": "Allow",
    "Action": [
        "logs>CreateLogStream",
        "logs:PutLogEvents",
        "logs:DescribeLogStreams"
    ],
    "Resource": [
        "arn:aws:logs:***:log-group:/aws/rds/*:log-stream:*",
        "arn:aws:logs:***:log-group:/aws/docdb/*:log-stream:*",
        "arn:aws:logs:***:log-group:/aws/neptune/*:log-stream:*
    ]
},
{
    "Effect": "Allow",
    "Action": [
        "kinesis>CreateStream",
        "kinesis:PutRecord",
        "kinesis:PutRecords",
        "kinesis:DescribeStream",
        "kinesis:SplitShard",
        "kinesis:MergeShards",
        "kinesis>DeleteStream",
        "kinesis:UpdateShardCount"
    ],
    "Resource": [
        "arn:aws:kinesis:***:stream/aws-rds-das-*"
    ]
}
]
}

```

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 IAM Account](#).

If you delete this service-linked role, and then need to create it again, you can use the same process to recreate the role in your account. When you create a DB cluster, Amazon Aurora creates the service-linked role for you again.

Editing a Service-Linked Role for Amazon Aurora

Amazon Aurora does not allow you to edit the AWSServiceRoleForRDS service-linked role. After you create a service-linked role, you cannot change the name of the role because various entities might reference the role. However, you can edit the description of the role using IAM. For more information, see [Editing a Service-Linked Role](#) in the *IAM User Guide*.

Deleting a Service-Linked Role for Amazon Aurora

If you no longer need to use a feature or service that requires a service-linked role, we recommend that you delete that role. That way you don't have an unused entity that is not actively monitored or maintained. However, you must delete all of your DB clusters before you can delete the service-linked role.

Cleaning Up a Service-Linked Role

Before you can use IAM to delete a service-linked role, you must first confirm that the role has no active sessions and remove any resources used by the role.

To check whether the service-linked role has an active session in the IAM console

1. Sign in to the AWS Management Console and open the IAM console at <https://console.aws.amazon.com/iam/>.
2. In the navigation pane of the IAM console, choose **Roles**. Then choose the name (not the check box) of the AWSServiceRoleForRDS role.
3. On the **Summary** page for the chosen role, choose the **Access Advisor** tab.
4. On the **Access Advisor** tab, review recent activity for the service-linked role.

Note

If you are unsure whether Amazon Aurora is using the AWSServiceRoleForRDS role, you can try to delete the role. If the service is using the role, then the deletion fails and you can view the AWS Regions where the role is being used. If the role is being used, then you must wait

for the session to end before you can delete the role. You cannot revoke the session for a service-linked role.

If you want to remove the AWSServiceRoleForRDS role, you must first delete *all* of your DB clusters.

Deleting All of Your Clusters

Use one of the following procedures to delete a single cluster. Repeat the procedure for each of your clusters.

To delete a cluster (console)

1. Open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. In the **Databases** list, choose the cluster that you want to delete.
3. For **Cluster Actions**, choose **Delete**.
4. Choose **Delete**.

To delete a cluster (CLI)

See [delete-db-cluster](#) in the *AWS CLI Command Reference*.

To delete a cluster (API)

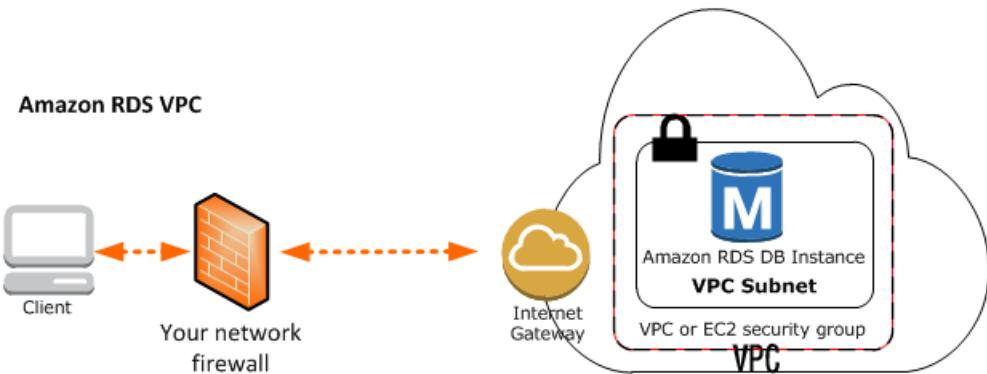
See [DeleteDBCluster](#) in the *Amazon RDS API Reference*.

You can use the IAM console, the IAM CLI, or the IAM API to delete the AWSServiceRoleForRDS service-linked role. For more information, see [Deleting a Service-Linked Role](#) in the *IAM User Guide*.

Amazon Virtual Private Cloud VPCs and Amazon Aurora

Amazon Virtual Private Cloud (Amazon VPC) enables you to launch AWS resources, such as Aurora DB clusters, into a virtual private cloud (VPC).

When you use an Amazon VPC, you have control over your virtual networking environment: you can choose your own IP address range, create subnets, and configure routing and access control lists. There is no additional cost to run your DB instance in an Amazon VPC.



Accounts that support only the *EC2-VPC* platform have a default VPC. All new DB instances are created in the default VPC unless you specify otherwise. If you are a new Amazon Aurora customer, if you have never created a DB instance before, or if you are creating a DB instance in an AWS Region you have not used before, you are most likely on the *EC2-VPC* platform and have a default VPC.

Topics

- [How to Create a VPC for Use with Amazon Aurora \(p. 239\)](#)
- [Scenarios for Accessing a DB Instance in a VPC \(p. 245\)](#)
- [Working with a DB Instance in a VPC \(p. 249\)](#)
- [Tutorial: Create an Amazon VPC for Use with a DB Instance \(p. 255\)](#)

This documentation only discusses VPC functionality relevant to Amazon Aurora DB clusters. For more information about Amazon VPC, see [Amazon VPC Getting Started Guide](#) and [Amazon VPC User Guide](#). For information about using a network address translation (NAT) gateway, see [NAT Gateways](#) in the [Amazon Virtual Private Cloud User Guide](#).

How to Create a VPC for Use with Amazon Aurora

The following sections discuss how to create a VPC for use with Amazon Aurora.

Note

For a helpful and detailed guide on connecting to an Amazon Aurora DB cluster, you can see [Aurora MySQL Database Administrator's Handbook – Connection Management](#).

Create a VPC and Subnets

You can only create an Amazon Aurora DB cluster in a Virtual Private Cloud (VPC) that spans two Availability Zones, and each zone must contain at least one subnet. You can create an Aurora DB cluster

in the default VPC for your AWS account, or you can create a user-defined VPC. For information, see [Amazon Virtual Private Cloud VPCs and Amazon Aurora \(p. 239\)](#).

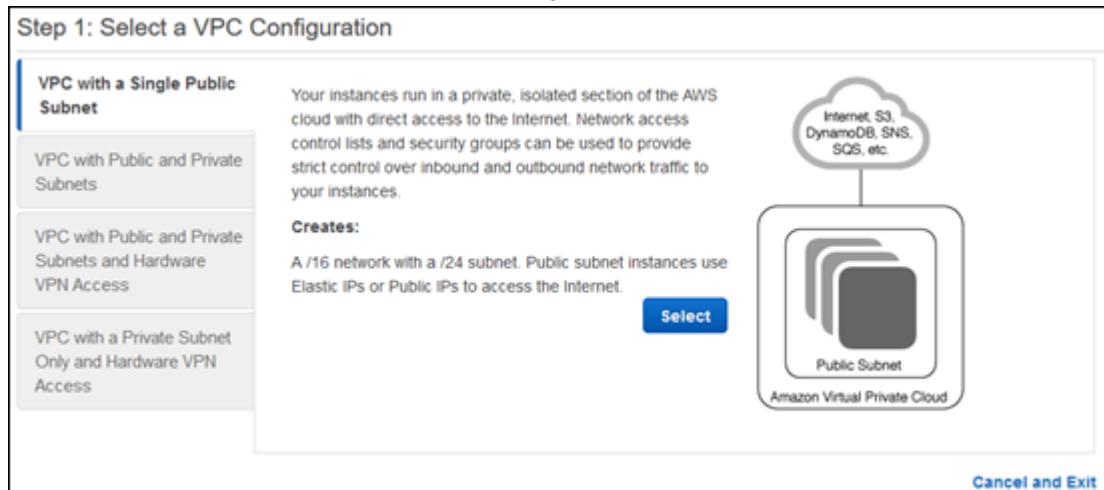
Amazon Aurora optionally can create a VPC and subnet group for you to use with your DB cluster. Doing this can be helpful if you have never created a VPC, or if you would like to create a new VPC that is separate from your other VPCs. If you want Amazon Aurora to create a VPC and subnet group for you, then skip this procedure and see [Create an Aurora MySQL DB Cluster \(p. 54\)](#).

Note

All VPC and EC2 resources that you use with your Aurora DB cluster must be in one of the following regions: US East (N. Virginia), US East (Ohio), US West (Oregon), Asia Pacific (Mumbai), Asia Pacific (Seoul), Asia Pacific (Sydney), Asia Pacific (Tokyo), Canada (Central), Europe (Ireland), Europe (London).

To create a VPC for use with an Aurora DB cluster

1. Sign in to the AWS Management Console and open the Amazon VPC console at <https://console.aws.amazon.com/vpc/>.
2. In the top-right corner of the AWS Management Console, choose the AWS Region to create your VPC in. This example uses the US East (Ohio) Region.
3. In the upper-left corner, choose **VPC Dashboard**. Choose **Start VPC Wizard** to begin creating a VPC.
4. In the Create VPC wizard, choose **VPC with a Single Public Subnet**. Choose **Select**.



5. Set the following values in the **Create VPC** panel:

- **IP CIDR block:** 10.0.0.0/16
- **VPC name:** gs-cluster-vpc
- **Public subnet:** 10.0.0.0/24
- **Availability Zone:** us-east-1a
- **Subnet name:** gs-subnet1
- **Enable DNS hostnames:** Yes
- **Hardware tenancy:** Default

Step 2: VPC with a Single Public Subnet

IPv4 CIDR block*: (65531 IP addresses available)

IPv6 CIDR block: No IPv6 CIDR Block
 Amazon provided IPv6 CIDR block

VPC name:

Public subnet's IPv4 CIDR*: (251 IP addresses available)

Availability Zone*:

Subnet name:

You can add more subnets after AWS creates the VPC.

Service endpoints

Enable DNS hostnames*: Yes No

Hardware tenancy*:

6. Choose **Create VPC**.

7. When your VPC has been created, choose **Close** on the notification page.

To create additional subnets

1. To add the second subnet to your VPC, in the VPC Dashboard choose **Subnets**, and then choose **Create Subnet**. An Amazon Aurora DB cluster requires at least two VPC subnets.
2. Set the following values in the **Create Subnet** panel:
 - **Name tag:** gs-subnet2
 - **VPC:** Choose the VPC that you created in the previous step, for example: vpc-a464d1c1 (10.0.0.0/16) | gs-cluster-vpc.
 - **Availability Zone:** us-east-1c
 - **CIDR block:** 10.0.1.0/24

Name tag	gs-subnet2	Status	Status Reason
VPC	vpc-b5754bcd gs-cluster-vpc		
VPC CIDRs	CIDR	Status	Status Reason
	10.0.0.0/16	associated	

Availability Zone: us-east-1c IPv4 CIDR block: 10.0.1.0/24

[Cancel](#) [Yes, Create](#)

3. Choose **Yes Create**.
4. To ensure that the second subnet that you created uses the same route table as the first subnet, in the VPC Dashboard, choose **Subnets**, and then choose the first subnet that was created for the VPC, **gs-subnet1**. Choose the **Route Table** tab, and note the **Current Route Table**, for example: **rtb-c16ce5bc**.
5. In the list of subnets, clear the first subnet and choose the second subnet, **gs-subnet2**. Choose the **Route Table** tab, and then choose **Edit**. In the **Change to** list, choose the route table from the previous step, for example: **rtb-c16ce5bc**. Choose **Save** to save your choice.

Route Table	
Current Route Table:	rtb-c16ce5bb
Change to:	rtb-c16ce5bc
Destination	Target
10.0.0.0/16	local
0.0.0.0/0	igw-3c84af45

[Cancel](#) [Save](#)

Create a Security Group and Add Inbound Rules

After you've created your VPC and subnets, the next step is to create a security group and add inbound rules.

To create a security group

The last step in creating a VPC for use with your Amazon Aurora DB cluster is to create a VPC security group, which identifies which network addresses and protocols are allowed to access DB instances in your VPC.

1. In the VPC Dashboard, choose **Security Groups**, and then choose **Create Security Group**.
2. Set the following values in the **Create Security Group** panel:

- **Name tag:** gs-securitygroup1
- **Group name:** gs-securitygroup1
- **Description:** Getting Started Security Group
- **VPC:** Choose the VPC that you created earlier, for example: vpc-b5754bcd | gs-cluster-vpc.



3. Choose **Yes, Create** to create the security group.

To add inbound rules to the security group

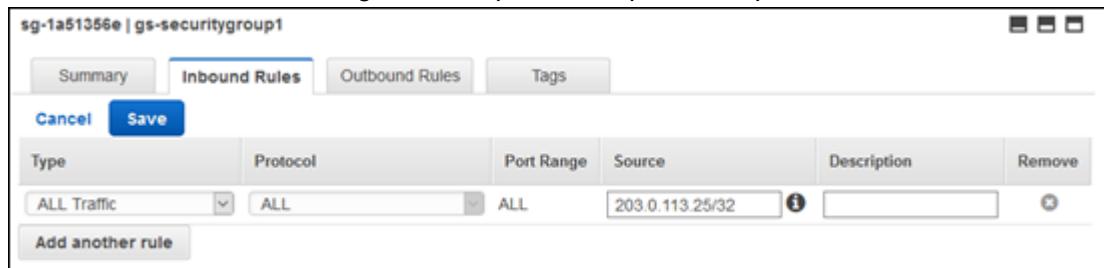
To connect to your Aurora DB cluster, you need to add an inbound rule to your VPC security group that allows inbound traffic to connect.

1. Determine the IP address to use to connect to the Aurora cluster. You can use the service at <https://checkip.amazonaws.com> to determine your public IP address. If you are connecting through an ISP or from behind your firewall without a static IP address, you need to find out the range of IP addresses used by client computers.

Warning

If you use 0.0.0.0/0, you enable all IP addresses to access your DB cluster. This is acceptable for a short time in a test environment, but it's unsafe for production environments. In production, you'll authorize only a specific IP address or range of addresses to access your DB cluster.

2. In the VPC Dashboard, choose **Security Groups**, and then choose the gs-securitygroup1 security group that you created in the previous procedure.
3. Choose the **Inbound** tab, and then choose the **Edit** button.
4. Set the following values for your new inbound rule:
 - **Type:** All Traffic
 - **Source:** The IP address or range from the previous step, for example 203.0.113.25/32.



5. Choose **Save** to save your settings.

Create a DB Subnet Group

The last thing that you need before you can create an Aurora DB cluster is a DB subnet group. Your DB subnet group identifies the subnets that your DB cluster uses from the VPC that you created in the previous steps. Your DB subnet group must include at least one subnet in at least two of the Availability Zones in the AWS Region where you want to deploy your DB cluster.

To create a DB subnet group for use with your Aurora DB cluster

1. Open the Amazon Aurora console at <https://console.aws.amazon.com/rds>.
2. Choose **Subnet Groups**, and then choose **Create DB Subnet Group**.
3. Set the following values for your new DB subnet group:
 - **Name:** gs-subnetgroup1
 - **Description:** Getting Started Subnet Group
 - **VPC ID:** Choose the VPC that you created in the previous procedure, for example, gs-cluster-vpc (vpc-b5754bcd).
4. Choose **Add all the subnets related to this VPC** to add the subnets for the VPC that you created in earlier steps. You can also add each subnet individually by choosing a value for **Availability zone** and **Subnet** and then choosing **Add subnet**.

RDS > Subnet groups > Create DB subnet group

Create DB subnet group

To create a new Subnet Group give it a name, description, and select an existing VPC below. Once you select an existing VPC, you will be able to add subnets related to that VPC.

Subnet group details

Name: gs-subnetgroup1

Description: Getting Started Subnet Group

VPC: gs-cluster-vpc (vpc-b5754bcd)

Add subnets

Add subnet(s) to this subnet group. You may add subnets one at a time below or add all the subnets related to this VPC. You may make additions/edits after this group is created. A minimum of 2 subnets is required.

Add all the subnets related to this VPC

Availability zone: select an availability zone

Subnet: select a subnet Add subnet

Subnets in this subnet group (2)

Availability zone	Subnet ID	CIDR block	Action
us-east-1a	subnet-52f6687d	10.0.0.0/24	Remove
us-east-1c	subnet-5d069500	10.0.1.0/24	Remove

Cancel Create

5. Choose **Create** to create the subnet group.

Scenarios for Accessing a DB Instance in a VPC

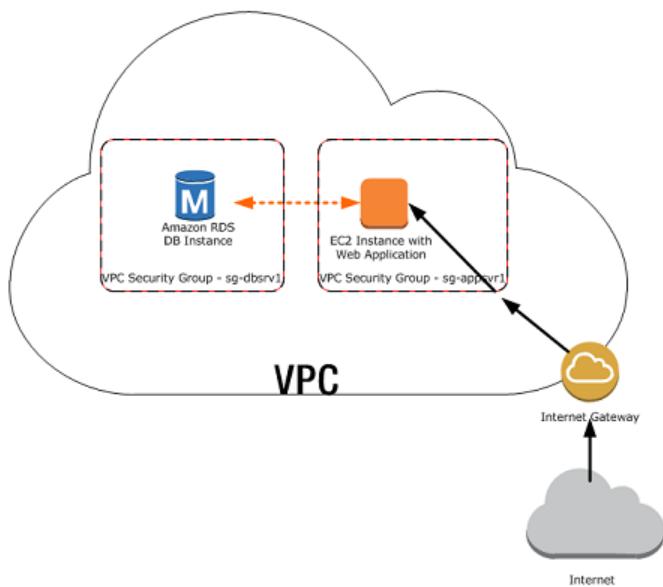
Amazon Aurora supports the following scenarios for accessing a DB instance:

- An EC2 Instance in the Same VPC (p. 246)
- An EC2 Instance in a Different VPC (p. 247)
- An EC2 Instance Not in a VPC (p. 248)
- A Client Application Through the Internet (p. 249)

A DB Instance in a VPC Accessed by an EC2 Instance in the Same VPC

A common use of a DB instance in a VPC is to share data with an application server that is running in an EC2 instance in the same VPC. This is the user scenario created if you use AWS Elastic Beanstalk to create an EC2 instance and a DB instance in the same VPC.

The following diagram shows this scenario.



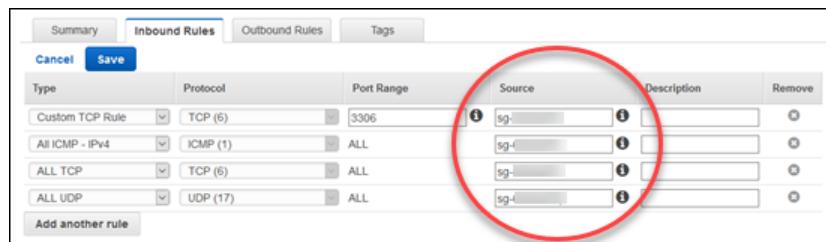
The simplest way to manage access between EC2 instances and DB instances in the same VPC is to do the following:

- Create a VPC security group for your DB instances to be in. This security group can be used to restrict access to the DB instances. For example, you can create a custom rule for this security group that allows TCP access using the port you assigned to the DB instance when you created it and an IP address you use to access the DB instance for development or other purposes.
- Create a VPC security group for your EC2 instances (web servers and clients) to be in. This security group can, if needed, allow access to the EC2 instance from the Internet via the VPC's routing table. For example, you can set rules on this security group to allow TCP access to the EC2 instance over port 22.
- Create custom rules in the security group for your DB instances that allow connections from the security group you created for your EC2 instances. This would allow any member of the security group to access the DB instances.

For a tutorial that shows you how to create a VPC with both public and private subnets for this scenario, see [Tutorial: Create an Amazon VPC for Use with a DB Instance \(p. 255\)](#).

To create a rule in a VPC security group that allows connections from another security group, do the following:

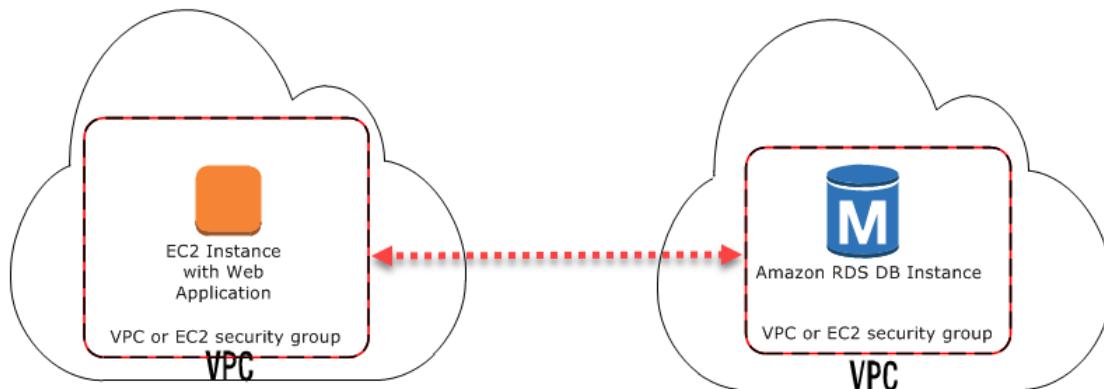
1. Sign in to the AWS Management Console and open the Amazon VPC console at <https://console.aws.amazon.com/vpc>.
2. In the navigation pane, choose **Security Groups**.
3. Choose or create a security group for which you want to allow access to members of another security group. In the scenario preceding, this is the security group that you use for your DB instances. Choose the **Inbound Rules** tab, and then choose **Edit rule**.
4. On the **Edit inbound rules** page, choose **Add Rule**.
5. From **Type**, choose one of the **All ICMP** options. In the **Source** box, start typing the ID of the security group; this provides you with a list of 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.
6. Repeat the steps for the TCP protocol by creating a rule with **All TCP** as the **Type** and your security group in the **Source** box. If you intend to use the UDP protocol, create a rule with **All UDP** as the **Type** and your security group in the **Source** box.
7. Create a custom TCP rule that permits access via the port you used when you created your DB instance, such as port 3306 for MySQL. Enter your security group or an IP address to use in the **Source** box.
8. Choose **Save** when you are done.



A DB Instance in a VPC Accessed by an EC2 Instance in a Different VPC

When your DB instance is in a different VPC from the EC2 instance you are using to access it, you can use VPC peering to access the DB instance.

The following diagram shows this scenario.

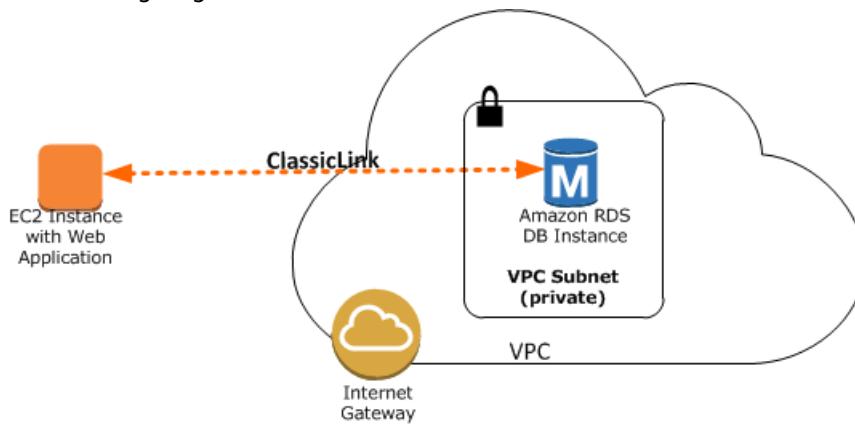


A VPC peering connection is a networking connection between two VPCs that enables you to route traffic between them using private IP addresses. Instances in either VPC can communicate with each other as if they are within the same network. You can create a VPC peering connection between your own VPCs, with a VPC in another AWS account, or with a VPC in a different AWS Region. To learn more about VPC peering, see [VPC Peering](#) in the *Amazon Virtual Private Cloud User Guide*.

A DB Instance in a VPC Accessed by an EC2 Instance Not in a VPC

You can communicate between an Amazon Aurora DB instance that is in a VPC and an EC2 instance that is not in an Amazon VPC by using *ClassicLink*. When you use Classic Link, an application on the EC2 instance can connect to the DB instance by using the endpoint for the DB instance. ClassicLink is available at no charge.

The following diagram shows this scenario.



Using ClassicLink, you can connect an EC2 instance to a logically isolated database where you define the IP address range and control the access control lists (ACLs) to manage network traffic. You don't have to use public IP addresses or tunneling to communicate with the DB instance in the VPC. This arrangement provides you with higher throughput and lower latency connectivity for inter-instance communications.

To enable ClassicLink between a DB instance in a VPC and an EC2 instance not in a VPC

1. Sign in to the AWS Management Console and open the Amazon VPC console at <https://console.aws.amazon.com/vpc>.
2. In the navigation pane, choose **Your VPCs**.
3. Choose the VPC used by the DB instance.
4. In **Actions**, choose **Enable ClassicLink**. In the confirmation dialog box, choose **Yes, Enable**.
5. On the EC2 console, choose the EC2 instance you want to connect to the DB instance in the VPC.
6. In **Actions**, choose **ClassicLink**, and then choose **Link to VPC**.
7. On the **Link to VPC** page, choose the security group you want to use, and then choose **Link to VPC**.

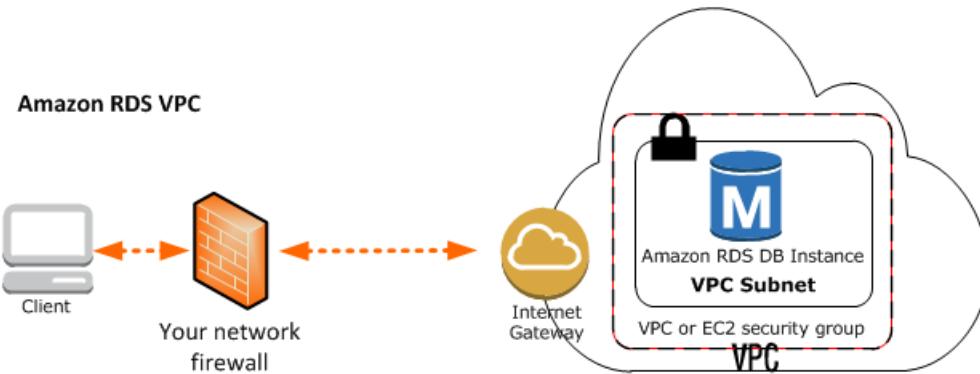
Note

The ClassicLink features are only visible in the consoles for accounts and regions that support EC2-Classic. For more information, see [ClassicLink](#) in the *Amazon EC2 User Guide for Linux Instances*.

A DB Instance in a VPC Accessed by a Client Application Through the Internet

To access a DB instance in a VPC from a client application through the internet, you configure a VPC with a single public subnet, and an Internet gateway to enable communication over the Internet.

The following diagram shows this scenario.



We recommend the following configuration:

- A VPC of size /16 (for example CIDR: 10.0.0.0/16). This size provides 65,536 private IP addresses.
- A subnet of size /24 (for example CIDR: 10.0.0.0/24). This size provides 256 private IP addresses.
- An Amazon Aurora DB instance that is associated with the VPC and the subnet. Amazon RDS assigns an IP address within the subnet to your DB instance.
- An Internet gateway which connects the VPC to the Internet and to other AWS products.
- A security group associated with the DB instance. The security group's inbound rules allow your client application to access to your DB instance.

For information about creating a DB instance in a VPC, see [Creating a DB Instance in a VPC \(p. 251\)](#).

Working with a DB Instance in a VPC

Your DB instance is in a virtual private cloud (VPC). A VPC is a virtual network that is logically isolated from other virtual networks in the AWS Cloud. Amazon VPC lets you launch AWS resources, such as an Amazon Aurora DB instance or Amazon EC2 instance, into a VPC. The VPC can either be a default VPC that comes with your account or one that you create. All VPCs are associated with your AWS account.

Your default VPC has three subnets you can use to isolate resources inside the VPC. The default VPC also has an Internet Gateway that can be used to provide access to resources inside the VPC from outside the VPC.

For a list of scenarios involving Amazon Aurora DB instances in a VPC , see [Scenarios for Accessing a DB Instance in a VPC \(p. 245\)](#).

For a tutorial that shows you how to create a VPC that you can use with a common Amazon Aurora scenario, see [Tutorial: Create an Amazon VPC for Use with a DB Instance \(p. 255\)](#).

To learn how to work with DB instances inside a VPC, see the following:

Topics

- [Working with a DB Instance in a VPC \(p. 250\)](#)
- [Working with DB Subnet Groups \(p. 250\)](#)
- [Hiding a DB Instance in a VPC from the Internet \(p. 251\)](#)
- [Creating a DB Instance in a VPC \(p. 251\)](#)

Working with a DB Instance in a VPC

Here are some tips on working with a DB instance in a VPC:

- Your VPC must have at least two subnets. These subnets must be in two different Availability Zones in the AWS Region where you want to deploy your DB instance. A subnet is a segment of a VPC's IP address range that you can specify and that lets you group instances based on your security and operational needs.
- If you want your DB instance in the VPC to be publicly accessible, you must enable the VPC attributes *DNS hostnames* and *DNS resolution*.
- Your VPC must have a DB subnet group that you create (for more information, see the next section). You create a DB subnet group by specifying the subnets you created. Amazon Aurora uses that DB subnet group and your preferred Availability Zone to choose a subnet and an IP address within that subnet to assign to your DB instance.
- Your VPC must have a VPC security group that allows access to the DB instance.
- The CIDR blocks in each of your subnets must be large enough to accommodate spare IP addresses for Amazon Aurora to use during maintenance activities, including failover and compute scaling.
- A VPC can have an *instance tenancy* attribute of either *default* or *dedicated*. All default VPCs have the instance tenancy attribute set to default, and a default VPC can support any DB instance class.

If you choose to have your DB instance in a dedicated VPC where the instance tenancy attribute is set to dedicated, the DB instance class of your DB instance must be one of the approved Amazon EC2 dedicated instance types. For example, the m3.medium EC2 dedicated instance corresponds to the db.m3.medium DB instance class. For information about instance tenancy in a VPC, go to [Using EC2 Dedicated Instances in the Amazon Virtual Private 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.

Working with DB Subnet Groups

Subnets are segments of a VPC's IP address range that you designate to group your resources based on security and operational needs. A DB subnet group is a collection of subnets (typically private) that you create in a VPC and that you then designate for your DB instances. A DB subnet group allows you to specify a particular VPC when creating DB instances using the CLI or API; if you use the console, you can just choose the VPC and subnets you want to use.

Each DB subnet group should have subnets in at least two Availability Zones in a given AWS Region. When creating a DB instance in a VPC, you must choose a DB subnet group. Amazon Aurora uses that DB subnet group and your preferred Availability Zone to choose a subnet and an IP address within that subnet to associate with your DB instance. If the primary DB instance of a Multi-AZ deployment fails, Amazon Aurora can promote the corresponding standby and subsequently create a new standby using an IP address of the subnet in one of the other Availability Zones.

When Amazon Aurora creates a DB instance in a VPC, it assigns a network interface to your DB instance by using an IP address from your DB subnet group. However, we strongly recommend that you use the DNS name to connect to your DB instance because the underlying IP address changes during failover.

Note

For each DB instance that you run in a VPC, you should reserve at least one address in each subnet in the DB subnet group for use by Amazon Aurora for recovery actions.

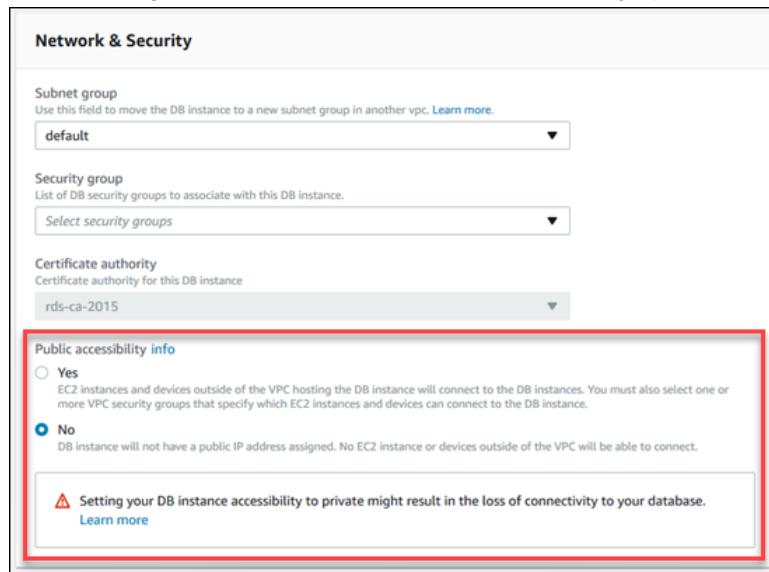
Hiding a DB Instance in a VPC from the Internet

One common Amazon Aurora scenario is to have a VPC in which you have an EC2 instance with a public-facing web application and a DB instance with a database that is not publicly accessible. For example, you can create a VPC that has a public subnet and a private subnet. Amazon EC2 instances that function as web servers can be deployed in the public subnet, and the DB instances are deployed in the private subnet. In such a deployment, only the web servers have access to the DB instances. For an illustration of this scenario, see [A DB Instance in a VPC Accessed by an EC2 Instance in the Same VPC \(p. 246\)](#).

When you launch a DB instance inside a VPC, you can designate whether the DB instance you create has a DNS that resolves to a public IP address by using the *Public accessibility* parameter. This parameter lets you designate whether there is public access to the DB instance. Access to the DB instance is ultimately controlled by the security group it uses, and that public access is not permitted if the security group assigned to the DB instance does not permit it.

You can modify a DB instance to turn on or off public accessibility by modifying the *Public accessibility* parameter. This parameter is modified just like any other DB instance parameter. For more information, see the modifying section for your DB engine.

The following illustration shows the **Public accessibility** option in the **Network & Security** section.



Creating a DB Instance in a VPC

The following procedures help you create a DB instance in a VPC. If your account has a default VPC, you can begin with step 3 because the VPC and DB subnet group have already been created for you. If your AWS account doesn't have a default VPC, or if you want to create an additional VPC, you can create a new VPC.

Note

If you want your DB instance in the VPC to be publicly accessible, you must update the DNS information for the VPC by enabling the VPC attributes *DNS hostnames* and *DNS resolution*. For information about updating the DNS information for a VPC instance, see [Updating DNS Support for Your VPC](#).

Follow these steps to create a DB instance in a VPC:

- [Step 1: Create a VPC \(p. 252\)](#)
- [Step 2: Add Subnets to the VPC \(p. 252\)](#)
- [Step 3: Create a DB Subnet Group \(p. 252\)](#)
- [Step 4: Create a VPC Security Group \(p. 253\)](#)
- [Step 5: Create a DB Instance in the VPC \(p. 253\)](#)

Step 1: Create a VPC

If your AWS account does not have a default VPC or if you want to create an additional VPC, follow the instructions for creating a new VPC. See [Create a VPC with Private and Public Subnets \(p. 255\)](#), or see [Step 1: Create a VPC](#) in the Amazon VPC documentation.

Step 2: Add Subnets to the VPC

Once you have created a VPC, you need to create subnets in at least two Availability Zones. You use these subnets when you create a DB subnet group. If you have a default VPC, a subnet is automatically created for you in each Availability Zone in the AWS Region.

For instructions on how to create subnets in a VPC, see [Create a VPC with Private and Public Subnets \(p. 255\)](#).

Step 3: Create a DB Subnet Group

A DB subnet group is a collection of subnets (typically private) that you create for a VPC and that you then designate for your DB instances. A DB subnet group allows you to specify a particular VPC when you create DB instances using the CLI or API. If you use the console, you can just choose the VPC and subnets you want to use. Each DB subnet group must have at least one subnet in at least two Availability Zones in the AWS Region.

Note

For a DB instance to be publicly accessible, the subnets in the DB subnet group must have an Internet gateway. For more information about Internet gateways for subnets, go to [Internet Gateways](#) in the Amazon VPC documentation.

When you create a DB instance in a VPC, you must choose a DB subnet group. Amazon Aurora then uses that DB subnet group and your preferred Availability Zone to choose a subnet and an IP address within that subnet. Amazon Aurora creates and associates an Elastic Network Interface to your DB instance with that IP address. For Multi-AZ deployments, defining a subnet for two or more Availability Zones in an AWS Region allows Amazon Aurora to create a new standby in another Availability Zone should the need arise. You need to do this even for Single-AZ deployments, just in case you want to convert them to Multi-AZ deployments at some point.

In this step, you create a DB subnet group and add the subnets you created for your VPC.

Console

To create a DB subnet group

1. Open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. In the navigation pane, choose **Subnet groups**.
3. Choose **Create DB Subnet Group**.
4. For **Name**, type the name of your DB subnet group.

5. For **Description**, type a description for your DB subnet group.
6. For **VPC**, choose the VPC that you created.
7. In the **Add subnets** section, choose **Add all the subnets related to this VPC**.

Create DB subnet group

To create a new Subnet Group give it a name, description, and select an existing VPC below. Once you select an existing VPC, you will be able to add subnets related to that VPC.

Subnet group details	
Name	<input type="text" value="mydbsubnetgroup"/>
Description	<input type="text" value="My DB Subnet Group"/>
VPC	VPC Identifier corresponding to the subnets you want to use for the DB subnet group
	<input type="text" value="tutorial-vpc (vpc-971c12ee)"/>

Add subnets																					
Add subnet(s) to this subnet group. You may add subnets one at a time below or add all the subnets related to this VPC. You may make additions/edits after this group is created. A minimum of 2 subnets is required.																					
Add all the subnets related to this VPC																					
Availability zone	<input type="text" value="select an availability zone"/>																				
Subnet	<input type="text" value="select a subnet"/> Add subnet																				
Subnets in this subnet group (4) <table border="1" style="width: 100%; border-collapse: collapse; margin-top: 5px;"> <thead> <tr> <th>Availability zone</th> <th>Subnet ID</th> <th>CIDR block</th> <th>Action</th> </tr> </thead> <tbody> <tr> <td>us-east-1a</td> <td>subnet-d8c8e7f4</td> <td>10.0.2.0/24</td> <td>Remove</td> </tr> <tr> <td>us-east-1f</td> <td>subnet-718fdc7d</td> <td>10.0.3.0/24</td> <td>Remove</td> </tr> <tr> <td>us-east-1a</td> <td>subnet-cbc8e7e7</td> <td>10.0.1.0/24</td> <td>Remove</td> </tr> <tr> <td>us-east-1a</td> <td>subnet-0ccde220</td> <td>10.0.0.0/24</td> <td>Remove</td> </tr> </tbody> </table>		Availability zone	Subnet ID	CIDR block	Action	us-east-1a	subnet-d8c8e7f4	10.0.2.0/24	Remove	us-east-1f	subnet-718fdc7d	10.0.3.0/24	Remove	us-east-1a	subnet-cbc8e7e7	10.0.1.0/24	Remove	us-east-1a	subnet-0ccde220	10.0.0.0/24	Remove
Availability zone	Subnet ID	CIDR block	Action																		
us-east-1a	subnet-d8c8e7f4	10.0.2.0/24	Remove																		
us-east-1f	subnet-718fdc7d	10.0.3.0/24	Remove																		
us-east-1a	subnet-cbc8e7e7	10.0.1.0/24	Remove																		
us-east-1a	subnet-0ccde220	10.0.0.0/24	Remove																		

Cancel
Create

8. Choose **Create**.

Your new DB subnet group appears in the DB subnet groups list on the RDS console. You can choose the DB subnet group to see details, including all of the subnets associated with the group, in the details pane at the bottom of the window.

Step 4: Create a VPC Security Group

Before you create your DB instance, you must create a VPC security group to associate with your DB instance. For instructions on how to create a security group for your DB instance, see [Create a VPC Security Group for a Private DB Instance \(p. 258\)](#), or see [Security Groups for Your VPC](#) in the Amazon VPC documentation.

Step 5: Create a DB Instance in the VPC

In this step, you create a DB instance and use the VPC name, the DB subnet group, and the VPC security group you created in the previous steps.

Note

If you want your DB instance in the VPC to be publicly accessible, you must enable the VPC attributes *DNS hostnames* and *DNS resolution*. For information on updating the DNS information for a VPC instance, see [Updating DNS Support for Your VPC](#).

For details on how to create a DB instance for your DB engine, see the topic following that discusses your DB engine. For each engine, when prompted in the **Network & Security** section, enter the VPC name, the DB subnet group, and the VPC security group you created in the previous steps.

Database Engine	Relevant Documentation
Amazon Aurora	Creating an Amazon Aurora DB Cluster (p. 100)

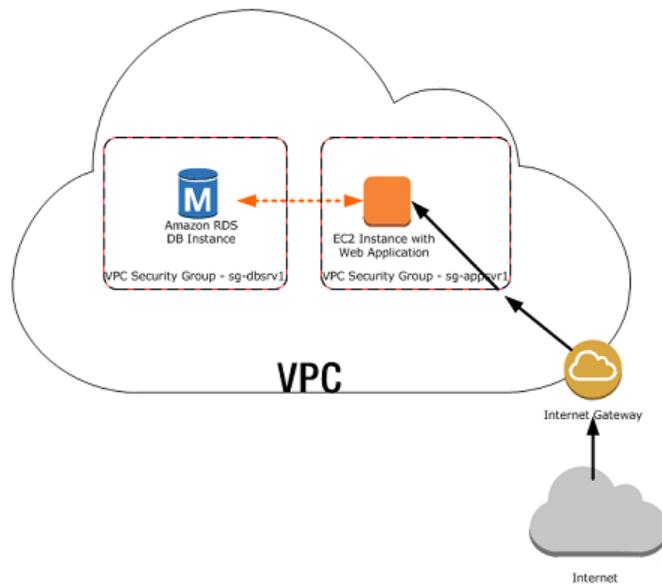
Note

Updating VPCs is not currently supported for Aurora clusters.

Tutorial: Create an Amazon VPC for Use with a DB Instance

A common scenario includes a DB instance in an Amazon VPC, that shares data with a web server that is running in the same VPC. In this tutorial you create the VPC for this scenario.

The following diagram shows this scenario. For information about other scenarios, see [Scenarios for Accessing a DB Instance in a VPC \(p. 245\)](#).



Because your DB instance only needs to be available to your web server, and not to the public Internet, you create a VPC with both public and private subnets. The web server is hosted in the public subnet, so that it can reach the public Internet. The DB instance is hosted in a private subnet. The web server is able to connect to the DB instance because it is hosted within the same VPC, but the DB instance is not available to the public Internet, providing greater security.

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 **Launch VPC Wizard**.
4. On the **Step 1: Select a VPC Configuration** page, choose **VPC with Public and Private Subnets**, and then choose **Select**.
5. On the **Step 2: VPC with Public and Private Subnets** page, set these values:
 - **IPv4 CIDR block:** 10.0.0.0/16
 - **IPv6 CIDR block:** No IPv6 CIDR Block
 - **VPC name:** tutorial-vpc

- **Public subnet's IPv4 CIDR:** 10.0.0.0/24
- **Availability Zone:** us-west-2a
- **Public subnet name:** Tutorial public
- **Private subnet's IPv4 CIDR:** 10.0.1.0/24
- **Availability Zone:** us-west-2a
- **Private subnet name:** Tutorial Private 1
- **Instance type:** t2.small

Important

If you don't see the **Instance type** box in the console, choose **Use a NAT instance instead**. This link is on the right.

Note

If the t2.small instance type is not listed, you can choose a different instance type.

- **Key pair name:** No key pair
- **Service endpoints:** Skip this field.
- **Enable DNS hostnames:** Yes
- **Hardware tenancy:** Default

6. When you're finished, choose **Create VPC**.

Create Additional Subnets

You must have either two private subnets or two public subnets available to create a DB subnet group for a DB instance to use in a VPC. Because the DB instance for this tutorial is private, add a second private subnet to the VPC.

To create an additional subnet

1. Open the Amazon VPC console at <https://console.aws.amazon.com/vpc/>.
2. To add the second private subnet to your VPC, choose **VPC Dashboard**, choose **Subnets**, and then choose **Create subnet**.
3. On the **Create subnet** page, set these values:
 - **Name tag:** Tutorial private 2
 - **VPC:** Choose the VPC that you created in the previous step, for example: vpc-*identifier* (10.0.0.0/16) | tutorial-vpc
 - **Availability Zone:** us-west-2b

Note

Choose an Availability Zone that is different from the one that you chose for the first private subnet.

- **IPv4 CIDR block:** 10.0.2.0/24
4. When you're finished, choose **Create**. Next, choose **Close** on the confirmation page.
 5. To ensure that the second private subnet that you created uses the same route table as the first private subnet, choose **VPC Dashboard**, choose **Subnets**, and then choose the first private subnet that you created for the VPC, Tutorial private 1.
 6. Below the list of subnets, choose the **Route Table** tab, and note the value for **Route Table**—for example: rtb-98b613fd.
 7. In the list of subnets, deselect the first private subnet.
 8. In the list of subnets, choose the second private subnet Tutorial private 2, and choose the **Route Table** tab.

9. If the current route table is not the same as the route table for the first private subnet, choose **Edit route table association**. For **Route Table ID**, choose the route table that you noted earlier—for example: `rtb-98b613fd`. Next, to save your selection, choose **Save**.

Create a VPC Security Group for a Public Web Server

Next you create a security group for public access. To connect to public instances in your VPC, you add inbound rules to your VPC security group that allow traffic to connect from the internet.

To create a VPC security group

1. Open the Amazon VPC console at <https://console.aws.amazon.com/vpc/>.
2. Choose **VPC Dashboard**, choose **Security Groups**, and then choose **Create security group**.
3. On the **Create security group** page, set these values:
 - **Security group name:** `tutorial-securitygroup`
 - **Description:** Tutorial Security Group
 - **VPC:** Choose the VPC that you created earlier, for example: `vpc-identifier (10.0.0.0/16)`
| `tutorial-vpc`
4. To create the security group, choose **Create**. Next, choose **Close** on the confirmation page.

Note the security group ID because you need it later in this tutorial.

To add inbound rules to the security group

1. Determine the IP address to use to connect to instances in your VPC. To determine your public IP address, you can use the service at <https://checkip.amazonaws.com>. An example of an IP address is `203.0.113.25/32`.

If you are connecting through an Internet service provider (ISP) or from behind your firewall without a static IP address, you need to find out the range of IP addresses used by client computers.

Warning

If you use `0.0.0.0/0`, you enable all IP addresses to access your public instances. This approach is acceptable for a short time in a test environment, but it's unsafe for production environments. In production, you'll authorize only a specific IP address or range of addresses to access your instances.

2. Open the Amazon VPC console at <https://console.aws.amazon.com/vpc/>.
3. Choose **VPC Dashboard**, choose **Security Groups**, and then choose the `tutorial-securitygroup` security group that you created in the previous procedure.
4. Under the list of security groups, choose the **Inbound Rules** tab, and then choose **Edit rules**.
5. On the **Edit inbound rules** page, choose **Add Rule**.
6. Set the following values for your new inbound rule to allow Secure Shell (SSH) access to your EC2 instance. If you do this, you can connect to your EC2 instance to install the web server and other utilities, and to upload content for your web server.
 - **Type:** SSH
 - **Source:** The IP address or range from Step 1, for example: `203.0.113.25/32`.
7. Choose **Add rule**.
8. 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`.

9. To save your settings, choose **Save rules**. Next, choose **Close** on the confirmation page.

Create a VPC Security Group for a Private DB Instance

To keep your DB instance private, create a second security group for private access. To connect to private instances in your VPC, you add inbound rules to your VPC security group that allow traffic from your web server only.

To create a VPC security group

1. Open the Amazon VPC console at <https://console.aws.amazon.com/vpc/>.
2. Choose **VPC Dashboard**, choose **Security Groups**, and then choose **Create security group**.
3. On the **Create security group** page, set these values:
 - **Security group name:** tutorial-db-securitygroup
 - **Description:** Tutorial DB Instance Security Group
 - **VPC:** Choose the VPC that you created earlier, for example: vpc-*identifier* (10.0.0.0/16) | tutorial-vpc
4. To create the security group, choose **Create**. Next, choose **Close** on the confirmation page.

To add inbound rules to the 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 the tutorial-db-securitygroup security group that you created in the previous procedure.
3. Under the list of security groups, choose the **Inbound Rules** tab, and then choose **Edit rules**.
4. On the **Edit inbound rules** page, choose **Add Rule**.
5. Set the following values for your new inbound rule to allow MySQL traffic on port 3306 from your EC2 instance. If you do this, you can connect from your web server to your DB instance to store and retrieve data from your web application to your database.
 - **Type:** MySQL/Aurora
 - **Source:** The identifier of the tutorial-securitygroup security group that you created previously in this tutorial, for example: sg-9edd5cfb.
6. To save your settings, choose **Save rules**. Next, choose **Close** on the confirmation page.

Create a DB Subnet Group

A DB subnet group is a collection of subnets that you create in a VPC and that you then designate for your DB instances. A DB subnet group allows you to specify a particular VPC when creating DB instances.

To create a DB subnet group

1. Open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. In the navigation pane, choose **Subnet groups**.
3. Choose **Create DB Subnet Group**.
4. On the **Create DB subnet group** page, set these values in **Subnet group details**:
 - **Name:** tutorial-db-subnet-group
 - **Description:** Tutorial DB Subnet Group
 - **VPC:** tutorial-vpc (vpc-*identifier*)

5. In the **Add subnets** section, choose **Add all the subnets related to this VPC**.
6. Choose **Create**.

Your new DB subnet group appears in the DB subnet groups list on the RDS console. You can click the DB subnet group to see details, including all of the subnets associated with the group, in the details pane at the bottom of the window.

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 \(p. 261\)](#)
- [Modifying an Amazon Aurora DB Cluster \(p. 264\)](#)
- [Adding Aurora Replicas to a DB Cluster \(p. 280\)](#)
- [Managing Performance and Scaling for Aurora DB Clusters \(p. 284\)](#)
- [Working with DB Parameter Groups and DB Cluster Parameter Groups \(p. 286\)](#)
- [Managing Connections with Amazon RDS Proxy \(Preview\) \(p. 308\)](#)
- [Cloning Databases in an Aurora DB Cluster \(p. 335\)](#)
- [Integrating Aurora with Other AWS Services \(p. 350\)](#)
- [Backing Up and Restoring an Amazon Aurora DB Cluster \(p. 379\)](#)
- [Maintaining an Amazon Aurora DB Cluster \(p. 408\)](#)
- [Rebooting a DB Instance in a DB Cluster \(p. 416\)](#)
- [Deleting a DB Instance in an Aurora DB Cluster \(p. 418\)](#)
- [Tagging Amazon RDS Resources \(p. 420\)](#)
- [Working with Amazon Resource Names \(ARNs\) in Amazon RDS \(p. 425\)](#)
- [Amazon Aurora Updates \(p. 432\)](#)

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 \(p. 261\)](#)
- [Limitations for Stopping and Starting Aurora DB Clusters \(p. 261\)](#)
- [Stopping an Aurora DB Cluster \(p. 261\)](#)
- [Possible Operations While an Aurora DB Cluster Is Stopped \(p. 262\)](#)
- [Starting an Aurora DB Cluster \(p. 263\)](#)

Overview of Stopping and Starting an Aurora DB Cluster

During periods where you don't need an Aurora cluster, you can stop all instances in that cluster at once. You can start the cluster again anytime you need to use it. Starting and stopping simplifies the setup and teardown processes for clusters used for development, testing, or similar activities that don't require continuous availability. You can perform all the AWS Management Console procedures involved with only a single action, regardless of how many instances are in the cluster.

While your DB cluster is stopped, you are charged only for cluster storage, manual snapshots, and automated backup storage within your specified retention window. You aren't charged for any DB instance hours. Aurora automatically starts your DB cluster after seven days so that it doesn't fall behind any required maintenance updates.

To minimize charges for a lightly loaded Aurora cluster, you can stop the cluster instead of deleting all of its Aurora Replicas. For clusters with more than one or two instances, frequently deleting and recreating the DB instances is only practical using the AWS CLI or Amazon RDS API. Such a sequence of operations can also be difficult to perform in the right order, for example to delete all Aurora Replicas before deleting the primary instance to avoid activating the failover mechanism.

Don't use starting and stopping if you need to keep your DB cluster running but it has more capacity than you need. If your cluster is too costly or not very busy, delete one or more DB instances or change all your DB instances to a small instance class. You can't stop an individual Aurora DB instance.

Limitations for Stopping and Starting Aurora DB Clusters

Some Aurora clusters can't be stopped and started:

- You can't stop and start a cluster that's part of an [Aurora global database \(p. 28\)](#).
- You can't stop and start a cluster that uses the [Aurora parallel query \(p. 644\)](#) feature.

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.
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 \(p. 381\)](#).

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.

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 \(p. 267\)](#).

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 \(p. 267\)](#). 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 \(p. 265\)](#).

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 \(p. 267\)](#).

Note

Some settings only apply to DB instances. To change those settings, follow the instructions in [Modify a DB Instance in a DB Cluster \(p. 265\)](#).

Example

The following command modifies `mydbcluster` by setting the backup retention period to 1 week (7 days).

For Linux, OS X, 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 \(p. 267\)](#).

Note

Some settings only apply to DB instances. To change those settings, follow the instructions in [Modify a DB Instance in a DB Cluster \(p. 265\)](#).

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 put into the pending modifications queue. During the next maintenance window, any pending changes in the queue are applied. If you choose to apply changes immediately, your new changes and any changes in the pending modifications queue are applied.

Important

If any of the pending 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.

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 \(p. 267\)](#).

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 \(p. 264\)](#).

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 \(p. 267\)](#).

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 \(p. 267\)](#).

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 \(p. 264\)](#).

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, OS X, 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 \(p. 267\)](#).

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 \(p. 264\)](#).

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.

Setting and Description	Method	Scope	Downtime Notes
<p>Auto minor version upgrade</p> <p>Whether you want the DB instance to receive preferred minor engine version upgrades automatically when they become available. Upgrades are installed only during your scheduled maintenance window.</p> <p>The Auto minor version upgrade setting only applies to Aurora PostgreSQL DB clusters.</p> <p>For more information about engine updates for Aurora PostgreSQL, see Database Engine Updates for Amazon Aurora PostgreSQL (p. 970).</p> <p>For more information about engine updates for Aurora MySQL, see Database Engine Updates for Amazon Aurora MySQL (p. 794).</p>	<p>Using the AWS Management Console, Modify a DB Instance in a DB Cluster (p. 265).</p> <p>Using the AWS CLI, run <code>modify-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 ModifyDBInstance and set the <code>AutoMinorVersionUpgrade</code> parameter.</p>	The entire DB cluster	An outage doesn't occur during this change.
<p>Backup retention period</p> <p>The number of days that automatic backups are retained. The minimum value is 1.</p>	<p>Using the AWS Management Console, Modifying the DB Cluster by Using the Console, CLI, and API (p. 264).</p> <p>Using the AWS CLI, run <code>modify-db-cluster</code></p>	The entire DB cluster	An outage doesn't occur during this change.

Setting and Description	Method	Scope	Downtime Notes
<p>For more information, see Backups (p. 380).</p>	<p>and set the <code>--backup-retention-period</code> option.</p> <p>Using the RDS API, call <code>ModifyDBCluster</code> and set the <code>BackupRetentionPeriod</code> parameter.</p>		
<p>Certificate Authority</p> <p>The certificate that you want to use.</p>	<p>Using the AWS Management Console, <code>Modify a DB Instance in a DB Cluster (p. 265)</code>.</p> <p>Using the AWS CLI, run <code>modify-db-instance</code> and set the <code>--ca-certificate-identifier</code> option.</p> <p>Using the RDS API, call <code>ModifyDBInstance</code> and set the <code>CACertificateIdentifier</code> parameter.</p>	Only the specified DB instance	An outage occurs during this change. The DB instance is rebooted.
<p>Copy tags to snapshots</p> <p>Select to specify that tags defined for this DB cluster are copied to DB snapshots created from this DB cluster. For more information, see <code>Tagging Amazon RDS Resources (p. 420)</code>.</p>	<p>Using the AWS Management Console, <code>Modifying the DB Cluster by Using the Console, CLI, and API (p. 264)</code>.</p> <p>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.</p> <p>Using the RDS API, call <code>ModifyDBCluster</code> and set the <code>CopyTagsToSnapshot</code> parameter.</p>	The entire DB cluster	An outage doesn't occur during this change.

Setting and Description	Method	Scope	Downtime Notes
<p>Data API</p> <p>You can access Aurora Serverless with web services-based applications, including AWS Lambda and AWS AppSync. This setting only applies to an Aurora Serverless DB cluster.</p> <p>For more information, see Using the Data API for Aurora Serverless (p. 139).</p>	<p>Using the AWS Management Console, Modifying the DB Cluster by Using the Console, CLI, and API (p. 264).</p> <p>Using the AWS CLI, run <code>modify-db-cluster</code> and set the <code>--enable-http-endpoint</code> option.</p> <p>Using the RDS API, call <code>ModifyDBCluster</code> and set the <code>EnableHttpEndpoint</code> parameter.</p>	The entire DB cluster	An outage doesn't occur during this change.
<p>Database authentication</p> <p>The database authentication option you want to use.</p> <p>Choose Password authentication to authenticate database users with database passwords only.</p> <p>Choose Password and IAM DB authentication to authenticate database users with database passwords and user credentials through IAM users and roles. For more information, see IAM Database Authentication (p. 207).</p>	<p>Using the AWS Management Console, Modifying the DB Cluster by Using the Console, CLI, and API (p. 264).</p> <p>Using the AWS CLI, run <code>modify-db-cluster</code> and set the <code>--enable-iam-database-authentication --no-enable-iam-database-authentication</code> option.</p> <p>Using the RDS API, call <code>ModifyDBCluster</code> and set the <code>EnableIAMDatabaseAuthentication</code> parameter.</p>	The entire DB cluster	An outage doesn't occur during this change.

Setting and Description	Method	Scope	Downtime Notes
Database port The port that you want to use to access the DB cluster.	Using the AWS Management Console, Modifying the DB Cluster by Using the Console, CLI, and API (p. 264) . Using the AWS CLI, run <code>modify-db-cluster</code> and set the <code>--port</code> option. Using the RDS API, call <code>ModifyDBCluster</code> and set the <code>Port</code> parameter.	The entire DB cluster	An outage occurs during this change. All of the DB instances in the DB cluster are rebooted immediately.
DB cluster identifier The DB cluster identifier. This value is stored as a lowercase string. When you change the DB cluster identifier, the DB cluster endpoint changes, and the endpoints of the DB instances in the DB cluster change.	Using the AWS Management Console, Modifying the DB Cluster by Using the Console, CLI, and API (p. 264) . Using the AWS CLI, run <code>modify-db-cluster</code> and set the <code>--new-db-cluster-identifier</code> option. Using the RDS API, call <code>ModifyDBCluster</code> and set the <code>NewDBClusterIdentifier</code> parameter.	The entire DB cluster	An outage doesn't occur during this change.
DB cluster parameter group The DB cluster parameter group that you want associated with the DB cluster. For more information, see Working with DB Parameter Groups and DB Cluster Parameter Groups (p. 286) .	Using the AWS Management Console, Modifying the DB Cluster by Using the Console, CLI, and API (p. 264) . Using the AWS CLI, run <code>modify-db-cluster</code> and set the <code>--db-cluster-parameter-group-name</code> option. Using the RDS API, call <code>ModifyDBCluster</code> and set the <code>DBClusterParameterGroupName</code> parameter.	The entire DB cluster	An outage doesn't occur during this change. When you change the parameter group, changes to some parameters are applied to the DB instances in the DB cluster immediately without a reboot. Changes to other parameters are applied only after the DB instances in the DB cluster are rebooted.

Setting and Description	Method	Scope	Downtime Notes
<p>DB engine version</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>Currently, you can only use this setting to upgrade the minor engine version of an Aurora PostgreSQL DB cluster.</p>	<p>Using the AWS Management Console, Modifying the DB Cluster by Using the Console, CLI, and API (p. 264).</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 <code>ModifyDBCluster</code> and set the <code>EngineVersion</code> parameter.</p>	The entire DB cluster	An outage occurs during this change.
<p>DB instance class</p> <p>The DB instance class that you want to use.</p> <p>For more information, see Choosing the DB Instance Class (p. 76).</p>	<p>Using the AWS Management Console, Modify a DB Instance in a DB Cluster (p. 265).</p> <p>Using the AWS CLI, run <code>modify-db-instance</code> and set the <code>--db-instance-class</code> option.</p> <p>Using the RDS API, call <code>ModifyDBInstance</code> and set the <code>DBInstanceClass</code> parameter.</p>	Only the specified DB instance	An outage occurs during this change.
<p>DB instance identifier</p> <p>The DB instance identifier. This value is stored as a lowercase string.</p>	<p>Using the AWS Management Console, Modify a DB Instance in a DB Cluster (p. 265).</p> <p>Using the AWS CLI, run <code>modify-db-instance</code> and set the <code>--new-db-instance-identifier</code> option.</p> <p>Using the RDS API, call <code>ModifyDBInstance</code> and set the <code>NewDBInstanceIdentifier</code> parameter.</p>	Only the specified DB instance	An outage occurs during this change. The DB instance is rebooted.

Setting and Description	Method	Scope	Downtime Notes
DB parameter group The DB parameter group that you want associated with the DB instance. For more information, see Working with DB Parameter Groups and DB Cluster Parameter Groups (p. 286) .	Using the AWS Management Console, Modify a DB Instance in a DB Cluster (p. 265) . Using the AWS CLI, run <code>modify-db-instance</code> and set the <code>--db-parameter-group-name</code> option. Using the RDS API, call ModifyDBInstance and set the <code>DBParameterGroupName</code> parameter.	Only the specified DB instance	An outage doesn't occur during this change. When you change the 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.
Deletion protection Enable deletion protection to prevent your DB cluster from being deleted. For more information, see Deleting a DB Instance in an Aurora DB Cluster (p. 418) .	Using the AWS Management Console, Modifying the DB Cluster by Using the Console, CLI, and API (p. 264) . Using the AWS CLI, run <code>modify-db-cluster</code> and set the <code>--deletion-protection --no-deletion-protection</code> option. Using the RDS API, call ModifyDBCluster and set the <code>DeletionProtection</code> parameter.	The entire DB cluster	An outage doesn't occur during this change.

Setting and Description	Method	Scope	Downtime Notes
Enhanced monitoring Enable enhanced monitoring to enable gathering metrics in real time for the operating system that your DB instance runs on. For more information, see Enhanced Monitoring (p. 469) .	Using the AWS Management Console, Modify a DB Instance in a DB Cluster (p. 265) . Using the AWS CLI, run <code>modify-db-instance</code> and set the <code>--monitoring-role-arn</code> and <code>--monitoring-interval</code> options. Using the RDS API, call ModifyDBInstance and set the <code>MonitoringRoleArn</code> and <code>MonitoringInterval</code> parameters.	Only the specified DB instance	An outage doesn't occur during this change.
Log exports Select the log types to publish to Amazon CloudWatch Logs. For more information, see MySQL Database Log Files (p. 567) .	Using the AWS Management Console, Modifying the DB Cluster by Using the Console, CLI, and API (p. 264) . 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 ModifyDBCluster 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>Maintenance window</p> <p>The time range during which system maintenance occurs. System maintenance includes upgrades, if applicable. The maintenance window is a start time in Universal Coordinated Time (UTC), and a duration in hours.</p> <p>If you set the window to the current time, there must be at least 30 minutes between the current time and end of the window to ensure any pending changes are applied.</p> <p>You can set the maintenance window independently for the DB cluster and for each DB instance in the DB cluster. When the scope of a modification is the entire DB cluster, the modification is performed during the DB cluster maintenance window. When the scope of a modification is the a DB instance, the modification is performed during maintenance window of that DB instance.</p> <p>For more information, see The Amazon RDS Maintenance Window (p. 411).</p>	<p>To change the maintenance window for the DB cluster using the AWS Management Console, Modifying the DB Cluster by Using the Console, CLI, and API (p. 264).</p> <p>To change the maintenance window for a DB instance using the AWS Management Console, Modify a DB Instance in a DB Cluster (p. 265).</p> <p>To change the maintenance window for the DB cluster using the AWS CLI, run <code>modify-db-cluster</code> and set the <code>--preferred-maintenance-window</code> option.</p> <p>To change the maintenance window for a DB instance using the AWS CLI, run <code>modify-db-instance</code> and set the <code>--preferred-maintenance-window</code> option.</p> <p>To change the maintenance window for the DB cluster using the RDS API, call ModifyDBCluster and set the <code>PreferredMaintenanceWindow</code> parameter.</p> <p>To change the maintenance window for a DB instance using the RDS API, call ModifyDBInstance and set the <code>PreferredMaintenanceWindow</code> parameter.</p>	<p>The entire DB cluster or a single DB instance</p>	<p>If there are one or more pending actions that cause an outage, and the maintenance window is changed to include the current time, then those pending actions are applied immediately, and an outage occurs.</p>

Setting and Description	Method	Scope	Downtime Notes
New master password The password for your master user. The password must contain from 8 to 41 alphanumeric characters.	Using the AWS Management Console, Modifying the DB Cluster by Using the Console, CLI, and API (p. 264) . Using the AWS CLI, run <code>modify-db-cluster</code> and set the <code>--master-user-password</code> option. Using the RDS API, call <code>ModifyDBCluster</code> and set the <code>MasterUserPassword</code> parameter.	The entire DB cluster	An outage doesn't occur during this change.
Performance Insights Whether to enable Performance Insights, a tool that monitors your DB instance load so that you can analyze and troubleshoot your database performance. For more information, see Using Amazon RDS Performance Insights (p. 476) .	Using the AWS Management Console, Modify a DB Instance in a DB Cluster (p. 265) . Using the AWS CLI, run <code>modify-db-instance</code> and set the <code>--enable-performance-insights --no-enable-performance-insights</code> option. Using the RDS API, call <code>ModifyDBInstance</code> and set the <code>EnablePerformanceInsights</code> parameter.	Only the specified DB instance	An outage doesn't occur during this change.

Setting and Description	Method	Scope	Downtime Notes
<p>Performance Insights AWS KMS key identifier</p> <p>The AWS KMS key identifier for encryption of Performance Insights data. The KMS key ID is the Amazon Resource Name (ARN), KMS key identifier, or the KMS key alias for the KMS encryption key.</p> <p>For more information, see Enabling Performance Insights (p. 477).</p>	<p>Using the AWS Management Console, Modify a DB Instance in a DB Cluster (p. 265).</p> <p>Using the AWS CLI, run <code>modify-db-instance</code> and set the <code>--performance-insights-kms-key-id</code> option.</p> <p>Using the RDS API, call <code>ModifyDBInstance</code> and set the <code>PerformanceInsightsKMSKeyId</code> parameter.</p>	Only the specified DB instance	An outage doesn't occur during this change.
<p>Performance Insights retention period</p> <p>The amount of time, in days, to retain Performance Insights data. Valid values are 7 or 731 (2 years).</p> <p>For more information, see Enabling Performance Insights (p. 477).</p>	<p>Using the AWS Management Console, Modify a DB Instance in a DB Cluster (p. 265).</p> <p>Using the AWS CLI, run <code>modify-db-instance</code> and set the <code>--performance-insights-retention-period</code> option.</p> <p>Using the RDS API, call <code>ModifyDBInstance</code> and set the <code>PerformanceInsightsRetentionPeriod</code> parameter.</p>	Only the specified DB instance	An outage doesn't occur during this change.
<p>Promotion tier</p> <p>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.</p> <p>For more information, see Fault Tolerance for an Aurora DB Cluster (p. 380).</p>	<p>Using the AWS Management Console, Modify a DB Instance in a DB Cluster (p. 265).</p> <p>Using the AWS CLI, run <code>modify-db-instance</code> and set the <code>--promotion-tier</code> option.</p> <p>Using the RDS API, call <code>ModifyDBInstance</code> and set the <code>PromotionTier</code> parameter.</p>	Only the specified DB instance	An outage doesn't occur during this change.

Setting and Description	Method	Scope	Downtime Notes
Public accessibility Whether to give the DB instance a public IP address, meaning that it is accessible outside the VPC. To be publicly accessible, the DB instance also has to be in a public subnet in the VPC. When the DB instance is not publicly accessible, it is accessible only from inside the VPC. For more information, see Hiding a DB Instance in a VPC from the Internet (p. 251) .	Using the AWS Management Console, Modify a DB Instance in a DB Cluster (p. 265) . Using the AWS CLI, run <code>modify-db-instance</code> and set the <code>--publicly-accessible --no-publicly-accessible</code> option. Using the RDS API, call <code>ModifyDBInstance</code> and set the <code>PubliclyAccessible</code> parameter.	Only the specified DB instance	An outage doesn't occur during this change.
Scaling configuration The scaling properties of the DB cluster. You can only modify scaling properties for DB clusters in serverless DB engine mode. This setting is available only for Aurora MySQL. For information about Aurora Serverless, see Using Amazon Aurora Serverless (p. 116) .	Using the AWS Management Console, Modifying the DB Cluster by Using the Console, CLI, and API (p. 264) . Using the AWS CLI, run <code>modify-db-cluster</code> and set the <code>--scaling-configuration</code> option. Using the RDS API, call <code>ModifyDBCluster</code> and set the <code>ScalingConfiguration</code> parameter.	The entire DB cluster	An outage doesn't occur during this change.

Setting and Description	Method	Scope	Downtime Notes
Security group The security group you want associated with the DB cluster. For more information, see Controlling Access with Security Groups (p. 231) .	Using the AWS Management Console, Modifying the DB Cluster by Using the Console, CLI, and API (p. 264) . Using the AWS CLI, run <code>modify-db-cluster</code> and set the <code>--vpc-security-group-ids</code> option. Using the RDS API, call <code>ModifyDBCluster</code> and set the <code>VpcSecurityGroupIds</code> parameter.	The entire DB cluster	An outage doesn't occur during this change.
Target Backtrack window The amount of time you want to be able to backtrack your DB cluster, in seconds. This setting is available only for Aurora MySQL and only if the DB cluster was created with Backtrack enabled.	Using the AWS Management Console, Modifying the DB Cluster by Using the Console, CLI, and API (p. 264) . Using the AWS CLI, run <code>modify-db-cluster</code> and set the <code>--backtrack-window</code> option. Using the RDS API, call <code>ModifyDBCluster</code> and set the <code>BacktrackWindow</code> parameter.	The entire DB cluster	An outage doesn't occur during this change.

Settings That Do Not Apply to Amazon Aurora

The following settings in the AWS CLI command `modify-db-instance` and the RDS API operation `ModifyDBInstance` do not apply to Amazon Aurora.

Note

The AWS Management Console doesn't allow you to modify these settings for Aurora.

AWS CLI Setting	RDS API Setting
<code>--allocated-storage</code>	<code>AllocatedStorage</code>
<code>--allow-major-version-upgrade --no-allow-major-version-upgrade</code>	<code>AllowMajorVersionUpgrade</code>

AWS CLI Setting	RDS API Setting
--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
--option-group-name	OptionGroupName
--preferred-backup-window	PreferredBackupWindow
--processor-features	ProcessorFeatures
--storage-type	StorageType
--tde-credential-arn	TdeCredentialArn
--tde-credential-password	TdeCredentialPassword
--use-default-processor-features --no-use-default-processor-features	UseDefaultProcessorFeatures

Note

Although the preferred backup window can be set for Aurora, the setting is ignored. Aurora backups are continuous and incremental.

Adding Aurora Replicas to a DB Cluster

In an Aurora DB cluster with single-master replication, there is one primary DB instance and up to 15 Aurora Replicas. The primary DB instance supports read and write operations, and performs all of the data modifications to the cluster volume. Aurora Replicas connect to the same storage volume as the primary DB instance and support only read operations. Aurora Replicas can offload read workloads from the primary DB instance.

We recommend that you distribute the primary instance and Aurora Replicas in your DB cluster over multiple Availability Zones to improve the availability of your DB cluster. For more information, see [Region Availability \(p. 80\)](#).

You can add Aurora Replicas to a DB cluster using the AWS Management Console, the AWS CLI, or the RDS API.

To remove an Aurora Replica from a DB cluster, delete the Aurora Replica DB instance by following the instructions in [Deleting a DB Instance in an Aurora DB Cluster \(p. 418\)](#).

For more information about Aurora Replicas, see [Aurora Replicas \(p. 47\)](#).

Note

Amazon Aurora also supports replication with an external database, or an RDS DB instance. When using Amazon Aurora, your RDS DB instance must be in the same AWS Region. For more information, see [Replication with Amazon Aurora \(p. 47\)](#).

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. For **Actions**, choose **Add reader**.

The **Add reader** page appears.

4. On the **Add reader** page, specify options for your Aurora Replica. The following table shows settings for an Aurora Replica.

For This Option	Do This
Availability zone	Determine if you want to specify a particular Availability Zone. The list includes only those Availability Zones that are mapped by the DB subnet group you specified earlier. For more information about Availability Zones, see Choosing the Regions and Availability Zones (p. 80) .
Publicly accessible	Select Yes to give the Aurora Replica a public IP address; otherwise, select No. For more information about hiding Aurora Replicas from public access, see Hiding a DB Instance in a VPC from the Internet (p. 251) .
Encryption	Select Enable encryption to enable encryption at rest for this Aurora Replica. For more information, see Encrypting Amazon Aurora Resources (p. 175) .

For This Option	Do This
DB instance class	Select a DB instance class that defines the processing and memory requirements for the Aurora Replica. For more information about DB instance class options, see Choosing the DB Instance Class (p. 76) .
Aurora replica source	Select the identifier of the primary instance to create an Aurora Replica for.
DB instance identifier	Type 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 <code>aurora-read-instance1</code> .
Priority	Choose a failover priority for the instance. If you don't select a value, the default is tier-1 . This priority determines the order in which Aurora Replicas are promoted when recovering from a primary instance failure. For more information, see Fault Tolerance for an Aurora DB Cluster (p. 380) .
Database port	The port for an Aurora Replica is the same as the port for the DB cluster.
DB parameter group	Select a parameter group. Aurora has a default parameter group you can use, or you can create your own parameter group. For more information about parameter groups, see Working with DB Parameter Groups and DB Cluster Parameter Groups (p. 286) .
Enhanced monitoring	Choose Enable enhanced monitoring to enable gathering metrics in real time for the operating system that your DB cluster runs on. For more information, see Enhanced Monitoring (p. 469) .
Monitoring Role	Only available if Enhanced Monitoring is set to Enable enhanced monitoring . Choose the IAM role that you created to permit Amazon RDS to communicate with Amazon CloudWatch Logs for you, or choose Default to have RDS create a role for you named <code>rds-monitoring-role</code> . For more information, see Enhanced Monitoring (p. 469) .
Granularity	Only available if Enhanced Monitoring is set to Enable enhanced monitoring . Set the interval, in seconds, between when metrics are collected for your DB cluster.

For This Option	Do This
Auto minor version upgrade	<p>Select Enable auto minor version upgrade if you want to enable your Aurora DB cluster to receive minor DB Engine version upgrades automatically when they become available.</p> <p>The Auto minor version upgrade setting only applies to Aurora PostgreSQL DB clusters.</p> <p>For more information about engine updates for Aurora PostgreSQL, see Database Engine Updates for Amazon Aurora PostgreSQL (p. 970).</p> <p>For more information about engine updates for Aurora MySQL, see Database Engine Updates for Amazon Aurora MySQL (p. 794).</p>

5. 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, OS X, 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.r4.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.r4.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, OS X, 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.r4.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.r4.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 \(p. 284\)](#)
- [Instance Scaling \(p. 284\)](#)
- [Read Scaling \(p. 284\)](#)
- [Managing Connections \(p. 285\)](#)
- [Managing Query Execution Plans \(p. 285\)](#)

Storage Scaling

Aurora storage automatically scales with the data in your cluster volume. As your data grows, your cluster volume storage grows in 10 gibibyte (GiB) increments up to 64 TiB.

The size of your cluster volume is checked on an hourly basis to determine your storage costs. For pricing information, see the [Aurora pricing page](#).

When Aurora data is removed, such as by dropping a table or partition, the overall allocated space remains the same. The free space is reused automatically when data volume increases in the future.

Note

Storage costs are based on the storage "high water mark," the maximum amount that was allocated for the Aurora cluster at any point in time. Thus, you can manage costs by avoiding extract, transform, load (ETL) practices that create large volumes of temporary information. Similarly, you can manage costs by avoiding ETL practices that load large volumes of new data before removing unneeded older data.

If removing data from an Aurora cluster results in a substantial amount of allocated but unused space, resetting the high water mark requires doing a logical data dump and restore to a new cluster, using a tool such as `mysqldump`. Creating and restoring a snapshot does *not* reduce the allocated storage because the physical layout of the underlying storage remains the same in the restored snapshot.

Instance Scaling

You can scale your Aurora DB cluster as needed by modifying the DB instance class for each DB instance in the DB cluster. Aurora supports several DB instance classes optimized for Aurora, depending on database engine compatibility.

Database Engine	Instance Scaling
Amazon Aurora MySQL	See Scaling Aurora MySQL DB Instances (p. 623)
Amazon Aurora PostgreSQL	See Scaling Aurora PostgreSQL DB Instances (p. 910)

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 \(p. 280\)](#).

Managing Connections

The maximum number of connections allowed to an Aurora DB instance is determined by the `max_connections` parameter in the instance-level parameter group for the DB instance. The default value of that parameter varies depends on the DB instance class used for the DB instance and database engine compatibility.

Database Engine	max_connections default Value
Amazon Aurora MySQL	See Maximum Connections to an Aurora MySQL DB Instance (p. 624)
Amazon Aurora PostgreSQL	See Maximum Connections to an Aurora PostgreSQL DB Instance (p. 910)

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 \(p. 916\)](#).

Working with DB Parameter Groups and DB Cluster Parameter Groups

You manage your DB engine configuration by associating your DB instances and Aurora clusters with parameter groups. Amazon RDS defines parameter groups with default settings that apply to newly created DB instances and Aurora clusters. You can define your own parameter groups with customized settings. Then you can modify your DB instances and Aurora clusters to use your own parameter groups.

A *DB parameter group* acts as a container for engine configuration values that are applied to one or more DB instances. DB parameter groups apply to DB instances in both Amazon RDS and Aurora. These configuration settings apply to properties that can vary among the DB instances within an Aurora cluster, such as the sizes for memory buffers.

A *DB cluster parameter group* acts as a container for engine configuration values that are applied to every DB instance in an Aurora DB cluster. For example, the Aurora shared storage model requires that every DB instance in an Aurora cluster use the same setting for parameters such as `innodb_file_per_table`. Thus, parameters that affect the physical storage layout are part of the cluster parameter group. The DB cluster parameter group also includes default values for all the instance-level parameters.

If you create a DB instance without specifying a DB parameter group, the DB instance uses a default DB parameter group. Likewise, if you create an Aurora DB cluster without specifying a DB cluster parameter group, the DB cluster uses a default DB cluster parameter group. Each default parameter group contains database engine defaults and Amazon RDS system defaults based on the engine, compute class, and allocated storage of the instance. You can't modify the parameter settings of a default parameter group. Instead, you create your own parameter group where you choose your own parameter settings. Not all DB engine parameters can be changed in a parameter group that you create.

If you want to use your own parameter group, you create a new parameter group and modify the parameters that you want to. You then modify your DB instance or DB cluster to use the new parameter group. If you update parameters within a DB parameter group, the changes apply to all DB instances that are associated with that parameter group. Likewise, if you update parameters within a DB cluster parameter group, the changes apply to all Aurora clusters that are associated with that DB cluster parameter group.

You can copy an existing DB parameter group with the AWS CLI [copy-db-parameter-group](#) command. You can copy an existing DB cluster parameter group with the AWS CLI [copy-db-cluster-parameter-group](#) command. Copying a parameter group can be convenient when you want to include most of an existing parameter group's custom parameters and values in a new parameter group.

Here are some important points about working with parameters in a parameter group:

- When you change a dynamic parameter and save the parameter group, the change is applied immediately regardless of the **Apply Immediately** setting. When you change a static parameter and save the DB parameter group, the parameter change takes effect after you manually reboot the DB instance. You can reboot a DB instance using the RDS console or by 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, such as calling `ModifyDBInstance` to change DB instance class or scale storage.

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 change the DB parameter group associated with a DB instance, you must manually reboot the instance before the new DB parameter group is used by the DB instance.

Note

Reboot isn't required when you change the DB cluster parameter group associated with a DB cluster.

- You can specify the value for a parameter as an integer or as an integer expression built from formulas, variables, functions, and operators. Functions can include a mathematical log expression. For more information, see [DB Parameter Values \(p. 305\)](#).
- Set any parameters that relate to the character set or collation of your database in your parameter group before creating the DB instance and before you create a database in your DB instance. This ensures that the default database and new databases in your DB instance use the character set and collation values that you specify. If you change character set or collation parameters for your DB instance, the parameter changes are not applied to existing databases.

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;
```

- Improperly setting parameters in a parameter group can have unintended adverse effects, including degraded performance and system instability. Always exercise caution when modifying database parameters and back up your data before modifying a parameter group. Try out parameter group setting changes on a test DB instance before applying those parameter group changes to a production DB instance.
- For an Aurora global database, you can specify different configuration settings for the individual Aurora clusters. Make sure that the settings are similar enough to produce consistent behavior if you promote a secondary cluster to be the primary cluster. For example, use the same settings for time zones and character sets across all the clusters of an Aurora global database.
- To determine the supported parameters for your DB engine, you can view the parameters in the DB parameter group and DB cluster parameter group used by the DB cluster. For more information, see [Viewing Parameter Values for a DB Parameter Group \(p. 302\)](#) and [Viewing Parameter Values for a DB Cluster Parameter Group \(p. 303\)](#).

Topics

- [Amazon Aurora DB Cluster and DB Instance Parameters \(p. 288\)](#)
- [Creating a DB Parameter Group \(p. 289\)](#)
- [Creating a DB Cluster Parameter Group \(p. 290\)](#)
- [Modifying Parameters in a DB Parameter Group \(p. 291\)](#)
- [Modifying Parameters in a DB Cluster Parameter Group \(p. 295\)](#)
- [Copying a DB Parameter Group \(p. 297\)](#)
- [Copying a DB Cluster Parameter Group \(p. 299\)](#)
- [Listing DB Parameter Groups \(p. 300\)](#)
- [Listing DB Cluster Parameter Groups \(p. 301\)](#)
- [Viewing Parameter Values for a DB Parameter Group \(p. 302\)](#)
- [Viewing Parameter Values for a DB Cluster Parameter Group \(p. 303\)](#)
- [Comparing Parameter Groups \(p. 304\)](#)
- [DB Parameter Values \(p. 305\)](#)

Amazon Aurora DB Cluster and DB Instance Parameters

Aurora uses a two-level system of configuration settings, as follows:

- Parameters in a *DB cluster parameter group* apply to every DB instance in a DB cluster. Your data is stored in the Aurora shared storage subsystem. Because of this, all parameters related to physical layout of table data must be the same for all DB instances in an Aurora cluster. Likewise, because Aurora DB instances are connected by replication, all the parameters for replication settings must be identical throughout an Aurora cluster.
- Parameters in a *DB parameter group* apply to a single DB instance in an Aurora DB cluster. These parameters are related to aspects such as memory usage that you can vary across DB instances in the same Aurora cluster. For example, a cluster often contains DB instances with different AWS instance classes.

Every Aurora cluster is associated with a DB cluster parameter group. Each DB instance within the cluster inherits the settings from that DB cluster parameter group, and is associated with a DB parameter group. Aurora assigns default parameter groups when you create a cluster or a new DB instance, based on the specified database engine and version. You can change the parameter groups later to ones that you create, where you can edit the parameter values.

The DB cluster parameter groups also include default values for all the instance-level parameters from the DB parameter group. These defaults are mainly intended for configuring Aurora Serverless clusters, which are only associated with DB cluster parameter groups, not DB parameter groups. You can modify the instance-level parameter settings in the DB cluster parameter group. Then, Aurora applies those settings to each new DB instance that's added to a Serverless cluster. To learn more about configuration settings for Aurora Serverless clusters and which settings you can modify, see [Aurora Serverless and Parameter Groups \(p. 121\)](#).

For non-Serverless clusters, any configuration values that you modify in the DB cluster parameter group override default values in the DB parameter group. If you edit the corresponding values in the DB parameter group, those values override the settings from the DB cluster parameter group.

Any DB parameter settings that you modify take precedence over the DB cluster parameter group values, even if you change the configuration parameters back to their default values. You can see which parameters are overridden by using the `describe-db-parameters` AWS CLI command or the `DescribeDBParameters` RDS API. The `Source` field contains the value `user` if you modified that parameter. To reset one or more parameters so that the value from the DB cluster parameter group takes precedence, use the `reset-db-parameter-group` AWS CLI command or the `ResetDBParameterGroup` RDS API operation.

The DB cluster and DB instance parameters available to you in Aurora vary depending on database engine compatibility.

Database Engine	Parameters
Aurora MySQL	See Aurora MySQL Parameters (p. 773) . For Aurora Serverless clusters, see additional details in Aurora Serverless and Parameter Groups (p. 121) .
Aurora PostgreSQL	See Amazon Aurora PostgreSQL Parameters (p. 960) .

Creating a DB Parameter Group

You can create a new DB parameter group using the AWS Management Console, the AWS CLI, or the RDS API.

Console

To create a DB parameter group

1. Sign in to the AWS Management Console and open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. In the navigation pane, choose **Parameter groups**.
3. Choose **Create parameter group**.

The **Create parameter group** window appears.

4. In the **Parameter group family** list, select a DB parameter group family.
5. In the **Type** list, select **DB Parameter Group**.
6. In the **Group name** box, enter the name of the new DB parameter group.
7. In the **Description** box, enter a description for the new DB parameter group.
8. Choose **Create**.

AWS CLI

To create a DB parameter group, use the AWS CLI `create-db-parameter-group` command. The following example creates a DB parameter group named *mydbparametergroup* for MySQL version 5.6 with a description of "My new parameter group."

Include the following required parameters:

- `--db-parameter-group-name`
- `--db-parameter-group-family`
- `--description`

To list all of the available parameter group families, use the following command:

```
aws rds describe-db-engine-versions --query "DBEngineVersions[].[DBParameterGroupFamily"]"
```

Note

The output contains duplicates.

Example

For Linux, OS X, or Unix:

```
aws rds create-db-parameter-group \
--db-parameter-group-name mydbparametergroup \
--db-parameter-group-family aurora5.6 \
--description "My new parameter group"
```

For Windows:

```
aws rds create-db-parameter-group ^
--db-parameter-group-name mydbparametergroup ^
--db-parameter-group-family aurora5.6 ^
--description "My new parameter group"
```

This command produces output similar to the following:

```
DBPARAMETERGROUP  mydbparametergroup  aurora5.6  My new parameter group
```

RDS API

To create a DB parameter group, use the RDS API [CreateDBParameterGroup](#) operation.

Include the following required parameters:

- `DBParameterGroupName`
- `DBParameterGroupFamily`
- `Description`

Creating a DB Cluster Parameter Group

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

Console

To create a DB cluster parameter group

1. Sign in to the AWS Management Console and open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. In the navigation pane, choose **Parameter groups**.
3. Choose **Create parameter group**.
The **Create parameter group** window appears.
4. In the **Parameter group family** list, select a DB parameter group family
5. In the **Type** list, select **DB Cluster Parameter Group**.
6. In the **Group name** box, enter the name of the new DB cluster parameter group.
7. In the **Description** box, enter a description for the new DB cluster parameter group.
8. Choose **Create**.

AWS CLI

To create a DB cluster parameter group, use the AWS CLI [create-db-cluster-parameter-group](#) command. The following example creates a DB cluster parameter group named `mydbclusterparametergroup` for MySQL version 5.6 with a description of "My new cluster parameter group."

Include the following required parameters:

- `--db-cluster-parameter-group-name`
- `--db-parameter-group-family`
- `--description`

To list all of the available parameter group families, use the following command:

```
aws rds describe-db-engine-versions --query "DBEngineVersions[].[DBParameterGroupFamily"
```

Note

The output contains duplicates.

Example

For Linux, OS X, or Unix:

```
aws rds create-db-cluster-parameter-group \
--db-cluster-parameter-group-name mydbclusterparametergroup \
--db-parameter-group-family aurora5.6 \
--description "My new cluster parameter group"
```

For Windows:

```
aws rds create-db-cluster-parameter-group ^
--db-cluster-parameter-group-name mydbclusterparametergroup ^
--db-parameter-group-family aurora5.6 ^
--description "My new cluster parameter group"
```

This command produces output similar to the following:

```
DBCLUSTERPARAMETERGROUP  mydbclusterparametergroup  mysql5.6  My cluster new parameter
group
```

RDS API

To create a DB cluster parameter group, use the RDS API [CreateDBClusterParameterGroup](#) action.

Include the following required parameters:

- `DBClusterParameterGroupName`
- `DBParameterGroupFamily`
- `Description`

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.

If you change a parameter value, when the change is applied is determined by the type of parameter. Changes to dynamic parameters are applied immediately. Changes to static parameters require that

the DB instance associated with DB parameter group be rebooted before the change takes effect. To determine the type of a parameter, list the parameters in a parameter group using one of the procedures shown in the section [Listing DB Parameter Groups \(p. 300\)](#).

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.

Connectivity Monitoring Logs & events Configuration

Instance

Configuration

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
[REDACTED]

Resource id
[REDACTED]

Created time
Fri Nov 30 2018 15:41:20 GMT-0800 (Pacific Standard Time)

Parameter group
testsqtnet (pending-reboot)

Deletion protection
Disabled

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 parameters:

- `--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, OS X, 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` command with the following required parameters:

- `DBParameterGroupName`
- `Parameters`

Modifying Parameters in a DB Cluster Parameter Group

You can modify parameter values in a customer-created DB cluster parameter group; you can't change the parameter values in a default DB cluster parameter group. Changes to parameters in a customer-created DB cluster parameter group are applied to all DB clusters that are associated with the DB cluster parameter group.

If you change a parameter value, when the change is applied is determined by the type of parameter. Changes to dynamic parameters are applied immediately. Changes to static parameters require that the DB instances associated with DB cluster that uses the DB cluster parameter group be rebooted before the change takes effect. To determine the type of a parameter, list the parameters in a parameter group using one of the procedures shown in the section [Listing DB Parameter Groups \(p. 300\)](#).

The RDS console shows the status of the DB cluster parameter group associated with a DB instance. For example, if the DB instance is not using the latest changes to the associated DB cluster parameter group, the RDS console shows the DB cluster parameter group with a status of **pending-reboot**. You would need to manually reboot the DB instance for the latest parameter changes to take effect for that DB instance.

Details

Configurations

ARN
arn:aws:rds:us-west-2:XXXXXXXXXX:db:cluster5

Engine
Aurora MySQL 5.6.10a

Created Time
Tue Aug 07 08:47:41 GMT-700 2018

DB Name
cluster5

Username
myadmin

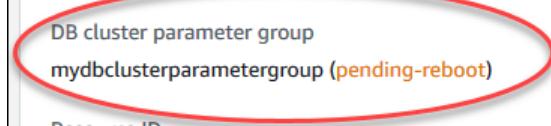
Option Group
default:aurora-5-6

Parameter group
[default.aurora5.6 \(in-sync\)](#)

DB cluster parameter group
[mydbclusterparametergroup \(pending-reboot\)](#)

Resource ID
XXXXXXXXXXXXXX

IAM DB Authentication Enabled
No



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**.

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, OS X, 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`

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 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.

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 parameters:

- `--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, OS X, or Unix:

```
aws rds copy-db-parameter-group \
--source-db-parameter-group-identifier mygroup1 \
--target-db-parameter-group-identifier mygroup2 \
--target-db-parameter-group-description "DB parameter group 2"
```

For Windows:

```
aws rds copy-db-parameter-group ^
--source-db-parameter-group-identifier mygroup1 ^
--target-db-parameter-group-identifier mygroup2 ^
--target-db-parameter-group-description "DB parameter group 2"
```

RDS API

To copy a DB parameter group, use the RDS API `CopyDBParameterGroup` operation with the following required parameters:

- `SourceDBParameterGroupIdentifier`
- `TargetDBParameterGroupIdentifier`

- TargetDBParameterGroupDescription

Copying a DB Cluster Parameter Group

You can copy custom DB cluster parameter groups that you create. Copying a parameter group is a convenient solution when you have already created a DB cluster parameter group and you want to include most of the custom parameters and values from that group in a new DB cluster parameter group. You can copy a DB cluster parameter group by using the AWS CLI [copy-db-cluster-parameter-group](#) command or the RDS API [CopyDBClusterParameterGroup](#) operation.

After you copy a DB cluster parameter group, wait at least 5 minutes before creating your first DB cluster that uses that DB cluster parameter group as the default parameter group. Doing this allows Amazon RDS to fully complete the copy action before the parameter group is used as the default for a new DB cluster. You can use the **Parameter Groups** option of the [Amazon RDS console](#) or the [describe-db-cluster-parameters](#) command to verify that your DB cluster parameter group is created.

Note

You can't copy a default parameter group. However, you can create a new parameter group that is based on a default parameter group.

Console

To copy a DB cluster parameter group

1. Sign in to the AWS Management Console and open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. In the navigation pane, choose **Parameter groups**.
3. In the list, choose the custom parameter group that you want to copy.
4. For **Parameter group actions**, choose **Copy**.
5. In **New DB parameter group identifier**, enter a name for the new parameter group.
6. In **Description**, enter a description for the new parameter group.
7. Choose **Copy**.

AWS CLI

To copy a DB cluster parameter group, use the AWS CLI [copy-db-cluster-parameter-group](#) command with the following required parameters:

- `--source-db-cluster-parameter-group-identifier`
- `--target-db-cluster-parameter-group-identifier`
- `--target-db-cluster-parameter-group-description`

The following example creates a new DB cluster parameter group named `mygroup2` that is a copy of the DB cluster parameter group `mygroup1`.

Example

For Linux, OS X, or Unix:

```
aws rds copy-db-cluster-parameter-group \
--source-db-cluster-parameter-group-identifier mygroup1 \
--target-db-cluster-parameter-group-identifier mygroup2 \
--target-db-cluster-parameter-group-description "DB parameter group 2"
```

For Windows:

```
aws rds copy-db-cluster-parameter-group ^
--source-db-cluster-parameter-group-identifier mygroup1 ^
--target-db-cluster-parameter-group-identifier mygroup2 ^
--target-db-cluster-parameter-group-description "DB parameter group 2"
```

RDS API

To copy a DB cluster parameter group, use the RDS API [CopyDBClusterParameterGroup](#) operation with the following required parameters:

- `SourceDBClusterParameterGroupIdentifier`
- `TargetDBClusterParameterGroupIdentifier`
- `TargetDBClusterParameterGroupDescription`

Listing DB Parameter Groups

You can list the DB parameter groups you've created for your AWS account.

Note

Default parameter groups are automatically created from a default parameter template when you create a DB instance for a particular DB engine and version. These default parameter groups contain preferred parameter settings and can't be modified. When you create a custom parameter group, you can modify parameter settings.

Console

To list all DB parameter groups for an AWS account

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

The DB parameter groups appear in a list.

AWS CLI

To list all DB parameter groups for an AWS account, use the AWS CLI [describe-db-parameter-groups](#) command.

Example

The following example lists all available DB parameter groups for an AWS account.

```
aws rds describe-db-parameter-groups
```

The command returns a response like the following:

DBPARAMETERGROUP	default.mysql5.5	mysql5.5	Default parameter group for MySQL5.5
DBPARAMETERGROUP	default.mysql5.6	mysql5.6	Default parameter group for MySQL5.6
DBPARAMETERGROUP	mydbparametergroup	mysql5.6	My new parameter group

The following example describes the *mydbparamgroup1* parameter group.

For Linux, OS X, or Unix:

```
aws rds describe-db-parameter-groups \
--db-parameter-group-name mydbparamgroup1
```

For Windows:

```
aws rds describe-db-parameter-groups ^
--db-parameter-group-name mydbparamgroup1
```

The command returns a response like the following:

```
DBPARAMETERGROUP mydbparametergroup1 mysql5.5 My new parameter group
```

RDS API

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

Listing DB Cluster Parameter Groups

You can list the DB cluster parameter groups you've created for your AWS account.

Note

Default parameter groups are automatically created from a default parameter template when you create a DB cluster for a particular DB engine and version. These default parameter groups contain preferred parameter settings and can't be modified. When you create a custom parameter group, you can modify parameter settings.

Console

To list all DB cluster parameter groups for an AWS account

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

The DB cluster parameter groups appear in the list with **DB cluster parameter group** for Type.

AWS CLI

To list all DB cluster parameter groups for an AWS account, use the AWS CLI [describe-db-cluster-parameter-groups](#) command.

Example

The following example lists all available DB cluster parameter groups for an AWS account.

```
aws rds describe-db-cluster-parameter-groups
```

The command returns a response like the following:

```
DBCLUSTERPARAMETERGROUPS      arn:aws:rds:us-west-2:1234567890:cluster-
pg:default.aurora5.6 default.aurora5.6      aurora5.6      Default cluster parameter
group for aurora5.6
DBCLUSTERPARAMETERGROUPS      arn:aws:rds:us-west-2:1234567890:cluster-
pg:mydbclusterparametergroup mydbclusterparametergroup      aurora5.6      My new cluster
parameter group
```

The following example describes the *mydbclusterparametergroup* parameter group.

For Linux, OS X, or Unix:

```
aws rds describe-db-cluster-parameter-groups \
--db-cluster-parameter-group-name mydbclusterparametergroup
```

For Windows:

```
aws rds describe-db-cluster-parameter-groups ^
--db-cluster-parameter-group-name mydbclusterparametergroup
```

The command returns a response like the following:

```
DBCLUSTERPARAMETERGROUPS      arn:aws:rds:us-west-2:1234567890:cluster-
pg:mydbclusterparametergroup mydbclusterparametergroup      aurora5.6      My new cluster
parameter group
```

RDS API

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

Viewing Parameter Values for a DB Parameter Group

You can get a list of all parameters in a DB parameter group and their values.

Console

To view the parameter values for a DB parameter group

1. Sign in to the AWS Management Console and open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. In the navigation pane, choose **Parameter groups**.
The DB parameter groups appear in a list.
3. Choose the name of the parameter group to see its list of parameters.

AWS CLI

To view the parameter values for a DB parameter group, use the AWS CLI [describe-db-parameters](#) command with the following required parameter.

- `--db-parameter-group-name`

Example

The following example lists the parameters and parameter values for a DB parameter group named *mydbparametergroup*.

```
aws rds describe-db-parameters --db-parameter-group-name mydbparametergroup
```

The command returns a response like the following:

DBPARAMETER	Parameter Name	Parameter Value	Source	Data Type	Apply
	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`

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": [
        {
            "ApplyMethod": "pending-reboot",
            "Description": "Controls whether user-defined functions that have only an xxx symbol for the main function can be loaded",
            "DataType": "boolean",
            "AllowedValues": "0,1",
            "SupportedEngineModes": [
                "provisioned"
            ],
            "Source": "engine-default",
            "IsModifiable": false,
            "ParameterName": "allow-suspicious-udfs",
            "ApplyType": "static"
        },
        {
            "ApplyMethod": "pending-reboot",
            "Description": "Enables new features in the Aurora engine.",
            "DataType": "boolean",
            "IsModifiable": true,
            "AllowedValues": "0,1",
            "SupportedEngineModes": [
                "provisioned"
            ],
            "Source": "engine-default",
            "ParameterValue": "0",
            "ParameterName": "aurora_lab_mode",
            "ApplyType": "static"
        },
        ...
    ]
}
```

The following example lists the parameters and parameter values for a DB cluster parameter group named *mydbclusterparametergroup*, in plain text format.

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

The command returns a response like the following:

```
PARAMETERS 0,1 pending-reboot static boolean Controls whether user-defined functions that have only an xxx symbol for the main function can be loaded False allow-suspicious-udfs engine-default SUPPORTEDENGINEMODES provisioned
PARAMETERS 0,1 pending-reboot static boolean Enables new features in the Aurora engine. True aurora_lab_mode 0 engine-default SUPPORTEDENGINEMODES provisioned
...
```

RDS API

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

- `DBClusterParameterGroupName`

Comparing Parameter Groups

You can use the AWS Management Console to view the differences between two parameter groups for the same DB engine and version.

To compare two parameter groups

1. Sign in to the AWS Management Console and open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. In the navigation pane, choose **Parameter groups**.
3. In the list, choose the two parameter groups that you want to compare.
4. For **Parameter group actions**, choose **Compare**.

Note

If the items you selected aren't equivalent, you can't choose **Compare**. For example, you can't compare a MySQL 5.6 and a MySQL 5.7 parameter group. You can't compare a DB parameter group and an Aurora DB cluster parameter group.

DB Parameter Values

You can specify the value for a DB parameter as any of the following:

- An integer constant
- A DB parameter formula
- A DB parameter function
- A character string constant
- A log expression (the log function represents log base 2), such as
`value={log(DBInstanceClassMemory/8187281418)*1000}`

DB Parameter Formulas

A DB parameter formula is an expression that resolves to an integer value or a Boolean value, and is enclosed in braces: {}. You can specify formulas 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 integer or a Boolean value. The names of the variables are case-sensitive.

AllocatedStorage

Returns the size, in bytes, of the data volume.

DBInstanceClassMemory

Returns the number of bytes of memory allocated to the DB instance class associated with the current DB instance, less the memory used by the Amazon RDS processes that manage the instance.

EndPointPort

Returns the number of 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

The parameter arguments can be specified as either integers or formulas. Each function must have at least one argument. Multiple arguments can be specified as a comma-separated list. The list can't have any empty members, such as *argument1,,argument3*. Function names are case-insensitive.

Note

DB Parameter functions are not currently supported in the AWS CLI.

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 Value Examples

These examples show using formulas and functions in the values for DB parameters.

Warning

Improperly setting parameters in a DB parameter group can have unintended adverse effects, including degraded performance and system instability. Always exercise caution when modifying database parameters and back up your data before modifying your DB parameter group. Try out parameter group changes on a test DB instances, created using point-in-time-restores, before applying those parameter group changes to your production DB instances.

You can specify the `LEAST()` function in an Aurora MySQL `table_definition_cache` parameter value 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)
```

Managing Connections with Amazon RDS Proxy (Preview)

This is preview documentation for Amazon RDS Proxy. It is subject to change.

Amazon RDS Proxy allows applications to pool and share database connections to improve scalability. It makes applications more resilient to database failures by automatically connecting to a standby DB instance while preserving application connections. RDS Proxy also allows you to enforce AWS IAM (Identity and Access Management) authentication to databases, and securely store credentials in Secrets Manager. RDS Proxy is fully compatible with MySQL and can be enabled for most applications with no code change.

You can handle unpredictable surges in database traffic, which otherwise could 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 sequences 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 (sheds load) while maintaining predictable performance for the load that can be served with 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.

Important

Currently, Amazon RDS Proxy is in preview. Don't use RDS Proxy for production workloads. Also, we strongly recommend not putting sensitive data in databases that you use with RDS Proxy preview. You might encounter issues during the preview that cause incorrect results, corrupted data, or both. Over the duration of the preview, we might introduce breaking changes without advance notice. These might include upgrades and changes to our APIs.

Topics

- [RDS Proxy Concepts and Terminology \(p. 308\)](#)
- [Planning for and Setting Up RDS Proxy \(p. 311\)](#)
- [Connecting to a Database through RDS Proxy \(p. 323\)](#)
- [Managing RDS Proxy \(p. 324\)](#)
- [Monitoring RDS Proxy \(p. 327\)](#)
- [Limitations for RDS Proxy \(p. 329\)](#)
- [Command-Line Examples for RDS Proxy \(p. 330\)](#)
- [Troubleshooting for RDS Proxy \(p. 331\)](#)

RDS Proxy Concepts and Terminology

You can simplify connection management for your Amazon RDS DB instances and 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 by understanding the database protocol and adjusting 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 independent of the database servers for the 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.

Topics

- [Overview of RDS Proxy Terminology \(p. 309\)](#)
- [Connection Pooling \(p. 310\)](#)
- [RDS Proxy Security \(p. 310\)](#)
- [Failover \(p. 310\)](#)
- [Transactions \(p. 311\)](#)

Overview of RDS Proxy Terminology

RDS Proxy handles the infrastructure to perform connection pooling and the other features described in this documentation. You see the proxies represented in the AWS Management Console for RDS on the [Proxies](#) page.

Each proxy handles connections to a single RDS DB instance or Aurora DB cluster. For RDS multi-AZ DB instances and Aurora provisioned clusters, the proxy determines the current writer instance.

The connections that a proxy keeps open and available for your database application to use form 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 the RDS Proxy can't be sure that it's safe to run different transactions from a session using different connections, it keeps the session on the same connection until the session ends. This fallback behavior is called *pinning*.

A proxy has an endpoint. You connect to this endpoint when you work with an RDS DB instance or Aurora DB cluster, instead of 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 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 referred to as the *targets* of that proxy. For convenience, when you create a proxy through the AWS Management 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. During the preview, RDS Proxy supports one engine family, consisting of RDS MySQL 5.6 and 5.7, and Aurora MySQL. Within the MySQL engine family, currently RDS Proxy supports Aurora provisioned clusters, Aurora parallel query clusters, and Aurora Global Databases. For a global database, you can create a proxy for the primary AWS Region, but not for the read-only secondary AWS Regions. RDS Proxy currently doesn't support Aurora Serverless clusters and Aurora multi-master clusters.

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 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 TLS 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 Transport Layer Security (TLS) and AWS Identity and Access Management (IAM). For general information about those security features, see [Security in Amazon Aurora \(p. 173\)](#). If you aren't familiar with how RDS and Aurora work with authentication, authorization, and other areas of security, consult those resources 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 only supports 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/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 still use native password authentication. These security features are a preferable alternative to embedding database credentials in your application code.

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 could also be part of normal maintenance procedures, such as while performing 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.

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.

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.

During DB failovers, RDS Proxy continues to accept connections at the same IP address and automatically directs connections to the new master. Clients connecting through RDS Proxy are not susceptible to DNS propagation delays on failover, local DNS caching, connection timeouts, uncertainty about which

DB instance is the current writer, or 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 cancelled. 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 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 `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.

If RDS Proxy detects a database request that makes it impractical to move your session to a different connection, 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 [Pinning \(p. 325\)](#).

Planning for and Setting Up RDS Proxy

This is preview documentation for Amazon RDS Proxy. It is subject to change.

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

- [Identifying DB Instances, Clusters, and Applications to Use with RDS Proxy \(p. 312\)](#)
- [Setting Up Network Prerequisites \(p. 312\)](#)
- [Setting Up Database Credentials in AWS Secrets Manager \(p. 312\)](#)
- [Setting Up AWS Identity and Access Management \(IAM\) Policies \(p. 313\)](#)
- [Creating an RDS Proxy \(p. 315\)](#)
- [Viewing RDS Proxy \(p. 317\)](#)
- [Deleting an RDS Proxy \(p. 319\)](#)
- [Modifying an RDS Proxy \(p. 319\)](#)

- [Adding a New Database User \(p. 323\)](#)
- [Changing the Password for a Database User \(p. 323\)](#)

Identifying DB Instances, Clusters, and Applications to Use with 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 ever 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 and 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 its limits for number of connections or 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 \(p. 327\)](#).
- AWS Lambda functions can also be good candidates for using with a proxy. These functions make frequent short database connections that benefit from connection pooling offered by RDS Proxy. You can take advantage of any IAM authentication you already have for Lambda functions, instead of managing database credentials in your Lambda application code.
- Applications that use languages and frameworks such as PHP and Ruby on Rails are typically good candidates for using with a proxy. Such applications typically open and close large numbers of database connections, and don't have built-in connection pooling mechanisms.
- Applications that keep a large number of connections open for long periods are typically good candidates for using with 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.
- 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.

Setting Up Network Prerequisites

Using RDS Proxy requires you to have a set of networking resources in place, such as a virtual private cloud (VPC), two or more subnets, EC2 instance within the same VPC, and Internet gateway. If you've successfully connected to any RDS DB instances or Aurora DB clusters, you already have the required network resources. If you are just getting started with RDS or Aurora, learn the basics of connecting to a database by following the procedures in [Setting Up Your Environment for Amazon Aurora \(p. 49\)](#). You can also follow the tutorial in [Getting Started with Amazon Aurora \(p. 54\)](#).

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 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**.

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. If you choose **Credentials for other database**, that option prompts you for the user name and password, but not the other connection details, which you specify in the settings for the proxy itself. If you choose **Other type of secrets**, you create the secret with keys named `username` and `password`.

Don't choose any rotation period for the secrets.

Because the secrets 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.

If a password associated with a secret is incorrect, you can update the associated secret in Secrets Manager. For example, you might update the secret if the password for the database account changes. Until you update the secret, you can't connect through the proxy to that database account. You can still connect to other accounts where the passwords match the credentials stored in the secrets.

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. For instructions about creating secrets in Secrets Manager, see [Creating a Secret](#) in the *AWS Secrets Manager User Guide*. 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 --description 'db admin user' \
  --secret-string '{"username":"admin","password":"choose_your_own_password"}'

aws secretsmanager create-secret \
  --name $PROXY_SECRET --description 'application user' \
  --secret-string '{"username":"app-user","password":"choose_your_own_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.

- Follow the **Create Role** process. Include the **Add Role to Database** step.
- For the new role, perform the **Add inline policy** step. Paste the following JSON, substituting the ARN for the secret that you created.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor0",
            "Effect": "Allow",
            "Action": [
                "secretsmanager:GetRandomPassword",
                "secretsmanager>CreateSecret",
                "secretsmanager>ListSecrets"
            ],
            "Resource": "*"
        },
        {
            "Sid": "VisualEditor1",
            "Effect": "Allow",
            "Action": "secretsmanager:*",
            "Resource": [
                "your_secret_ARN"
            ]
        }
    ]
}
```

- Optionally, edit the trust policy for this IAM policy. Paste the following JSON.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "rds.amazonaws.com"
            },
            "Action": "sts:AssumeRole"
        },
        {
            "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
    '{"Version":"2012-10-17","Statement":[{"Sid":"getsecretvalue","Effect":"Allow","Action":\
    ["secretsmanager:GetSecretValue","kms:Decrypt"],"Resource":"*"}]}'

aws kms create-key --description "$PREFIX-test-key" --policy "{\"Id\":\"$PREFIX-kms-policy\
\", \"Version\":\"2012-10-17\", \"Statement\":[{\\"Sid\\":\\"Enable IAM User Permissions\\",\
\\\"Effect\\\":\\\"Allow\\\", \\"Principal\\\":{\\\"AWS\\\":\\\"arn:aws:iam::account_id:root\\\"}, \\"Action\\\":\
\\\"kms:*\\\", \\"Resource\\\":\\\"*\\\"},{\\\"Sid\\\":\\\"Allow access for Key Administrators\\\", \\"Effect\\\":\
\\\"Allow\\\", \\"Principal\\\":{\\\"AWS\\\":[$USER_ARN]}, \\"arn:aws:iam::account_id:role/\
Admin\\\"}], \\"Action\\\":{[\\\"kms:Create*\\\", \\"kms:Describe*\\\", \\"kms:Enable*\\\", \\"kms>List*\
\\\", \\"kms:Put*\\\", \\"kms:Update*\\\", \\"kms:Revoke*\\\", \\"kms:Disable*\\\", \\"kms:Get*\\\",\
\\\"kms>Delete*\\\", \\"kms:TagResource\\\", \\"kms:UntagResource\\\", \\"kms:ScheduleKeyDeletion\
\\\", \\"kms:CancelKeyDeletion\\\"], \\"Resource\\\":\\\"*\\\"},{\\\"Sid\\\":\\\"Allow use of the key\\\",\
\\\"Effect\\\":\\\"Allow\\\", \\"Principal\\\":{\\\"AWS\\\":\\\"$ROLE_ARN\\\"}, \\"Action\\\":{\\\"kms:Decrypt\\\",\
\\\"kms:DescribeKey\\\"}, \\"Resource\\\":\\\"*\\\"}}]}"

```

Creating an RDS Proxy

To manage connections for a specified set of DB instances, you can create a proxy. You can associate a proxy with an RDS MySQL DB instance or a Aurora MySQL 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 the public preview, some choices are restricted or required. Consider the following guidance:

Proxy configuration:

- **Proxy identifier.** Specify a name of your choosing, unique within your AWS account ID and current AWS Region.
- **Engine compatibility.** Currently, **MySQL** is the only choice.
- **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. 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.

Target group configuration:

- **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. Then try creating the proxy again.
- **Connection pool maximum connections.** Specify a value between 1 and 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 it to 100. For details about how RDS Proxy uses this setting, see [Connection Limits and Timeouts \(p. 324\)](#).

- **Session pinning filters.** This is an advanced setting, for troubleshooting performance issues with particular applications. Currently, the only choice is EXCLUDE_VARIABLE_SETS. Only choose a filter if your application isn't reusing connections due to certain kinds of SQL statements, and if you can verify that reusing connections with those SQL statements doesn't affect application correctness.
- **Connection borrow timeout.** If you expect the proxy to sometimes use up all available connections, you can specify how long the proxy waits for a connection to become available before returning a timeout error. This setting only applies when the proxy has the maximum number of connections open and all connections are already in use.

Connectivity:

- **Secrets Manager ARNs.** Choose at least one Secrets Manager secret associated with 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 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.
- **Subnets.** This field is prepopulated with all the subnets associated with your VPC. Remove any subnets that you don't need for this proxy. You must leave at least two subnets.

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.

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.
5. Select **RDS Proxy is now available in Preview** to acknowledge the terms and conditions for the preview.
 6. Choose **Create Proxy**.

AWS CLI

To create a DB instance, use the AWS CLI command [create-db-proxy](#).

Example

For Linux, OS X, or Unix:

```
aws rds create-db-proxy \
    --db-proxy-name proxy_name \
    --role-arn iam_role \
```

```
--engine-family { MYSQL } \
--vpc-subnet-ids space_separated_list \
[--vpc-security-group-ids space_separated_list] \
[--auth ProxyAuthenticationConfig_JSON_string] \
[--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 ^
--role-arn iam_role ^
--engine-family { MYSQL } ^
--vpc-subnet-ids space_separated_list ^
[--vpc-security-group-ids space_separated_list] ^
[--auth ProxyAuthenticationConfig_JSON_string] ^
[--require-tls | --no-require-tls] ^
[--idle-client-timeout value] ^
[--debug-logging | --no-debug-logging] ^
[--tags comma_separated_list]
```

To create the required information and associations for the proxy, you also use these commands:

```
aws rds create-db-proxy-target-group
--name target_group_name
--db-proxy-name proxy_name
--connection-pool-defaults ConnectionPoolConfiguration
--tags comma_separated_list
```

```
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, you call the Amazon RDS API function [CreateDBProxy](#). You pass a parameter with the [AuthConfig](#) data structure.

You also create the associated target group by calling the function [CreateDBProxyTargetGroup](#). Then you associate an RDS DB instance or Aurora DB cluster with the target group by calling the function [RegisterDBProxyTargets](#).

Viewing 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.

In particular, you need the endpoint for the proxy to use in the connection string for any database applications that use the proxy.

AWS Management Console

To view your RDS Proxy

1. Sign in to the AWS Management Console and open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. In the upper-right corner of the AWS Management Console, choose the AWS Region in which you created the RDS Proxy.
3. In the navigation pane, choose **Proxies**.
4. Choose the name of an RDS proxy to display its details.
5. On the details page, the **Target groups** section shows how the proxy is associated with a specific RDS DB instance or Aurora DB cluster. You can follow the link to the **default** target group page to see more details about the association between the proxy and the database. This page is where you see settings that you specified when creating the proxy, such as maximum connection percentage, connection borrow timeout, engine compatibility, and session pinning filters.

CLI

To view your RDS Proxy using the CLI, use the `describe-db-proxies` command. By default, it displays all proxies owned by your AWS account. You can specify `--db-proxy-name` parameter to see details for a single proxy.

```
aws rds describe-db-proxies
[--db-proxy-name proxy_name]
```

To view the other information associated with the proxy, use these 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:

- Run `describe-db-proxies` to get a list of proxies.
- Run `describe-db-proxy-target-groups --db-proxy-name` using the name of the proxy as the parameter. This output shows connection parameters such as the maximum percentage of connections that the proxy can use.
- Run `describe-db-proxy-targets` to see the details of the RDS DB instance or Aurora DB cluster associated with the returned target group.

RDS API

To view your proxies using the RDS API, use the [DescribeDBProxies](#) operation. It returns values of the [DBProxy](#) data type.

You can use the proxy identifiers from the return value to see details of the connection settings for the proxy by using the [DescribeDBProxyTargetGroups](#) operation. It returns values of the [DBProxyTargetGroup](#) data type.

You can see the RDS instance or Aurora DB cluster associated with the proxy by using the [DescribeDBProxyTargets](#) operation. It returns values of the [DBProxyTarget](#) data type.

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`.

Example

```
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`.

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 rename the proxy by entering a new identifier.
 - You can turn the requirement for Transport layer Security (TLS) on or off.
 - You can enter a time period for the idle connection timeout.
 - You can add or remove Secrets Manager secrets. These secrets correspond to database user names and passwords.
 - You can change the IAM role used to retrieve the secrets from Secrets Manager.
 - You can require or disallow IAM authentication for connections to the proxy.
 - You can add or remove VPC subnets for the proxy to use.
 - You can add or remove VPC security groups for the proxy to use.
 - You can enable or disable enhanced logging.
6. Choose **Modify**.

If you didn't find the settings you intended to change, continue with the following procedure to update the target group for the proxy. Each proxy has one associated target group, which is created automatically along with the proxy.

To modify the settings for a proxy target group

The target group associated with a proxy controls the settings related to the physical database connections. You can only modify the target group from the proxy details page, not from the list on the **Proxies** page.

1. From 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. Certain properties, such as the target group identifier and the database engine, are fixed. You can't change these properties.

Choose new settings for the properties you can modify:

- You can choose a different RDS DB instance or Aurora cluster.
- You can adjust what percentage of the maximum available connections the proxy can use.
- You can select a session pinning filter. This setting 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.
- You can adjust the connection borrow timeout interval. This setting determines how long the proxy waits for a connection to become available before returning a timeout error, when the proxy already has the maximum number of connections being used.

5. Choose **Modify target group**.

AWS CLI

To modify a proxy using the AWS CLI, you 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 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, and 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
```

With the `modify-db-proxy-target-group` command, 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.

The following example shows how to first check the `MaxConnectionsPercent` 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 cluster named `cluster-56-2019-11-14-1399`. 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.

```
$ 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-56-2019-11-14-1399"
        },
        {
            "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-56-2019-11-14-1399

$ 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"
        }
    ]
}
```

```
        "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 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, and 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 \(p. 312\)](#).
- 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.

Connecting to a Database through RDS Proxy

This is preview documentation for Amazon RDS Proxy. It is subject to change.

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, all proxy connections have read-write capability and use the writer instance. If you use the reader endpoint for read-only connections, you continue using the reader endpoint the same way.

Connecting to a Proxy Using Native Authentication

Use the following general procedure to connect to a proxy using native authentication:

- 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"
    }
  ]
]
```

- Specify that endpoint as the host parameter in the connection string for your client application. For example, specify the proxy endpoint as the parameter for the `mysql -h` option.
- Supply the same database user name and password as you normally do.

Connecting to a Proxy Using IAM Authentication

Follow the same general procedure as for connecting to an RDS DB instance or Aurora cluster using IAM authentication. Instead of the instance, cluster, or reader endpoint, specify the proxy endpoint. For details of that procedure, see [Connecting to Your DB Instance from the Command Line: AWS CLI and mysql Client](#).

Managing RDS Proxy

The following explain aspects of RDS Proxy operation and configuration that help your application to make the most efficient use of database connections and achieve maximum connection reuse. The more you can take advantage of connection reuse, the more CPU and memory overhead 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.

Connection Limits and Timeouts

RDS Proxy makes use of the `max_connections` setting for your RDS DB instance or Aurora DB cluster. This setting represents the overall upper limit on the connections that the proxy can open at any one time. In Aurora clusters and RDS multi-AZ configurations, the `max_connections` value that the proxy uses is the one for the Aurora primary instance or the RDS writer instance.

The proxy setting for maximum connections represents a percentage of the `max_connections` value. If you have multiple applications all using the same database, you can effectively divide their connection quotas by using a proxy for each application with a specific percentage of `max_connections`. If you do so, ensure that the percentages add up to 100 or less for all proxies associated with the same database.

RDS Proxy periodically disconnects idle connections and returns them to the connection pool. You can adjust this timeout interval. Doing so helps your applications to deal with stale resources, especially if the application mistakenly leaves a connection open while holding important database resources.

Connection Pooling

As described in [Connection Pooling \(p. 310\)](#), connection pooling is a crucial RDS Proxy feature. Following, you can learn how to make the most efficient use of connection pooling and transaction-level connection reuse (multiplexing).

Because the connection pool is managed by RDS Proxy, you can monitor it and adjust connection limits and timeout intervals without changing your application code.

For each proxy, you can specify an upper limit on the number of connections used by the connection pool. You specify the limit as a percentage. This percentage applies to the maximum connections configured in the database. The exact number varies depending on the DB instance size and configuration settings. For example, suppose that you configured RDS Proxy to use 75% of the maximum connections for the database. For MySQL, the maximum value is defined by the `max_connections` configuration parameter. In this case, the other 25% of maximum connections would remain available to assign to other proxies or for connections that don't go through a proxy. If the database doesn't have many simultaneous connections, or some connections stay idle for long periods, the proxy might keep less than 75% of the maximum connections open at a particular time.

The overall number of connections available for the connection pool changes as you update the `max_connections` configuration setting that applies to an RDS DB instance or a Aurora MySQL cluster.

The proxy doesn't reserve all of these connections in advance. Thus, you can specify a relatively large percentage, and those connections are only opened when the proxy becomes busy enough to need them.

You can choose how long to wait for a connection to become available for use by your application. This setting is represented by the **Connection borrow timeout** option when you create a proxy. This setting specifies how long to wait for a connection to become available in the connection pool before returning a timeout error. It 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 writer instance is available because a failover operation is in process. This setting enables you to limit the wait period that is most suitable for your application without having to change the query timeout in your application code.

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. 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, you might modify your application code or workload to reduce the conditions that cause the pinning.

For example, if your application changes a session variable or configuration parameter, later statements can rely on the new variable or parameter to be in effect. Thus, when RDS Proxy processes requests to change session variables or configuration settings, it pins that session to the DB connection. That

way, the session state remains in effect for all later transactions in the same session. This rule doesn't apply to all parameters you can set. RDS Proxy tracks changes to the character set, collation, time zone, autocommit, current database, SQL mode, and `session_track_schema` settings. RDS Proxy doesn't pin the session when you modify them. In that case, RDS Proxy only reuses the connection for other sessions that have the same values for those settings.

Performance tuning for RDS Proxy involves trying to maximize transaction-level connection reuse (multiplexing) by minimizing pinning. You can do so by using the following techniques:

- Avoid unnecessary database requests that could 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.
- 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.
- Monitor the CloudWatch metric `DatabaseConnectionsCurrentlySessionPinned` to see how frequently pinning occurs. For information about this and other CloudWatch metrics, see [Monitoring RDS Proxy \(p. 327\)](#).
- If you use `SET` statements to perform identical initialization for each client connection, you can do so while still 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. You can work with the initialization query through the AWS CLI and RDS API. Currently, you can't set this property through the AWS Management Console.

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 4 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.
- An explicit `LOCK TABLE`, `LOCK TABLES`, or `FLUSH TABLES WITH READ LOCK` statement 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.
- 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 `ROW_COUNT()`, `FOUND_ROWS()`, or `LAST_INSERT_ID()` functions sometimes causes pinning and sometimes not. The behavior might be different 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, ensure 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 choose to skip the pinning behavior for certain application statements. You do so by choosing the **Session pinning filters** option when creating the proxy. Currently, you can opt out of session pinning for the following kinds of application statements:

- Setting session variables and configuration settings.

To see metrics about how often pinning occurs for a proxy, see [Monitoring RDS Proxy \(p. 327\)](#).

Monitoring RDS Proxy

This is preview documentation for Amazon RDS Proxy. It is subject to change.

You can monitor RDS Proxy 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**.

Note

RDS publishes these metrics for each underlying EC2 instance associated with the 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.

All RDS Proxy metrics are in the group `proxy`, with a dimension of `ProxyName`.

Metric	Valid Period	Description
<code>ClientConnectionsReceived</code>	1 minute and above	The number of client connection requests received. The most useful statistic for this metric is Sum.
<code>ClientConnectionsSetupSucceeded</code>	1 minute and above	The number of client connections successfully established with any authentication mechanism with or without TLS. The most useful statistic for this metric is Sum.
<code>ClientConnectionsSetupFailedAuth</code>	1 minute and above	The number of client connection attempts which failed due to bad authentication or TLS misconfiguration. The most useful statistic for this metric is Sum.
<code>ClientConnectionsClosed</code>	1 minute and above	The number of client connections closed. The most useful statistic for this metric is Sum.
<code>ClientConnections</code>	1 minute	The current number of client connections. This is reported every minute. The most useful statistic for this metric is Sum.

Metric	Valid Period	Description
QueryRequests	1 minute and above	The number of queries received. Note: Multi-statement query is counted as one query. The most useful statistic for this metric is Sum.
DatabaseConnectionRequests	1 minute and above	The number of requests to create a database connection. The most useful statistic for this metric is Sum.
DatabaseConnectionsSetupSucceeded	1 minute and above	The number of database connections successfully established with or without TLS. The most useful statistic for this metric is Sum.
DatabaseConnectionsSetupFailed	1 minute and above	The number of database connection requests which failed. The most useful statistic for this metric is Sum.
MaxDatabaseConnectionsAllowed	1 minute	The maximum number of database connections allowed. This is reported every minute. The most useful statistic for this metric is Sum.
DatabaseConnections	1 minute	The current number of database connections. This is reported every minute. The most useful statistic for this metric is Sum.
DatabaseConnectionsCurrentlyBorrowed	1 minute	The current number of database connections in the borrow state. This is reported every minute. The most useful statistic for this metric is Sum.

Metric	Valid Period	Description
DatabaseConnectionsCurrentlySessionPinned	1 minute	The current number of database connections currently pinned due to session state-altering operations in client requests. This is reported every minute. The most useful statistic for this metric is Sum.
DatabaseConnectionsCurrentlyInTransaction	1 minute	The current number of database connections in transaction. This is reported every minute. The most useful statistic for this metric is Sum.

Limitations for RDS Proxy

This is preview documentation for Amazon RDS Proxy. It is subject to change.

The following limitations apply to RDS Proxy during the public preview:

- The public preview is available only in these AWS Regions: US East (N. Virginia), US East (Ohio), US West (Oregon), Asia Pacific (Tokyo), and Europe (Ireland).
- You can have up to 20 proxies for each AWS account ID.
- In an Aurora cluster, all of the connections in the connection pool are handled by the Aurora primary instance. To perform load balancing for read-intensive workloads, you still use the reader endpoint directly for the Aurora cluster.
- Currently, RDS Proxy is only available for the MySQL engine family. This engine family includes RDS MySQL 5.6 and 5.7, and Aurora versions 1 and 2. Proxy support for PostgreSQL databases isn't available in the public preview.
- You can't use RDS Proxy with RDS MySQL 8.0.
- You can't use RDS Proxy with Aurora Serverless clusters.
- You can't use RDS Proxy with Aurora multi-master clusters.
- You can use RDS Proxy with Amazon RDS MySQL and Aurora MySQL. You can't use it with self-managed MySQL databases in EC2 instances.
- Currently, all proxies listen on port 3306.
- Your RDS Proxy must be in the same VPC as the database. Although the database can be publicly accessible, the proxy can't be.
- Currently, proxies don't track any changes to the set of DB instances within an Aurora DB cluster. Those changes include operations such as host replacements, instance renames, port changes, scaling instances up or down, or adding or removing DB instances.
- Not all logic is implemented to pin sessions to database connections based on SQL statements and functions. For the most current pinning behavior, see [Pinning \(p. 325\)](#).

Command-Line Examples for RDS Proxy

This is preview documentation for Amazon RDS Proxy. It is subject to change.

To see how combinations of mysql connection commands and SQL statements interact with RDS Proxy, look at the following examples.

Example Preserving Connections Across a Failover

This 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 the failover-db-cluster command is run 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-id 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-id 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

This is preview documentation for Amazon RDS Proxy. It is subject to change.

Common Issues and Solutions

You can refer to the following list of symptoms to see the possible causes and solutions to problems you might encounter using RDS Proxy. During the public preview, error handling code isn't complete, so you might experience cryptic error messages from underlying services.

You might encounter the following issues while creating a new proxy.

Error or Symptom	Causes or Workarounds
403: The security token included in the request is invalid	<ul style="list-style-type: none"> Select an existing IAM role instead of choosing to create a new one.

You might encounter the following issues while connecting to a proxy.

Error or Symptom	Causes or Workarounds
ERROR 2003 (HY000): Can't connect to MySQL server on ' proxy_endpoint ' (11)	<p>The proxy endpoint exists, but couldn't establish a connection to the RDS DB instance or Aurora DB cluster. This error typically occurs after a timeout period. Some possible reasons include the following:</p> <ul style="list-style-type: none"> You might have specified a nonexistent database user name. You might have entered an incorrect database user password. You might not have a Secrets Manager secret with the credentials of the specified database user. The credentials in the Secrets Manager secret might not match the credentials of the corresponding database user. You might not have permission to retrieve the corresponding Secrets Manager secret.
ERROR 2005 (HY000): Unknown MySQL server host ' proxy_endpoint ' (0)	<p>The proxy endpoint doesn't exist. This error typically occurs immediately. Some possible reasons include the following:</p> <ul style="list-style-type: none"> You might have deleted the proxy. You might have copied or typed the proxy endpoint incorrectly. You might have specified an incorrect port. Currently, all proxies listen on port 3306.
ERROR 2013 (HY000): Lost connection to MySQL server at 'reading initial communication packet', system error: 2	<p>The proxy couldn't log into the database using the credentials from the Secrets Manager secrets. Some possible reasons include the following:</p> <ul style="list-style-type: none"> The Secrets Manager secrets associated with the proxy don't include one with the user name you're connecting as. You can add a secret with correct credentials to your proxy if so. The Secrets Manager secret for the user name you're connecting as has the wrong password. You can update the secret with the incorrect credentials if so. The Secrets Manager secret for some other user, not the one you're connecting as, has an incorrect password. You can remove the secret with the incorrect credentials if so.

CloudWatch Logs

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 (especially during preview).

Verifying Connectivity to 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. Also examine the associated target groups and the targets specified in the 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.

```
aws rds describe-db-proxies --db-proxy-name $DB_PROXY_NAME
aws rds describe-db-proxy-target-groups --db-proxy-name $DB_PROXY_NAME
aws rds describe-db-proxy-targets --db-proxy-name $DB_PROXY_NAME
```

If the following nc command reports success, 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 proxy_endpoint 3306
```

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
}
```

Examine the RDS DB instance or Aurora DB cluster that you associated with the proxy. For an Aurora DB cluster, also examine each of the DB instances in the cluster. Make sure that their properties match what's expected for the proxy, such as the port being 3306.

```
aws rds describe-db-clusters --db-cluster-id $DB_CLUSTER_ID
aws rds describe-db-instances --db-instance-id $DB_INSTANCE_ID1
aws rds describe-db-instances --db-instance-id $DB_INSTANCE_ID2
```

Cloning Databases in an Aurora DB Cluster

Using database cloning, you can quickly and cost-effectively create clones of all of the databases within an Aurora DB cluster. The clone databases require only minimal additional space when first created.

Database cloning uses a *copy-on-write protocol*, in which data is copied at the time that data changes, either on the source databases or the clone databases. You can make multiple clones from the same DB cluster. You can also create additional clones from other clones. For more information on how the copy-on-write protocol works in the context of Aurora storage, see [Copy-on-Write Protocol for Database Cloning \(p. 335\)](#).

You can use database cloning in a variety of use cases, especially where you don't want to have an impact on your production environment. Some examples are the following:

- Experiment with and assess the impact of changes, such as schema changes or parameter group changes.
- Perform workload-intensive operations, such as exporting data or running analytical queries.
- Create a copy of a production DB cluster in a nonproduction environment for development or testing.

Topics

- [Limitations \(p. 335\)](#)
- [Copy-on-Write Protocol for Database Cloning \(p. 335\)](#)
- [Deleting Source Databases \(p. 337\)](#)
- [Cloning an Aurora Cluster Through the AWS Management Console \(p. 337\)](#)
- [Cloning an Aurora Cluster Through the AWS CLI \(p. 338\)](#)
- [Cross-Account Cloning \(p. 339\)](#)

Limitations

There are some limitations involved with database cloning, described following:

- You cannot create clone databases across AWS regions. The clone databases must be created in the same region as the source databases.
- Currently, you are limited to 15 clones based on a copy, including clones based on other clones. After that, only copies can be created. However, each copy can also have up to 15 clones.
- Cross-account database cloning is currently supported only for Aurora MySQL.
- Currently, you cannot clone from a cluster without the parallel query feature, to a cluster where parallel query is enabled. To bring data into a cluster that uses parallel query, create a snapshot of the original cluster and restore it to a cluster where the parallel query option is enabled.
- You can provide a different virtual private cloud (VPC) for your clone. However, the subnets in those VPCs must map to the same set of Availability Zones.

Copy-on-Write Protocol for Database Cloning

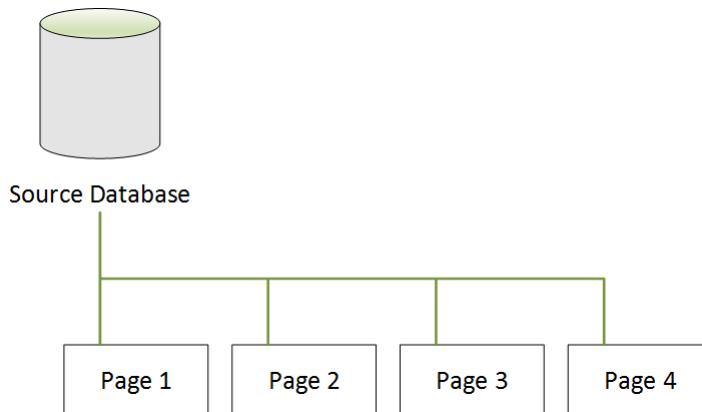
The following scenarios illustrate how the copy-on-write protocol works.

- [Before Database Cloning \(p. 336\)](#)
- [After Database Cloning \(p. 336\)](#)

- When a Change Occurs on the Source Database (p. 336)
- When a Change Occurs on the Clone Database (p. 337)

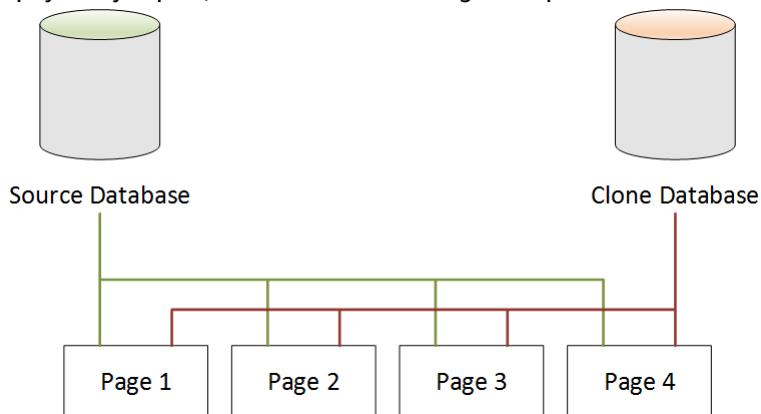
Before Database Cloning

Data in a source database is stored in pages. In the following diagram, the source database has four pages.



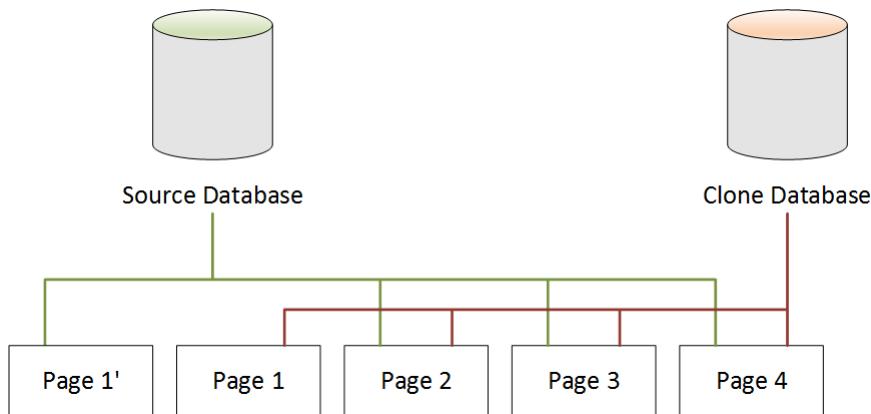
After Database Cloning

As shown in the following diagram, there are no changes in the source database after database cloning. Both the source database and the clone database point to the same four pages. None of the pages has been physically copied, so no additional storage is required.



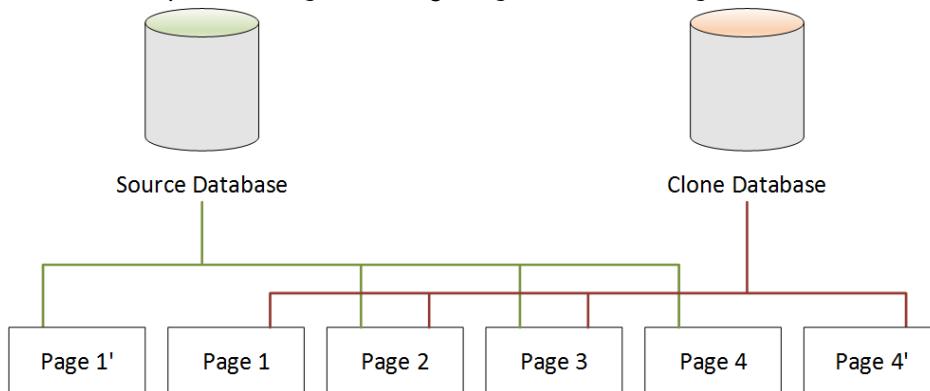
When a Change Occurs on the Source Database

In the following example, the source database makes a change to the data in Page 1. Instead of writing to the original Page 1, additional storage is used to create a new page, called Page 1'. The source database now points to the new Page 1', and also to Page 2, Page 3, and Page 4. The clone database continues to point to Page 1 through Page 4.



When a Change Occurs on the Clone Database

In the following diagram, the clone database has also made a change, this time in Page 4. Instead of writing to the original Page 4, additional storage is used to create a new page, called Page 4'. The source database continues to point to Page 1', and also Page 2 through Page 4, but the clone database now points to Page 1 through Page 3, and also Page 4'.



As shown in the second scenario, after database cloning there is no additional storage required at the point of clone creation. However, as changes occur in the source database and clone database, only the changed pages are created, as shown in the third and fourth scenarios. As more changes occur over time in both the source database and clone database, you need incrementally more storage to capture and store the changes.

Deleting Source Databases

When deleting a source database that has one or more clone databases associated with it, the clone databases are not affected. The clone databases continue to point to the pages that were previously owned by the source database.

Cloning an Aurora Cluster Through the AWS Management Console

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

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 \(p. 339\)](#) 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**. Choose the DB cluster that you want to create a clone of.
 3. For **Actions**, choose **Create clone**.
 4. On the **Create clone** page, type a name for the primary instance of the clone DB cluster as the **DB instance identifier**.
- If you want to, set any other settings for the clone DB cluster. For information about the different DB cluster settings, see [New Console \(p. 101\)](#).
5. Choose **Create clone** to launch the clone DB cluster.

Cloning an Aurora Cluster Through the AWS CLI

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

To create a clone of a DB cluster using the AWS CLI

- Call the [restore-db-cluster-to-point-in-time](#) AWS CLI command and supply the following values:
 - **--source-db-cluster-identifier** – the name of the source DB cluster to create a clone of.
 - **--db-cluster-identifier** – the name of the clone DB cluster.
 - **--restore-type copy-on-write** – values that indicate to create a clone DB cluster.
 - **--use-latest-restorable-time** – specifies that the latest restorable backup time is used.

The following example creates a clone of the DB cluster named `sample-source-cluster`. The name of the clone DB cluster is `sample-cluster-clone`.

For Linux, OS X, or Unix:

```
aws rds restore-db-cluster-to-point-in-time \
  --source-db-cluster-identifier sample-source-cluster \
  --db-cluster-identifier sample-cluster-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 sample-source-cluster ^
  --db-cluster-identifier sample-cluster-clone ^
  --restore-type copy-on-write ^
  --use-latest-restorable-time
```

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-instance-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.

Cross-Account Cloning

With Amazon Aurora, you can share an Aurora DB cluster with another AWS account or AWS organization. By sharing this way, you can clone the DB cluster and access the clone from the other account or organization.

For example, if you use separate accounts for production and testing, you can create a clone of production data in your test account. You can use this clone to experiment with different parameters and run expensive online analytical processing (OLAP) queries, all without having an impact on the production environment. You can give outside companies access to your database, for example for an external vendor to train a machine learning model. Cross-account cloning is much faster in such situations than creating and restoring a database snapshot.

To authorize sharing, you use AWS Resource Access Manager (AWS RAM). For more information about controlling access through AWS RAM, 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 owner modifies the cluster to allow one or more other accounts to clone it. If any of the accounts are in a different AWS organization, AWS generates a sharing invitation and 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).

You're only charged for additional storage space if you make data changes. If the source cluster is deleted, storage costs are distributed equally among remaining cloned clusters.

Topics

- [Limitations of Cross-Account Cloning \(p. 339\)](#)
- [Allowing Other AWS Accounts to Clone Your Cluster \(p. 340\)](#)
- [Cloning a Cluster Owned by Another AWS Account \(p. 342\)](#)

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 clone an Aurora global database cluster across AWS accounts.
- Viewing and accepting sharing invitations requires using the AWS CLI the Amazon RDS API, or the AWS RAM console. Currently, you can't perform this procedure using the Amazon RDS console.
- When you make a cross-account cluster, you can't make additional clones of that new cluster or share the cloned cluster with other AWS accounts.
- The maximum number of cross-account clones that you can have for any Aurora cluster is 15.
- Your cluster must be in ACTIVE state at the time that you share it with other AWS accounts.
- While an Aurora cluster is shared with other AWS accounts, you can't rename the cluster.
- You can't create a cross-account clone of a cluster that is encrypted with the default RDS key.
- When an encrypted cluster is shared with you, you must encrypt the cloned cluster. The key you use can be different from the encryption key for the original cluster. The cluster owner must also grant you permission to access the AWS Key Management Service (AWS KMS) key for the original cluster.

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 \(p. 340\)](#)
- [Checking If a Cluster You Own Is Shared with Other AWS Accounts \(p. 342\)](#)

Granting Permission to Other AWS Accounts to Clone Your Cluster

If the cluster that you're sharing is encrypted, you also share the customer master key (CMK) for the cluster. You can allow AWS Identity and Access Management (IAM) users or roles in one AWS account to use a CMK in a different account. To do this, you first add the external account (root user) to the CMK'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 CMK that you create, not the default RDS service key. For information about access control for CMKs, see [Authentication and Access Control for AWS KMS](#).

Console

To grant permission to clone your cluster using the AWS Management Console

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

1. Gather the information for the required parameters. You need the ARN for your cluster and the numeric ID for the other AWS account.

- Run the RAM CLI command [create-resource-share](#).

For Linux, OS X, 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`.

RAM API

To grant permission to clone your cluster using the RAM API

- Gather the information for the required parameters. You need the ARN for your cluster and the numeric ID for the other AWS account.
- Call the RAM API operation [CreateResourceShare](#), and specify the following values:
 - Specify the account IDs of one or more AWS accounts as the `principals` parameter.
 - Specify the ARNs of one or more Aurora DB clusters as the `resourceArns` parameter.
 - Specify whether the permitted account IDs can be outside your AWS organization or not by including a Boolean value for the `allowExternalPrincipals` parameter.

Recreating a Cluster that Uses Default RDS Key

To recreate an encrypted cluster that uses the default RDS key

If the encrypted cluster that you intend to share uses the default RDS key, you must recreate it using a customer master key (CMK) and share the new cluster instead. You do so by creating a manual snapshot of your DB cluster and restoring it to a new cluster. Use the following steps.

- Sign in to the AWS Management Console and open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
- Choose **Snapshots** from the navigation pane.
- Choose your snapshot.
- For **Actions**, choose **Copy Snapshot**, and then choose **Enable encryption**.
- For **Master key**, choose the new encryption key that you want to use.
- Restore the copied snapshot. To do so, follow the procedure in [Restoring from a DB Cluster Snapshot \(p. 385\)](#). 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 \(p. 407\)](#). Before you do, confirm that your new cluster has all necessary data and that your application can access it successfully.

Checking If a Cluster 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 you own is shared with other AWS accounts using the AWS CLI

- Call the 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
```

RAM API

To find out if a cluster you own is shared with other AWS accounts using the RAM API

- Call the 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 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, you 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 Owned by Other AWS Accounts \(p. 342\)](#)
- [Accepting Invitations to Share Clusters Owned by Other AWS Accounts \(p. 343\)](#)
- [Cloning an Aurora Cluster Owned by Another AWS Account \(p. 344\)](#)
- [Checking If a DB Cluster is a Cross-Account Clone \(p. 347\)](#)

Viewing Invitations to Clone Clusters 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 owned by other AWS accounts using the AWS CLI

1. Run the 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`]`.

RAM API

To see invitations to clone clusters owned by other AWS accounts using the RAM API

1. Call the 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 RAM and RDS APIs, or the 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 using the AWS CLI

1. Find the invitation ARN by running the RAM CLI command `get-resource-share-invitations`, as shown preceding.
2. Accept the invitation by calling the RAM CLI command `accept-resource-share-invitation`, as shown following.

For Linux, OS X, 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
```

RAM and RDS API

To accept invitations to share somebody's cluster using the RAM and RDS APIs

1. Find the invitation ARN by calling the 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 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 owned by another 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**.

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 [Cloning an Aurora Cluster Through the AWS Management Console \(p. 337\)](#) 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 key to encrypt the clone, or your own customer master key (CMK). You can't create a cross-account clone for a cluster that is encrypted with the default RDS key.

The account owning 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 using the AWS CLI

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, OS X, 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 customer master key (CMK). Your account must have permission to use that encryption key.

For Linux, OS X, 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 owning 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}}
}
]
}

```

RDS API

To clone an Aurora cluster owned by another AWS account using the RDS API

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 customer master key (CMK). Your account must have permission to use that encryption key.

When cloning 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 owning 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}}
        }
    ]
}

```

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 using the AWS CLI

- 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 using the RDS API

- 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 \(p. 350\)](#)
- [Integrating AWS Services with Amazon Aurora PostgreSQL \(p. 350\)](#)
- [Using Amazon Aurora Auto Scaling with Aurora Replicas \(p. 351\)](#)
- [Using Machine Learning \(ML\) Capabilities with Amazon Aurora \(p. 366\)](#)

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 \(p. 351\)](#).

For more information about integrating Aurora MySQL with other AWS services, see [Integrating Amazon Aurora MySQL with Other AWS Services \(p. 727\)](#).

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 \(p. 351\)](#).

For more information about integrating Aurora PostgreSQL with other AWS services, see [Integrating Amazon Aurora PostgreSQL with Other AWS Services \(p. 916\)](#).

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 \(p. 351\)](#)
- [Aurora Auto Scaling Policies \(p. 352\)](#)
- [Adding a Scaling Policy \(p. 353\)](#)
- [Editing a Scaling Policy \(p. 362\)](#)
- [Deleting a Scaling Policy \(p. 364\)](#)
- [Related Topics \(p. 365\)](#)

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 \(p. 100\)](#).

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 [Choosing the DB Instance Class \(p. 76\)](#). 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 \(p. 380\)](#).

Aurora Auto Scaling only removes Aurora Replicas that it created.

To benefit from Aurora Auto Scaling, your applications must support connections to new Aurora Replicas. To do so, we recommend using the Aurora reader endpoint. For Aurora MySQL you can use a driver such as the MariaDB Connector/J utility. For more information, see [Connecting to an Amazon Aurora DB Cluster \(p. 166\)](#).

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 API Reference](#).

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.

Topics

- [Adding a Scaling Policy Using the AWS Management Console \(p. 353\)](#)
- [Adding a Scaling Policy Using the AWS CLI or the Application Auto Scaling API \(p. 356\)](#)

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 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 `max-capacity`.
- `--max-capacity` – 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 `min-capacity`.

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, OS X, 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 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 **MaxCapacity**.
- **MaxCapacity** – 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 **MinCapacity**.

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 \(p. 358\)](#)
- [Using a Custom Metric \(p. 359\)](#)
- [Using Cooldown Periods \(p. 359\)](#)
- [Disabling Scale-in Activity \(p. 360\)](#)

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 Aurora Metrics \(p. 457\)](#).

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, OS X, 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 \(p. 360\)](#).

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, OS X, 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"
}
```

Related Topics

- [Integrating Aurora with Other AWS Services \(p. 350\)](#)
- [Managing an Amazon Aurora DB Cluster \(p. 260\)](#)

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. For instance, you can build product recommendation systems by writing SQL queries in Aurora. The queries pass customer profile, shopping history, and product catalog data to an Amazon 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

- [Overview of Aurora Machine Learning \(p. 366\)](#)
- [Prerequisites for Aurora Machine Learning \(p. 367\)](#)
- [Enabling Aurora Machine Learning \(p. 367\)](#)
- [Exporting Data to Amazon S3 for Amazon SageMaker Model Training \(p. 372\)](#)
- [Using Amazon SageMaker to Run Your Own ML Models \(p. 373\)](#)
- [Using Amazon Comprehend for Sentiment Detection \(p. 375\)](#)
- [Performance Considerations for Aurora Machine Learning \(p. 376\)](#)
- [Monitoring Aurora Machine Learning \(p. 377\)](#)
- [Limitations of Aurora Machine Learning \(p. 378\)](#)

Overview of Aurora Machine Learning

Aurora machine learning enables you to add machine learning-based predictions to database applications using the SQL language. Aurora machine learning makes use of a highly optimized integration between the Aurora database and the AWS machine learning (ML) services Amazon 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 Amazon 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 Sentiment Detection \(p. 375\)](#).

For general information about Amazon SageMaker, see [Amazon SageMaker](#). For details about using Aurora and Amazon SageMaker together, see [Using Amazon SageMaker to Run Your Own ML Models \(p. 373\)](#).

Prerequisites for Aurora Machine Learning

Currently, Aurora machine learning requires that the cluster use the Aurora MySQL database engine. This feature is available on any Aurora cluster running Aurora MySQL 2.07.0 and higher. You can upgrade an older Aurora cluster to one of these releases and use this feature with that cluster.

Enabling Aurora Machine Learning

Enabling the ML capabilities involves the following steps:

- You enable the Aurora cluster to access the Amazon machine learning services Amazon SageMaker or Amazon Comprehend, depending the kinds of ML algorithms you want for your application.
- For Amazon 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 Amazon SageMaker \(p. 367\)](#)
- [Granting SQL Privileges for Invoking Aurora Machine Learning Services \(p. 371\)](#)
- [Enabling Network Communication from Aurora MySQL to Other AWS Services \(p. 372\)](#)

Setting Up IAM Access to Amazon Comprehend and Amazon SageMaker

Before you can access Amazon SageMaker and Amazon Comprehend services, enable the Aurora MySQL cluster to access the 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, Amazon SageMaker, or Amazon Comprehend Using the Console \(p. 368\)](#).

Setting up the IAM roles for Amazon 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 Amazon 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 Amazon 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 \(p. 728\)](#).

[Connecting an Aurora DB Cluster to Amazon S3, Amazon SageMaker, or Amazon Comprehend Using the Console](#)

Aurora machine learning requires that your DB cluster use some combination of Amazon S3, Amazon SageMaker, and Amazon Comprehend. Amazon Comprehend is for sentiment analysis. Amazon SageMaker is for a wide variety of machine learning algorithms. For Aurora Machine Learning, Amazon S3 is only for training Amazon 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, Amazon 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.
5. Choose the service that you want to connect to from the dropdown list.
 - **Amazon S3**
 - **Amazon Comprehend**
 - **Amazon SageMaker**
6. Choose **Connect service**.
7. Enter the required information for the specific service on the **Connect cluster** window:
 - For Amazon SageMaker, enter the Amazon Resource Name (ARN) of an Amazon SageMaker endpoint. For details about what the endpoint represents, see [Deploy a Model on Amazon SageMaker Hosting Services](#).

In the navigation pane of the [Amazon SageMaker console](#), choose **Endpoints** and copy the ARN of the endpoint you want to use.

- For Amazon Comprehend, you 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 you use is set up with the requirements for training Amazon 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 Developer Guide. For more about using an Amazon S3 bucket with Amazon SageMaker, see [Step 1: Create an Amazon 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-policy-timestamp`.
 - The Amazon 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 Amazon 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 Amazon SageMaker function on your behalf. You can specify all of your Amazon 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 Amazon SageMaker endpoint. However, an Aurora MySQL cluster can only invoke Amazon 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 AWS 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 \(p. 734\)](#).

Creating an IAM Role to Access Amazon 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 \(p. 734\)](#). 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,method=pending-reboot" \
--parameters
"ParameterName=aws_default_sagemaker_role,ParameterValue=arn:aws:iam::123456789012:role/AllowAuroraSageMakerRole,method=pending-reboot" \
--parameters
"ParameterName=aws_default_comprehend_role,ParameterValue=arn:aws:iam::123456789012:role/AllowAuroraComprehendRole,method=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-name AllowAWSAccessToExternalServices
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 Amazon SageMaker and built-in functions for Amazon Comprehend.

The database user invoking a native function must be granted the `Invoke SAGEMAKER` or `Invoke COMPREHEND` privilege. To grant this privilege to a user, connect to the DB instance as the master user, and run the following statements. Substitute the appropriate details for the database user.

```
GRANT INVOKESAGEMAKER ON *.* TO user@domain-or-ip-address
GRANT INVOKECOMPREHEND ON *.* TO user@domain-or-ip-address
```

For Amazon 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 Amazon 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 Amazon 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 \(p. 738\)](#).

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 Amazon SageMaker Model Training

Depending on how your team divides the machine learning tasks, you might not perform this task. If someone else provides the Amazon SageMaker model for you, you can skip this section.

To train Amazon SageMaker models, you export data to an Amazon S3 bucket. The Amazon S3 bucket is used by a Jupyter Amazon 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 Amazon SageMaker to Run Your Own ML Models

Amazon SageMaker is a fully managed machine learning service. With Amazon 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. Amazon 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, Amazon SageMaker offers flexible distributed training options that adjust to your specific workflows.

Currently, Aurora Machine Learning supports any Amazon SageMaker endpoint that can read and write comma-separated value format, through a `ContentType` of `text/csv`. The built-in Amazon 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 Amazon 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 Amazon SageMaker, Amazon Comprehend, or Lambda for the global database cluster in each AWS Region.
- Ensure that all AWS Regions have the same trained Amazon 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 Amazon 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 Amazon SageMaker endpoint hosting the model. When you define such a function, you specify the input parameters to the model, the specific Amazon SageMaker endpoint to invoke, and the return type. The function returns the inference computed by the Amazon SageMaker endpoint after executing the model on 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''
    [MAX_BATCH_SIZE max_batch_size];      -- default is 10,000

```

This is a variation of the existing CREATE FUNCTION DDL statement. In the CREATE FUNCTION statement that defines the Amazon 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 Amazon SageMaker. An Amazon 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 Amazon 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 Amazon SageMaker features can't directly use the CSV exported with this option. The manifest format is not compatible with the expected manifest format from Amazon SageMaker.

You create a separate stored function for each of your Amazon 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 execute 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 Amazon SageMaker endpoint to detect anomalies. There is an Amazon 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 Amazon SageMaker Functions that Return Strings

We recommend specifying a character set of `utf8mb4` as the return type type for your Amazon 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 Amazon 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_confidence()` 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 \(p. 376\)](#).

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 to the executing query one row at a time. 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 Amazon SageMaker endpoint, you define the batch size parameter. This parameter influences how many rows are transferred for every underlying call to Amazon SageMaker. For queries that process large numbers of rows, the overhead to make a separate Amazon 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 Amazon SageMaker function, you can tell by checking the execution 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 Amazon 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          |           |       |             | NULL   | NULL    | NULL  |
+-----+-----+-----+-----+-----+-----+-----+
| 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 execution, 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 executed 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 executed 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 Amazon 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. See [the MySQL documentation](#) for information about generated columns.

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. See [the MySQL documentation](#) for information about this binlog format.

Backing Up and Restoring an Amazon Aurora DB Cluster

This section shows how to back up and restore Amazon Aurora DB clusters.

Topics

- [Overview of Backing Up and Restoring an Aurora DB Cluster \(p. 380\)](#)
- [Understanding Aurora Backup Storage Usage \(p. 382\)](#)
- [Creating a DB Cluster Snapshot \(p. 383\)](#)
- [Restoring from a DB Cluster Snapshot \(p. 385\)](#)
- [Copying a Snapshot \(p. 388\)](#)
- [Sharing a DB Cluster Snapshot \(p. 399\)](#)
- [Restoring a DB Cluster to a Specified Time \(p. 405\)](#)
- [Deleting a Snapshot \(p. 407\)](#)

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.

Fault Tolerance for an Aurora DB Cluster

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

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

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

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

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 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.

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 \(p. 684\)](#).

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.

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 cannot 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 \(p. 382\)](#). 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.

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 instance, look for the `Latest Restorable Time` or `Earliest Restorable Time` values on the RDS console. For information about viewing these values, see [Viewing an Amazon Aurora DB Cluster \(p. 446\)](#). 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 \(p. 405\)](#).

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 Databases in an Aurora DB Cluster \(p. 335\)](#).

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 \(p. 625\)](#).

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 \(p. 380\)](#). 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 [Overview of Monitoring Amazon RDS \(p. 434\)](#) and the table of metrics in [Amazon RDS Metrics \(p. 438\)](#).

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 your databases. Since 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.

Note

Amazon bills you based upon the amount of 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 \(p. 382\)](#). For pricing information about Aurora storage, see [Amazon RDS for Aurora Pricing](#).

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

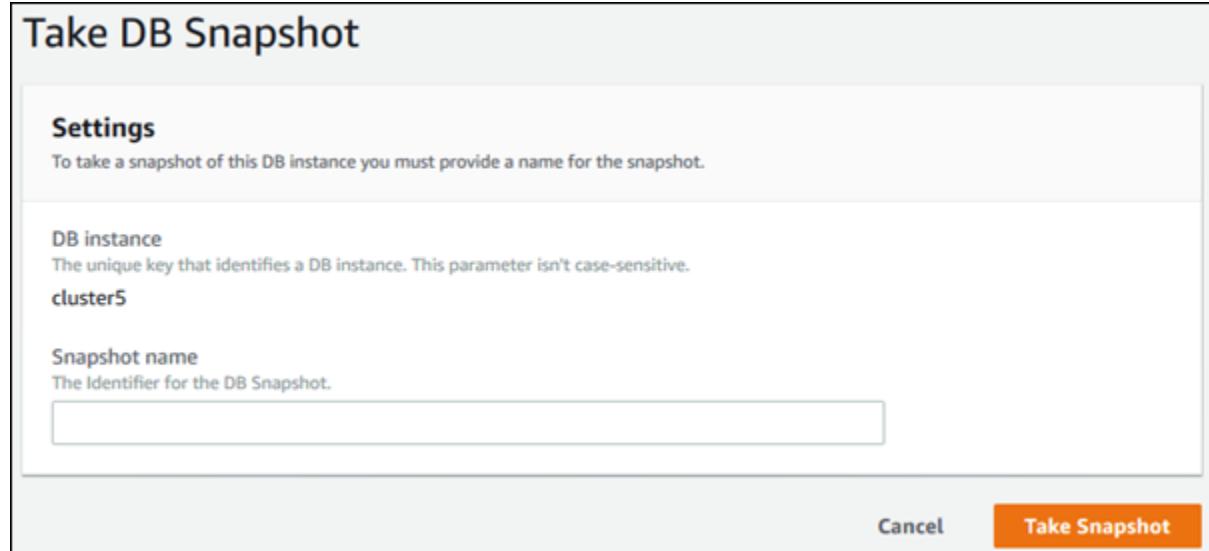
Console

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, OS X, 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`

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 DB cluster by restoring from this DB cluster snapshot. When you restore the DB cluster, 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.

Note

You can't restore a DB cluster from a DB cluster snapshot that is both shared and encrypted. Instead, you can make a copy of the DB cluster snapshot and restore the DB cluster from the copy.

Parameter Group Considerations

We recommend that you retain the parameter group for any DB cluster snapshots you create, so that you can associate your restored DB cluster with the correct parameter group. You can specify the parameter group when you restore the DB cluster.

Security Group Considerations

When you restore a DB cluster, the default security group is associated with the restored cluster by default.

Note

- If you're using the AWS CLI, you can specify a custom 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.
- The Amazon RDS console has no option for associating a custom security group while restoring.

As soon as the restore is complete and your new DB cluster is available, you can associate any custom security groups used by the snapshot you restored from. You must apply these changes by modifying the DB cluster with the RDS console, the AWS CLI `modify-db-cluster` command, or the `ModifyDBCluster` Amazon RDS API operation. For more information, see [Modifying an Amazon Aurora DB Cluster \(p. 264\)](#).

Amazon Aurora Considerations

With Aurora, you restore a DB cluster snapshot to a DB cluster.

With Aurora MySQL, you can also restore a DB cluster snapshot to an Aurora Serverless DB cluster. For more information, see [Restoring an Aurora Serverless DB Cluster \(p. 129\)](#).

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 \(p. 644\)](#).

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 DB Instance** page, for **DB Instance Identifier**, enter the name for your restored DB cluster.
6. Choose **Restore DB Instance**.
7. If you want to restore the functionality of the DB cluster to that of the DB cluster that the snapshot was created from, you must modify the DB cluster to use the security group. The next steps assume that your DB cluster is in a VPC. If your DB cluster is not in a VPC, use the EC2 Management Console to locate the security group you need for the DB cluster.
 - a. Sign in to the AWS Management Console and open the Amazon VPC console at <https://console.aws.amazon.com/vpc/>.
 - b. In the navigation pane, choose **Security Groups**.
 - c. Select the security group that you want to use for your DB clusters. If necessary, add rules to link the security group to a security group for an EC2 instance. For more information, see [A DB Instance in a VPC Accessed by an EC2 Instance in the Same VPC \(p. 246\)](#).

AWS CLI

To restore a DB cluster from a DB cluster snapshot, use the AWS CLI command [restore-db-cluster-from-snapshot](#).

In this example, you restore from a previously created DB cluster snapshot named *mydbclustersnapshot*. You restore to a new DB cluster named *mynewdbcluster*.

Example

For Linux, OS X, or Unix:

```
aws rds restore-db-cluster-from-snapshot \
--db-cluster-identifier mynewdbcluster \
--snapshot-identifier mydbclustersnapshot \
--engine aurora/aurora-postgresql
```

For Windows:

```
aws rds restore-db-cluster-from-snapshot ^
--db-instance-identifier mynewdbcluster ^
--snapshot-identifier mydbclustersnapshot ^
--engine aurora/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.

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 Amazon RDS API function [RestoreDBClusterFromSnapshot](#) with the following parameters:

- `DBClusterIdentifier`
- `SnapshotIdentifier`

Copying a Snapshot

With Amazon RDS, you can copy automated or manual DB cluster snapshots. After you copy a snapshot, the copy is a manual snapshot.

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 \(p. 399\)](#).

Note

Amazon bills you based upon the amount of 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 \(p. 382\)](#). 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) or China (Ningxia).
- You can copy a snapshot between AWS GovCloud (US-East) and AWS GovCloud (US-West), but you can't copy a snapshot between these AWS GovCloud (US) regions and other AWS Regions.
- If you delete a source snapshot before the target snapshot becomes available, the snapshot copy may fail. Verify that the target snapshot has a status of `AVAILABLE` before you delete a source snapshot.
- You can have up to five snapshot copy requests in progress to a single destination region per account.
- Depending on the regions involved and the amount of data to be copied, a cross-region snapshot copy can take hours to complete. If there is a large number of cross-region snapshot copy requests from a given source AWS Region, 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 copy starts.

Snapshot Retention

Amazon RDS deletes automated snapshots at the end of their retention period, when you disable automated snapshots for a DB cluster, or when you delete a DB cluster. If you want to keep an automated snapshot 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. If you are copying an encrypted snapshot that has been shared from another AWS account, you must have access to the KMS encryption 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 an Encrypted Snapshot \(p. 399\)](#).

Handling Encryption

You can copy a snapshot that has been encrypted using an AWS KMS encryption 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 encryption key as the original snapshot, or you can specify a different KMS encryption key. If you copy an encrypted snapshot across regions, you can't use the same KMS encryption key for the copy as used for the source snapshot, because KMS keys are region-specific. Instead, you must specify a KMS key valid in the destination AWS Region. You can't use the default AWS KMS encryption key when copying snapshots across AWS Regions, even in the same AWS account.

Note

For Amazon Aurora DB cluster snapshots, you can't encrypt an unencrypted DB cluster snapshot when you copy the snapshot.

Copying Snapshots Across AWS Regions

When you copy a snapshot to an AWS Region that is different from the source snapshot's AWS Region, the first copy is a full snapshot copy, even if you copy an incremental snapshot. A full snapshot copy contains all of the data and metadata required to restore the DB instance. After the first snapshot copy, you can copy incremental snapshots of the same DB instance to the same destination region within the same AWS account.

An incremental snapshot contains only the data that has changed after the most recent snapshot of the same DB instance. Incremental snapshot copying is faster and results in lower storage costs than full snapshot copying. Incremental snapshot copying across AWS Regions is supported for both unencrypted and encrypted snapshots.

Important

For shared snapshots, copying incremental snapshots is not supported. For shared snapshots, all of the copies are full snapshots, even within the same region.

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 these 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 copy starts.

Note

When you copy a source snapshot that is a snapshot copy, the copy isn't incremental because the snapshot copy doesn't include the required metadata for incremental copies.

Aurora doesn't support incremental snapshot copying. Aurora DB cluster snapshot copies are always full copies.

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, you must 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.
mydbinstancesnapshot

Destination Region [Info](#)

US East (N. Virginia) ▾

New DB Snapshot Identifier

DB Snapshot Identifier for the new snapshot

Target Option Group (Optional) [Info](#)

No preference ▾

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. For **Enable Encryption**, choose one of the following options:

- Choose **Disable encryption** if the DB cluster snapshot isn't encrypted and you don't want to encrypt the copy.
- Choose **Enable encryption** if the DB cluster snapshot isn't encrypted but you want to encrypt the copy. In this case, for **Master Key**, specify the KMS key identifier to use to encrypt the DB cluster snapshot copy.

- Choose **Enable encryption** if the DB cluster snapshot is encrypted. In this case, you must encrypt the copy, so **Yes** is already selected. For **Master Key**, specify the KMS key identifier to use to encrypt the DB cluster snapshot copy.

9. 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, OS X, 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-1 region. When the copy is made, all tags on the original snapshot are copied to the snapshot copy.

Example

```
https://rds.us-west-1.amazonaws.com/
?Action=CopyDBClusterSnapshot
&CopyTags=true
&SignatureMethod=HmacSHA256
&SignatureVersion=4
&SourceDBSnapshotIdentifier=arn%3Aaws%3Ards%3Aus-east-1%3A123456789012%3Acluster-
snapshot%3Aaurora-cluster1-snapshot-20130805
&TargetDBSnapshotIdentifier=myclustersnapshotcopy
&Version=2013-09-09
&X-Amz-Algorithm=AWS4-HMAC-SHA256
&X-Amz-Credential=AKIADQKE4SARGYLE/20140429/us-west-1/rds/aws4_request
&X-Amz-Date=20140429T175351Z
&X-Amz-SignedHeaders=content-type;host;user-agent;x-amz-content-sha256;x-amz-date
&X-Amz-Signature=9164337efa99caf850e874a1cb7ef62f3cea29d0b448b9e0e7c53b288ddffed2
```

Copying an Encrypted DB Cluster Snapshot by Using the AWS CLI or Amazon RDS API

Use the procedures in the following sections to copy an encrypted DB cluster snapshot by using the AWS CLI or Amazon RDS API.

To cancel a copy operation once it is in progress, delete the target DB cluster snapshot identified by `--target-db-cluster-snapshot-identifier` or `TargetDBClusterSnapshotIdentifier` while that DB cluster snapshot is in **copying** status.

AWS CLI

To copy a DB cluster snapshot, use the AWS CLI `copy-db-cluster-snapshot` command. If you are copying the snapshot to another AWS Region, run the command in the AWS Region to which the snapshot will be copied.

The following options are used to copy an encrypted DB cluster snapshot:

- `--source-region` – If you are copying the snapshot to another AWS Region, specify the AWS Region that the encrypted DB cluster snapshot will be copied from.

If you are copying the snapshot to another AWS Region and you don't specify `source-region`, you must specify the `pre-signed-url` option instead. The `pre-signed-url` value must be a URL that

contains a Signature Version 4 signed request for the `CopyDBClusterSnapshot` action to be called in the source AWS Region where the DB cluster snapshot is copied from. To learn more about the `presigned-url`, see [copy-db-cluster-snapshot](#).

- `--source-db-cluster-snapshot-identifier` – The identifier for the encrypted DB cluster snapshot to be copied. If you are copying the snapshot to another AWS Region, this identifier must be in the ARN format for the source AWS Region. If that is the case, the AWS Region specified in `source-db-cluster-snapshot-identifier` must match the AWS Region specified for `--source-region`.
- `--target-db-cluster-snapshot-identifier` – The identifier for the new copy of the encrypted DB cluster snapshot.
- `--kms-key-id` – The KMS key identifier for the key to use to encrypt the copy of the DB cluster snapshot.

You can optionally use this option if the DB cluster snapshot is encrypted, you are copying the snapshot in the same AWS Region, and you want to specify a new KMS encryption 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 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-2 region to the us-east-1 region. The command is called in the us-east-1 region.

Example

For Linux, OS X, or Unix:

```
aws rds copy-db-cluster-snapshot \
--source-db-cluster-snapshot-identifier arn:aws:rds:us-west-2:123456789012:cluster-
snapshot:aurora-cluster1-snapshot-20161115 \
--target-db-cluster-snapshot-identifier myclustersnapshotcopy \
--source-region us-west-2 \
--kms-key-id my-us-east-1-key
```

For Windows:

```
aws rds copy-db-cluster-snapshot ^
--source-db-cluster-snapshot-identifier arn:aws:rds:us-west-2:123456789012:cluster-
snapshot:aurora-cluster1-snapshot-20161115 ^
--target-db-cluster-snapshot-identifier myclustersnapshotcopy ^
--source-region us-west-2 ^
--kms-key-id my-us-east-1-key
```

RDS API

To copy a DB cluster snapshot, use the Amazon RDS API `CopyDBClusterSnapshot` operation. If you are copying the snapshot to another AWS Region, perform the action in the AWS Region to which the snapshot will be copied.

The following parameters are used to copy an encrypted DB cluster snapshot:

- `SourceDBClusterSnapshotIdentifier` – The identifier for the encrypted DB cluster snapshot to be copied. If you are copying the snapshot to another AWS Region, this identifier must be in the ARN format for the source AWS Region.
- `TargetDBClusterSnapshotIdentifier` – The identifier for the new copy of the encrypted DB cluster snapshot.

- **KmsKeyId** – The KMS key identifier for the key to use to encrypt the copy of the DB cluster snapshot.

You can optionally use this parameter if the DB cluster snapshot is encrypted, you are copying the snapshot in the same AWS Region, and you want to specify a new KMS encryption key to use to encrypt the copy. Otherwise, the copy of the DB cluster snapshot is encrypted with the same KMS key as the source DB cluster snapshot.

You must use this parameter if the DB cluster snapshot is encrypted and you are copying the snapshot to another AWS Region. In that case, you must specify a KMS key for the destination AWS Region.

- **PreSignedUrl** – If you are copying the snapshot to another AWS Region, you must specify the **PreSignedUrl** parameter. The **PreSignedUrl** value must be a URL that contains a Signature Version 4 signed request for the `CopyDBClusterSnapshot` action to be called in the source AWS Region where the DB cluster snapshot is copied from. To learn more about using a presigned URL, see [CopyDBClusterSnapshot](#).

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

The following code example copies the encrypted DB cluster snapshot from the us-west-2 region to the us-east-1 region. The action is called in the us-east-1 region.

Example

```
https://rds.us-east-1.amazonaws.com/
?Action=CopyDBClusterSnapshot
&KmsKeyId=my-us-east-1-key
&PreSignedUrl=https%253A%252F%252Frds.us-west-2.amazonaws.com%252F
%253FAction%253DCopyDBClusterSnapshot
%2526DestinationRegion%253Dus-east-1
%2526KmsKeyId%253Dmy-us-east-1-key
%2526SourceDBClusterSnapshotIdentifier%253Darn%25253Aaws%25253Ards%25253Aus-
west-2%25253A123456789012%25253Acluster-snapshot%25253Aaurora-cluster1-snapshot-20161115
%2526SignatureMethod%253DHmacSHA256
%2526SignatureVersion%253D4
%2526Version%253D2014-10-31
%2526X-Amz-Algorithm%253DAWS4-HMAC-SHA256
%2526X-Amz-Credential%253DAKIADQKE4SARGYLE%252F20161117%252Fus-west-2%252Frds
%252Faws4_request
%2526X-Amz-Date%253D20161117T215409Z
%2526X-Amz-Expires%253D3600
%2526X-Amz-SignedHeaders%253Dcontent-type%253Bhost%253Buser-agent%253Bx-amz-
content-sha256%253Bx-amz-date
%2526X-Amz-Signature
%253D255a0f17b4e717d3b67fad163c3ec26573b882c03a65523522cf890a67fca613
&SignatureMethod=HmacSHA256
&SignatureVersion=4
&SourceDBClusterSnapshotIdentifier=arn%3Aaws%3Ards%3Aus-
west-2%3A123456789012%3Acluster-snapshot%3Aaurora-cluster1-snapshot-20161115
&TargetDBClusterSnapshotIdentifier=myclustersnapshotcopy
&Version=2014-10-31
&X-Amz-Algorithm=AWS4-HMAC-SHA256
&X-Amz-Credential=AKIADQKE4SARGYLE/20161117/us-east-1/rds/aws4_request
&X-Amz-Date=20161117T221704Z
&X-Amz-SignedHeaders=content-type;host;user-agent;x-amz-content-sha256;x-amz-date
&X-Amz-Signature=da4f2da66739d2e722c85fcfd225dc27bba7e2b8dbea8d8612434378e52adccf
```

Copying a DB Cluster Snapshot Across Accounts

You can enable other AWS accounts to copy DB cluster snapshots that you specify by using the Amazon RDS API `ModifyDBClusterSnapshotAttribute` and `CopyDBClusterSnapshot` actions. You can

only copy DB cluster snapshots across accounts in the same AWS Region. The cross-account copying process works as follows, where Account A is making the snapshot available to copy, and Account B is copying it.

1. Using Account A, call `ModifyDBClusterSnapshotAttribute`, specifying `restore` for the `AttributeName` parameter, and the ID for Account B for the `ValuesToAdd` parameter.
2. (If the snapshot is encrypted) Using Account A, update the key policy for the KMS key, first adding the ARN of Account B as a Principal, and then allow the `kms:CreateGrant` action.
3. (If the snapshot is encrypted) Using Account B, choose or create an IAM user and attach an IAM policy to that user that allows it to copy an encrypted DB cluster snapshot using your KMS key.
4. Using Account B, call `CopyDBClusterSnapshot` and use the `SourceDBClusterSnapshotIdentifier` parameter to specify the ARN of the DB cluster snapshot to be copied, which must include the ID for Account A.

To list all of the AWS accounts permitted to restore a DB cluster snapshot, use the [DescribeDBSnapshotAttributes](#) or [DescribeDBClusterSnapshotAttributes](#) API operation.

To remove sharing permission for an AWS account, use the `ModifyDBSnapshotAttribute` or `ModifyDBClusterSnapshotAttribute` action with `AttributeName` set to `restore` and the ID of the account to remove in the `ValuesToRemove` parameter.

Copying an Unencrypted DB Cluster Snapshot to Another Account

Use the following procedure to copy an unencrypted DB cluster snapshot to another account in the same AWS Region.

1. In the source account for the DB cluster snapshot, call `ModifyDBClusterSnapshotAttribute`, specifying `restore` for the `AttributeName` parameter, and the ID for the target account for the `ValuesToAdd` parameter.

Running the following example using the account 987654321 permits two AWS account identifiers, 123451234512 and 123456789012, to restore the DB cluster snapshot named manual-snapshot1.

```
https://rds.us-west-2.amazonaws.com/
?Action=ModifyDBClusterSnapshotAttribute
&AttributeName=restore
&DBClusterSnapshotIdentifier>manual-snapshot1
&SignatureMethod=HmacSHA256&SignatureVersion=4
&ValuesToAdd.member.1=123451234512
&ValuesToAdd.member.2=123456789012
&Version=2014-10-31
&X-Amz-Algorithm=AWS4-HMAC-SHA256
&X-Amz-Credential=AKIADQKE4SARGYLE/20150922/us-west-2/rds/aws4_request
&X-Amz-Date=20150922T220515Z
&X-Amz-SignedHeaders=content-type;host;user-agent;x-amz-content-sha256;x-amz-date
&X-Amz-Signature=ef38f1ce3dab4e1dbf113d8d2a265c67d17ece1999ffd36be85714ed36ddbb3
```

2. In the target account, call `CopyDBClusterSnapshot` and use the `SourceDBClusterSnapshotIdentifier` parameter to specify the ARN of the DB cluster snapshot to be copied, which must include the ID for the source account.

Running the following example using the account 123451234512 copies the DB cluster snapshot aurora-cluster1-snapshot-20130805 from account 987654321 and creates a DB cluster snapshot named dbclustersnapshot1.

```
https://rds.us-west-2.amazonaws.com/
?Action=CopyDBClusterSnapshot
```

```
&CopyTags=true
&SignatureMethod=HmacSHA256
&SignatureVersion=4
&SourceDBClusterSnapshotIdentifier=arn:aws:rds:us-west-2:987654321:cluster-
snapshot:aurora-cluster1-snapshot-20130805
&TargetDBClusterSnapshotIdentifier=dbclustersnapshot1
&Version=2013-09-09
&X-Amz-Algorithm=AWS4-HMAC-SHA256
&X-Amz-Credential=AKIADQKE4SARGYLE/20150922/us-west-2/rds/aws4_request
&X-Amz-Date=20140429T175351Z
&X-Amz-SignedHeaders=content-type;host;user-agent;x-amz-content-sha256;x-amz-date
&X-Amz-Signature=9164337efa99caf850e874a1cb7ef62f3cea29d0b448b9e0e7c53b288ddffed2
```

Copying an Encrypted DB Cluster Snapshot to Another Account

Use the following procedure to copy an encrypted DB cluster snapshot to another account in the same AWS Region.

1. In the source account for the DB cluster snapshot, call `ModifyDBClusterSnapshotAttribute`, specifying `restore` for the `AttributeName` parameter, and the ID for the target account for the `ValuesToAdd` parameter.

Running the following example using the account 987654321 permits two AWS account identifiers, 123451234512 and 123456789012, to restore the DB cluster snapshot named `manual-snapshot1`.

```
https://rds.us-west-2.amazonaws.com/
?Action=ModifyDBClusterSnapshotAttribute
&AttributeName=restore
&DBClusterSnapshotIdentifier=manual-snapshot1
&SignatureMethod=HmacSHA256&SignatureVersion=4
&ValuesToAdd.member.1=123451234512
&ValuesToAdd.member.2=123456789012
&Version=2014-10-31
&X-Amz-Algorithm=AWS4-HMAC-SHA256
&X-Amz-Credential=AKIADQKE4SARGYLE/20150922/us-west-2/rds/aws4_request
&X-Amz-Date=20150922T220515Z
&X-Amz-SignedHeaders=content-type;host;user-agent;x-amz-content-sha256;x-amz-date
&X-Amz-Signature=ef38f1ce3dab4e1dbf113d8d2a265c67d17ece1999ffd36be85714ed36ddbb3
```

2. In the source account for the DB cluster snapshot, update the key policy for the KMS key, first adding the ARN of the target account as a `Principal`, and then allow the `kms:CreateGrant` action. For more information, see [Allowing Access to an AWS KMS Encryption Key \(p. 400\)](#).
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 \(p. 401\)](#).
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.

For more information on copying a snapshot, see [Copying a Snapshot \(p. 388\)](#). For more information on restoring a DB instance from a DB cluster snapshot, see [Restoring from a DB Cluster Snapshot \(p. 385\)](#).

For more information on restoring a DB cluster from a DB cluster snapshot, see [Overview of Backing Up and Restoring an Aurora DB Cluster \(p. 380\)](#).

You can share a manual snapshot with up to 20 other AWS accounts. You can also share an unencrypted manual snapshot as public, which makes the snapshot available to all AWS accounts. Take care when sharing a snapshot as public so that none of your private information is included in any of your public snapshots.

The following limitations apply 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 an Encrypted Snapshot

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 \(p. 175\)](#). To do this, you must take the following steps:

1. Share the AWS Key Management Service (AWS KMS) encryption key that was used to encrypt the snapshot with any accounts that you want to be able to access the snapshot.

You can share AWS KMS encryption 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 Encryption Key \(p. 400\)](#) 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 AWS KMS encryption key of the AWS account that shared the snapshot.

Allowing Access to an AWS KMS Encryption 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 KMS key that encrypted the snapshot. To allow another AWS account access to an AWS KMS key, update the key policy for the KMS key with the ARN of the AWS account that you are sharing to as a `Principal` in the KMS key policy, and then allow the `kms:CreateGrant` action.

After you have given an AWS account access to your KMS encryption key, to copy your encrypted snapshot, that AWS account must create an AWS Identity and Access Management (IAM) user if it doesn't already have one. In addition, that AWS account must also attach an IAM policy to that IAM user that allows the IAM 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 KMS security restrictions.

In the following key policy example, user `111122223333` is the owner of the KMS encryption 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.

Warning

If you set **DB snapshot visibility** to **Public**, all AWS accounts can restore a DB cluster from your manual DB cluster snapshot and have access to your data. Do not share any manual DB cluster snapshots that contain private information as **Public**.

- If the source is encrypted, **DB snapshot visibility** is set as **Private** because encrypted snapshots can't be shared as public.
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

Add

AWS account ID

Delete

Please add AWS account ID

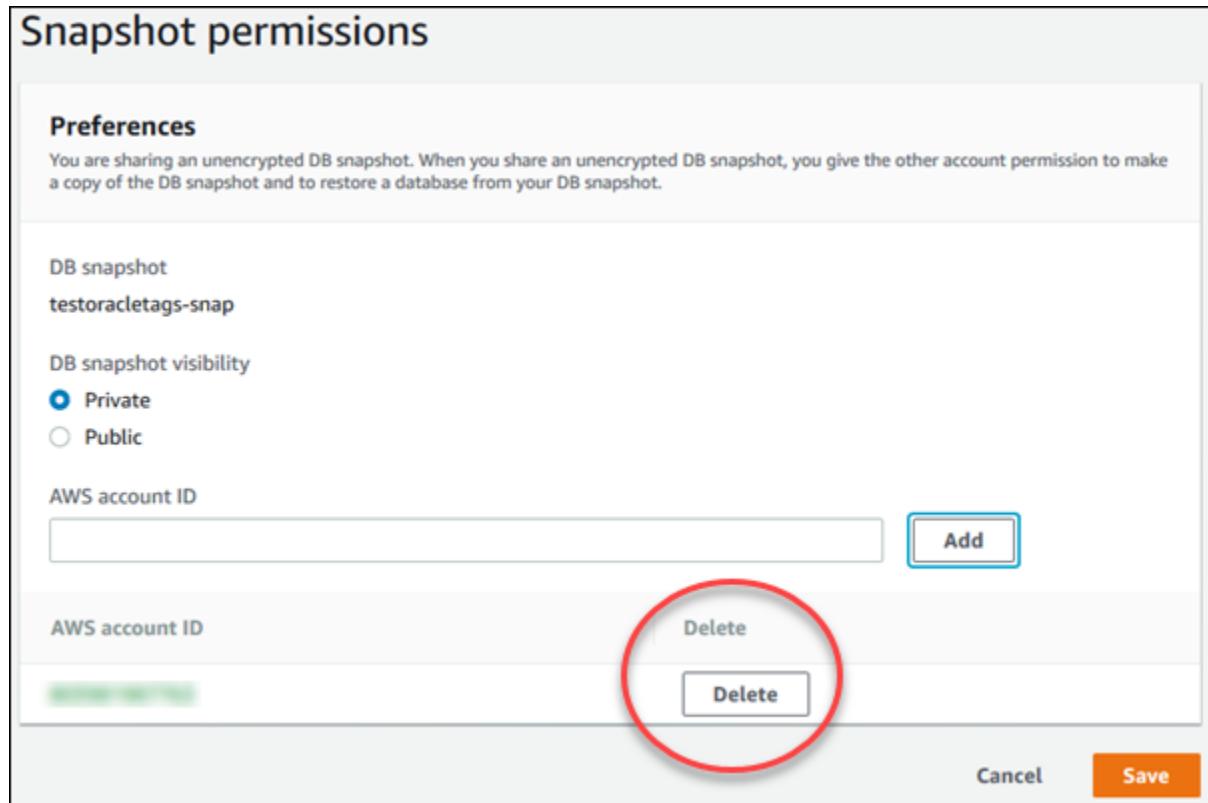
Cancel

Save

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**.
5. To remove permission for an AWS account, choose **Delete** for the AWS account identifier for that account from the list of authorized accounts.



6. 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.

The following example permits two AWS account identifiers, `123451234512` and `123456789012`, to restore the DB cluster snapshot named `manual-cluster-snapshot1`, and removes the `all` attribute value to mark the snapshot as private.

```
aws rds modify-db-cluster-snapshot-attribute \
--db-cluster-snapshot-identifier manual-cluster-snapshot1 \
--attribute-name restore \
--values-to-add '[ "111122223333", "444455556666" ]'
```

To remove an AWS account identifier from the list, use the `-- values-to-remove` parameter. The following example prevents AWS account ID 444455556666 from restoring the snapshot.

```
aws rds modify-db-cluster-snapshot-attribute \
--db-cluster-snapshot-identifier manual-cluster-snapshot1 \
--attribute-name restore \
--values-to-remove '[ "444455556666 " ]'
```

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.

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, the default DB security group is applied to the new DB cluster. If you need custom DB security groups applied to your DB cluster, you must apply them explicitly using the AWS Management Console, the AWS CLI `modify-db-cluster` command, or the Amazon RDS API `ModifyDBCluster` operation after the DB instance is available.

Note

For more information about backing up and restoring an Aurora DB cluster, see [Overview of Backing Up and Restoring an Aurora DB Cluster \(p. 380\)](#). For Aurora MySQL, you can restore a provisioned DB cluster to an Aurora Serverless DB cluster. For more information, see [Restoring an Aurora Serverless DB Cluster \(p. 129\)](#).

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 **Launch DB Instance** 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 that you want to restore the cluster to.
6. For **DB instance identifier**, enter the name of the restored DB instance, and then complete the other options.
7. Choose **Launch DB Instance**.

AWS CLI

To restore a DB cluster to a specified time, use the AWS CLI command `restore-db-cluster-to-point-in-time` to create a new DB cluster.

Example

For Linux, OS X, 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
```

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`

Deleting a Snapshot

You can delete DB cluster snapshots managed by Amazon RDS when you no longer need them.

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, OS X, 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.

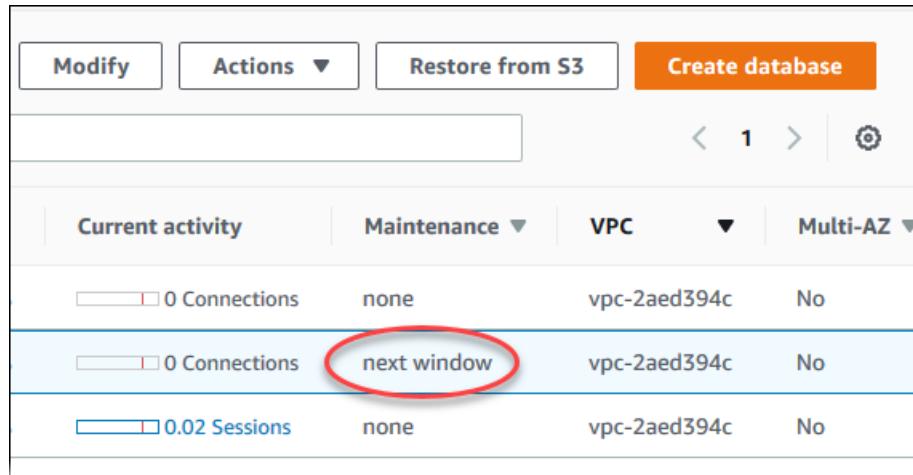
- `DBClusterSnapshotIdentifier` – The identifier for the DB cluster snapshot.

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.

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.



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.
- **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 a navigation bar with tabs: Connectivity & security, Monitoring, Logs & events, Configuration, Maintenance & backups (which is highlighted in orange), and Tags. Below the tabs is a section titled "Maintenance". It contains three status boxes: "Auto minor version upgrade" (Enabled), "Maintenance window" (mon:11:28-mon:11:58 UTC (GMT)), and "Pending maintenance next window". Under "Pending maintenance (1)", there is a table with columns: Description, Type, Status, and Apply date. A single row is listed: "Automatic minor version upgrade to postgres 9.6.11" (db-upgrade, next window, February 25th 2019, 3:28:00 am UTC-8 (local)).

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 execution 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 \(p. 411\)](#).

For information about updates to Amazon Aurora engines and instructions for upgrading and patching them, see [Database Engine Updates for Amazon Aurora MySQL \(p. 794\)](#) and [Database Engine Updates for Amazon Aurora PostgreSQL \(p. 970\)](#).

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.

AWS CLI

To apply a pending update to a DB cluster, use the [apply-pending-maintenance-action](#) AWS CLI command.

Example

For Linux, OS X, 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

You can defer a maintenance action by specifying `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 return a list of resources that have at least one pending update, use the [describe-pending-maintenance-actions](#) AWS CLI command.

Example

For Linux, OS X, 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, OS X, 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
Asia Pacific (Hong Kong)	ap-east-1	06:00–14:00 UTC
Asia Pacific (Mumbai)	ap-south-1	17:30–01:30 UTC
Asia Pacific (Osaka-Local)	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	23:00–07:00 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 (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
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. For **Actions**, choose **Modify cluster**.
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 **Apply immediately**.
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, OS X, 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 = my-cluster`
- `PreferredMaintenanceWindow = Tue:04:00-Tue:04:30`

Example

The following code example sets the maintenance window to Tuesdays from 4:00–4:30 AM UTC.

```
https://rds.us-west-2.amazonaws.com/  
?Action=ModifyDBCluster  
&DBClusterIdentifier=my-cluster  
&PreferredMaintenanceWindow=Tue:04:00-Tue:04:30  
&SignatureMethod=HmacSHA256  
&SignatureVersion=4  
&Version=2014-10-31  
&X-Amz-Algorithm=AWS4-HMAC-SHA256  
&X-Amz-Credential=AKIADQKE4SARGYLE/20140725/us-east-1/rds/aws4_request  
&X-Amz-Date=20161017T161457Z  
&X-Amz-SignedHeaders=content-type;host;user-agent;x-amz-content-sha256;x-amz-date  
&X-Amz-Signature=d6d1c65c2e94f5800ab41a3f7336625820b103713b6c063430900514e21d784
```

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 \(p. 797\)](#).

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 Amazon RDS MySQL, and you prefer to keep the Aurora MySQL clusters and RDS 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 Amazon RDS 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 a DB Instance in a DB Cluster

You might need to reboot your DB instance, usually for maintenance reasons. For example, if you make certain modifications, or if you change the DB parameter group associated with the DB instance or its DB cluster, you must reboot the instance for the changes to take effect.

Note

If a DB instance isn't using the latest changes to its associated DB parameter group, the AWS Management Console shows the DB parameter group with a status of **pending-reboot**. The **pending-reboot** parameter groups status doesn't result in an automatic reboot during the next maintenance window. To apply the latest parameter changes to that DB instance, manually reboot the DB instance. For more information about parameter groups, see [Working with DB Parameter Groups and DB Cluster Parameter Groups \(p. 286\)](#).

Rebooting a DB instance restarts the database engine service. Rebooting a DB instance results in a momentary outage, during which the DB instance status is set to *rebooting*.

You can't reboot your DB instance if it is not in the available state. Your database can be unavailable for several reasons, such as an in-progress backup, a previously requested modification, or a maintenance-window action.

Important

When you reboot the primary instance of an Amazon Aurora DB cluster, RDS also automatically restarts all of the Aurora Replicas in that DB cluster. When you reboot the primary instance of an Aurora DB cluster, no failover occurs. When you reboot an Aurora Replica, no failover occurs. To fail over an Aurora DB cluster, call the AWS CLI command [failover-db-cluster](#), or the API operation [FailoverDBCluster](#).

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.

Alternatively, choose **Cancel**.

AWS CLI

To reboot a DB instance by using the AWS CLI, call the [reboot-db-instance](#) command.

Example Simple Reboot

For Linux, OS X, 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](#) action.

Deleting a DB Instance in an Aurora DB Cluster

You can delete a DB instance in a DB cluster, including deleting the primary DB instance in a DB cluster or an Amazon Aurora Replica. To delete a DB instance, you must specify the name of the instance.

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 \(p. 674\)](#).

Deletion Protection

You can enable deletion protection so that users can't delete a DB cluster. 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. For more information about turning deletion protection on and off, see [Modifying the DB Cluster by Using the Console, CLI, and API \(p. 264\)](#).

Aurora enforces deletion protection for a DB cluster whether you perform the operation from the console, the CLI, or the API. If you try to delete a DB cluster that has deletion protection enabled, you can't do so. To be certain that you can delete the cluster, modify the cluster and disable deletion protection.

Aurora Clusters 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, but 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.

Deleting a DB Instance by Using the Console, CLI, and API

You can delete a DB instance using the AWS Management Console, the AWS CLI, or the RDS API.

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` option.

Example

For Linux, OS X, 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.

Tagging Amazon RDS Resources

You can use Amazon RDS tags to add metadata to your Amazon RDS resources. In addition, these tags can be used with IAM policies to manage access to Amazon RDS resources and to control what actions can be applied to the Amazon RDS resources. Finally, these tags can be used to track costs by grouping expenses for similarly tagged resources.

All Amazon RDS resources can be tagged

- DB instances
- DB clusters
- Read Replicas
- DB snapshots
- DB cluster snapshots
- Reserved DB instances
- Event subscriptions
- DB option groups
- DB parameter groups
- DB cluster parameter groups
- DB security groups
- DB subnet groups

For information on managing access to tagged resources with IAM policies, see [Identity and Access Management in Amazon Aurora \(p. 191\)](#).

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.

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 [Cost Allocation and Tagging in About AWS Billing and Cost Management](#).

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.

Note

You can add a tag to a snapshot, however, your bill will not reflect this grouping.

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 \(p. 425\)](#).

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.

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.

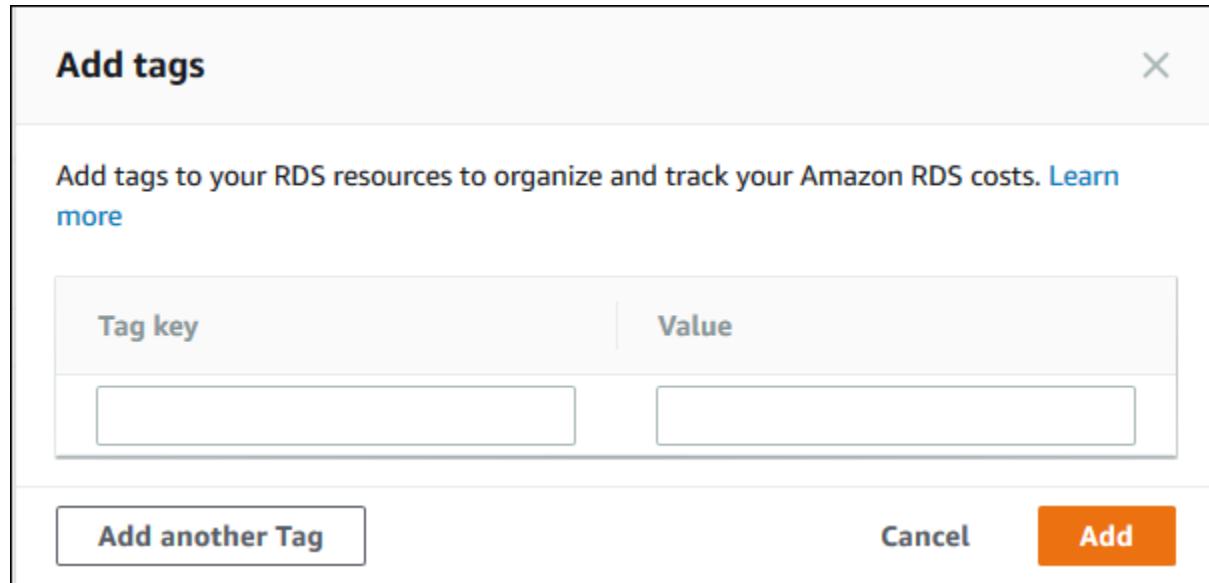
To add a tag to a DB instance

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.

Tags (1)		Edit	Remove	Add
<input type="text"/>	Filter tag key	< 1 >		
<input checked="" type="checkbox"/> Tag key	workload-type	Value	other	

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 \(p. 425\)](#).

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 \(p. 425\)](#).

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	<p>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 "rds:" or "aws:". The string can only contain only the set of Unicode letters, digits, white-space, '_', '.', '/', '=', '+', '-' (Java regex: "<code>^([\\p{L}\\p{Z}]\\p{N}_-:=+\\-)*\$</code>").</p> <p>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.</p>

Tagging Element	Description
Value	<p>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 "rds:" or "aws:". The string can only contain only the set of Unicode letters, digits, white-space, '_', '!', '/', '=', '+', '-' (Java regex: "<code>^([\p{L}\p{Z}]\p{N}_:=+\\-]*\$</code>").</p> <p>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.</p>

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	
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	
Asia Pacific (Hong Kong)	ap-east-1	rds.ap-east-1.amazonaws.com	HTTPS	
Asia Pacific (Mumbai)	ap-south-1	rds.ap-south-1.amazonaws.com	HTTPS	
Asia Pacific (Osaka-Local)	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	

Region Name	Region	Endpoint	Protocol	
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	
China (Beijing)	cn-north-1	rds.cn-north-1.amazonaws.com.cn	HTTPS	
China (Ningxia)	cn-northwest-1	rds.cn-northwest-1.amazonaws.com.cn	HTTPS	
EU (Frankfurt)	eu-central-1	rds.eu-central-1.amazonaws.com	HTTPS	
EU (Ireland)	eu-west-1	rds.eu-west-1.amazonaws.com	HTTPS	
EU (London)	eu-west-2	rds.eu-west-2.amazonaws.com	HTTPS	
EU (Paris)	eu-west-3	rds.eu-west-3.amazonaws.com	HTTPS	
EU (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 (Sao 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	

The following table shows the format that you should use when constructing an ARN for a particular Amazon RDS resource type.

Resource Type	ARN Format
DB instance	<p>arn:aws:rds:<region>:<account>:db:<name></p> <p>For example:</p> <pre>arn:aws:rds:us-east-2:123456789012:db:my-mysql-instance-1</pre>

Resource Type	ARN Format
DB cluster	<p>arn:aws:rds:<region>:<account>:cluster:<name></p> <p>For example:</p> <pre>arn:aws:rds:us-east-2:123456789012:cluster:my-aurora-cluster-1</pre>
Event subscription	<p>arn:aws:rds:<region>:<account>:es:<name></p> <p>For example:</p> <pre>arn:aws:rds:us-east-2:123456789012:es:my-subscription</pre>
DB parameter group	<p>arn:aws:rds:<region>:<account>:pg:<name></p> <p>For example:</p> <pre>arn:aws:rds:us-east-2:123456789012:pg:my-param-enable-logs</pre>
DB cluster parameter group	<p>arn:aws:rds:<region>:<account>:cluster-pg:<name></p> <p>For example:</p> <pre>arn:aws:rds:us-east-2:123456789012:cluster-pg:my-cluster-param-timezone</pre>
Reserved DB instance	<p>arn:aws:rds:<region>:<account>:ri:<name></p> <p>For example:</p> <pre>arn:aws:rds:us-east-2:123456789012:ri:my-reserved-postgresql</pre>
DB security group	<p>arn:aws:rds:<region>:<account>:secgrp:<name></p> <p>For example:</p> <pre>arn:aws:rds:us-east-2:123456789012:secgrp:my-public</pre>
Automated DB snapshot	<p>arn:aws:rds:<region>:<account>:snapshot:rds:<name></p> <p>For example:</p> <pre>arn:aws:rds:us-east-2:123456789012:snapshot:rds:my-mysql-db-2019-07-22-07-23</pre>
Automated DB cluster snapshot	<p>arn:aws:rds:<region>:<account>:cluster-snapshot:rds:<name></p> <p>For example:</p> <pre>arn:aws:rds:us-east-2:123456789012:cluster-snapshot:rds:my-aurora-cluster-2019-07-22-16-16</pre>

Resource Type	ARN Format
Manual DB snapshot	<p>arn:aws:rds:<region>:<account>:snapshot:<name></p> <p>For example:</p> <pre>arn:aws:rds:us-east-2:123456789012:snapshot:my-mysql-db-snap</pre>
Manual DB cluster snapshot	<p>arn:aws:rds:<region>:<account>:cluster-snapshot:<name></p> <p>For example:</p> <pre>arn:aws:rds:us-east-2:123456789012:cluster-snapshot:my-aurora-cluster-snap</pre>
DB subnet group	<p>arn:aws:rds:<region>:<account>:subgrp:<name></p> <p>For example:</p> <pre>arn:aws:rds:us-east-2:123456789012:subgrp:my-subnet-10</pre>

Getting an Existing ARN

You can get the ARN of an RDS resource by using the AWS Management Console, AWS Command Line Interface (AWS CLI), or RDS API.

Console

To get an ARN from the AWS Management Console, navigate to the resource you want an ARN for, and view the details for that resource. For example, you can get the ARN for a DB instance from the **Configuration** tab of the DB instance details, as shown following.

Connectivity Monitoring Logs & events Configuration

Instance

Configuration

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:██████████:db:oracle-instance1

Resource id
██████████

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, OS X, or Unix:

```
aws rds describe-db-instances \
--db-instance-identifier DBInstanceIdentifier \
--region us-west-2
```

For Windows:

```
aws rds describe-db-instances ^
--db-instance-identifier DBInstanceIdentifier ^
--region us-west-2
```

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
<code>DescribeEventSubscriptions</code>	<code>EventSubscriptionArn</code>

RDS API Operation	ARN Property
DescribeCertificates	CertificateArn
DescribeDBParameterGroups	DBParameterGroupArn
DescribeDBClusterParameterGroups	DBClusterParameterGroupArn
DescribeDBInstances	DBInstanceArn
DescribeDBSecurityGroups	DBSecurityGroupArn
DescribeDBSnapshots	DBSnapshotArn
DescribeEvents	SourceArn
DescribeReservedDBInstances	ReservedDBInstanceArn
DescribeDBSubnetGroups	DBSubnetGroupArn
DescribeDBClusters	DBClusterArn
DescribeDBClusterSnapshots	DBClusterSnapshotArn

Amazon Aurora Updates

Amazon Aurora releases updates regularly. Updates are applied to Amazon Aurora DB clusters during system maintenance windows. The timing when updates are applied depends on the region and maintenance window setting for the DB cluster, and also the type of update. Updates require a database restart, so you 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](#).

Following, you can find information on general updates to Amazon Aurora. Some of the updates applied to Amazon Aurora are specific to a database engine supported by Aurora. For more information about database engine updates for Aurora, see the following table.

Database Engine	Updates
Amazon Aurora MySQL	See Database Engine Updates for Amazon Aurora MySQL (p. 794)
Amazon Aurora PostgreSQL	See Database Engine Updates for Amazon Aurora PostgreSQL (p. 970)

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> <pre>SELECT AURORA_VERSION();</pre>

Monitoring an Amazon Aurora DB Cluster

This section shows how to monitor Amazon Aurora. Aurora involves clusters of database servers that are connected in a replication topology. Monitoring a Aurora cluster typically requires checking the health of multiple DB instances. The instances might have specialized roles, handling mostly write operations, only read operations, or a combination of both. You also monitor the overall health of the cluster by measuring the *replication lag*, the amount of time for changes made by one DB instance to be available to the other instances.

Topics

- [Overview of Monitoring Amazon RDS \(p. 434\)](#)
- [Viewing an Amazon Aurora DB Cluster \(p. 446\)](#)
- [DB Cluster Status \(p. 452\)](#)
- [DB Instance Status \(p. 454\)](#)
- [Monitoring Amazon Aurora DB Cluster Metrics \(p. 457\)](#)
- [Enhanced Monitoring \(p. 469\)](#)
- [Using Amazon RDS Performance Insights \(p. 476\)](#)
- [Using Amazon Aurora Recommendations \(p. 520\)](#)
- [Using Database Activity Streams with Aurora PostgreSQL \(p. 525\)](#)
- [Using Amazon RDS Event Notification \(p. 543\)](#)
- [Viewing Amazon RDS Events \(p. 561\)](#)
- [Amazon Aurora Database Log Files \(p. 563\)](#)
- [Logging Amazon RDS API Calls with AWS CloudTrail \(p. 574\)](#)

Overview of Monitoring Amazon RDS

Monitoring is an important part of maintaining the reliability, availability, and performance of Amazon RDS 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. Before you start monitoring Amazon RDS, we recommend that you create a monitoring plan that includes answers to the following questions:

- What are your monitoring goals?
- What resources will you monitor?
- How often will you monitor these resources?
- What monitoring tools will you use?
- Who will perform the monitoring tasks?
- Who should be notified when something goes wrong?

The next step is to establish a baseline for normal Amazon RDS performance in your environment, by measuring performance at various times and under different load conditions. As you monitor Amazon RDS, you should consider storing historical monitoring data. This stored data will give you a baseline to compare against with current performance data, identify normal performance patterns and performance anomalies, and devise methods to address issues.

For example, with Amazon RDS, you can monitor network throughput, I/O for read, write, and/or metadata operations, client connections, and burst credit balances for your DB instances. When performance falls outside your established baseline, you might need to change the instance class of your DB instance or the number of DB instances and Read Replicas that are available for clients in order to optimize your database availability for your workload.

In general, acceptable values for performance metrics depend on what your baseline looks like and what your application is doing. Investigate consistent or trending variances from your baseline. Advice about specific types of metrics follows:

- **High CPU or RAM consumption** – High values for CPU or RAM consumption might be appropriate, provided that they are 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** – Consider constraining database connections if you see high numbers of user connections in conjunction with decreases in instance performance and response time. The best number of user connections for your DB instance will vary based on your instance class and the complexity of the operations being performed. You can determine the number of database connections by associating your DB instance with a parameter group where the `User_Connections` parameter is set to a value other than 0 (unlimited). You can either use an existing parameter group or create a new one. For more information, see [Working with DB Parameter Groups and DB Cluster Parameter Groups \(p. 286\)](#).
- **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 your typical working set will fit into memory to minimize read and write operations.

Monitoring Tools

AWS provides various tools that you can use to monitor Amazon RDS. You can configure some of these tools to do the monitoring for you, while some of the tools require manual intervention. We recommend that you automate monitoring tasks as much as possible.

Automated Monitoring Tools

You can use the following automated monitoring tools to watch Amazon RDS and report when something is wrong:

- **Amazon RDS Events** – Subscribe to Amazon RDS events to be notified when changes occur with a DB instance, DB cluster, DB cluster snapshot, DB parameter group, or DB security group. For more information, see [Using Amazon RDS Event Notification \(p. 543\)](#).
- **Database log files** – View, download, or watch database log files using the Amazon RDS console or Amazon RDS API operations. You can also query some database log files that are loaded into database tables. For more information, see [Amazon Aurora Database Log Files \(p. 563\)](#).
- **Amazon RDS Enhanced Monitoring** — Look at metrics in real time for the operating system. For more information, see [Enhanced Monitoring \(p. 469\)](#).

In addition, Amazon RDS integrates with Amazon CloudWatch for additional monitoring capabilities:

- **Amazon CloudWatch Metrics** – Amazon RDS automatically sends metrics to CloudWatch every minute for each active database. You are not charged additionally for Amazon RDS metrics in CloudWatch. For more information, see [the section called "Viewing DB Instance Metrics" \(p. 443\)](#).
- **Amazon CloudWatch Alarms** – You can watch a single Amazon RDS metric over a specific time period, and perform one or more actions based on the value of the metric relative to a threshold you set. For more information, see [Monitoring with Amazon CloudWatch \(p. 436\)](#)
- **Amazon CloudWatch Logs** – Most DB engines enable you to monitor, store, and access your database log files in CloudWatch Logs. For more information, see [Amazon CloudWatch Logs User Guide](#)

Manual Monitoring Tools

Another important part of monitoring Amazon RDS involves manually monitoring 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 utilizing
 - The amount of memory and CPU being utilized for a DB instance
 - The amount of network traffic to and from a DB instance
- From the AWS 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 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

Monitoring with Amazon CloudWatch

You can monitor DB instances using Amazon CloudWatch, which collects and processes raw data from Amazon RDS into readable, near real-time metrics. These statistics are recorded for a period of two weeks, so that you can access historical information and gain a better perspective on how your web application or service is performing. By default, Amazon RDS metric data is automatically sent to CloudWatch in 1-minute periods. For more information about CloudWatch, see [What Are Amazon CloudWatch, Amazon CloudWatch Events, and Amazon CloudWatch Logs?](#) in the *Amazon CloudWatch User Guide*.

Note

If you are using Amazon RDS Performance Insights, additional metrics are available. For more information, see [Performance Insights Metrics Published to Amazon CloudWatch \(p. 509\)](#).

Amazon RDS Metrics and Dimensions

When you use Amazon RDS resources, Amazon RDS sends metrics and dimensions to Amazon CloudWatch every minute. You can use the following procedures to view the metrics for Amazon RDS.

To view metrics using the Amazon CloudWatch console

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

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. If necessary, change the AWS Region. From the navigation bar, choose the AWS Region where your AWS resources reside. For more information, see [Regions and Endpoints](#).
3. In the navigation pane, choose **Metrics**. Choose the **RDS** metric namespace.

The screenshot shows the CloudWatch Metrics console interface. At the top, there are three tabs: "All metrics" (selected), "Graphed metrics", and "Graph options". Below the tabs, the navigation path is "All > RDS". A search bar contains the placeholder text "Search for any metric, dimension or resource id". The main content area displays "168 Metrics" and is organized into three sections: "DbClusterIdentifier, EngineName" (3 Metrics), "Per-Database Metrics" (45 Metrics), and "By Database Class" (45 Metrics).

4. Choose a metric dimension, for example **By Database Class**.
5. 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 screenshot shows the "By Database Class" section from the previous screenshot. The table lists metrics grouped by database class. A context menu is open over the row for "db.r4.large" under the "Metric Name" column. The menu options are: "Add to search", "Search for this only", "Remove from graph", "Graph this metric only", "Graph all search results", and "What is this?".

	DatabaseClass (45)	Metric Name
<input checked="" type="checkbox"/>	db.r4.large	BufferCacheHitRatio
<input type="checkbox"/>	db.t2.micro	Add to search
<input type="checkbox"/>	db.r4.large	Search for this only
<input type="checkbox"/>	db.r4.large	Remove from graph
<input type="checkbox"/>	db.t2.micro	Graph this metric only
<input type="checkbox"/>	db.t2.micro	Graph all search results
<input type="checkbox"/>	db.r4.large ▾	What is this?

To view metrics using the AWS CLI

- At a command prompt, use the following command:

```
aws cloudwatch list-metrics --namespace AWS/RDS
```

Amazon RDS Metrics

The AWS/RDS namespace includes the following metrics.

Metric	Description
BinLogDiskUsage	<p>The amount of disk space occupied by binary logs on the master. Applies to MySQL read replicas.</p> <p>Units: Bytes</p>
BurstBalance	<p>The percent of General Purpose SSD (gp2) burst-bucket I/O credits available.</p> <p>Units: Percent</p>
CPUUtilization	<p>The percentage of CPU utilization.</p> <p>Units: Percent</p>
CPUCreditUsage	<p>(T2 instances) The number of CPU credits spent by the instance for CPU utilization. One CPU credit equals one vCPU running at 100 percent utilization for one minute or an equivalent combination of vCPUs, utilization, and time. For example, you might have one vCPU running at 50 percent utilization for two minutes or two vCPUs running at 25 percent utilization for two minutes.</p> <p>CPU credit metrics are available at a five-minute frequency only. If you specify a period greater than five minutes, use the <code>Sum</code> statistic instead of the <code>Average</code> statistic.</p> <p>Units: Credits (vCPU-minutes)</p>
CPUCreditBalance	<p>(T2 instances) The number of earned CPU credits that an instance has accrued since it was launched or started. For T2 Standard, the CPUCreditBalance also includes the number of launch credits that have been accrued.</p> <p>Credits are accrued in the credit balance after they are earned, and removed from the credit balance when they are spent. The credit balance has a maximum limit, determined by the instance size. After the limit is reached, any new credits that are earned are discarded. For T2 Standard, launch credits don't count towards the limit.</p> <p>The credits in the CPUCreditBalance are available for the instance to spend to burst beyond its baseline CPU utilization.</p> <p>When an instance is running, credits in the CPUCreditBalance don't expire. When the instance stops, the CPUCreditBalance does not persist, and all accrued credits are lost.</p> <p>CPU credit metrics are available at a five-minute frequency only.</p>

Metric	Description
	Units: Credits (vCPU-minutes)
DatabaseConnections	The number of database connections in use. Units: Count
DiskQueueDepth	The number of outstanding IOs (read/write requests) waiting to access the disk. Units: Count
FailedSQLServerAgentJobs	The number of failed SQL Server Agent jobs during the last minute. Unit: Count/Minute
FreeableMemory	The amount of available random access memory. For Aurora, this metric reports the value of the <code>MemAvailable</code> field of <code>/proc/meminfo</code> . Units: Bytes
FreeStorageSpace	The amount of available storage space. Units: Bytes
MaximumUsedTransactionIDs	The maximum transaction ID that has been used. Applies to PostgreSQL. Units: Count
NetworkReceiveThroughput	The incoming (Receive) network traffic on the DB instance, including both customer database traffic and Amazon RDS traffic used for monitoring and replication. Units: Bytes/Second
NetworkTransmitThroughput	The outgoing (Transmit) network traffic on the DB instance, including both customer database traffic and Amazon RDS traffic used for monitoring and replication. Units: Bytes/Second
OldestReplicationSlotLag	The lagging size of the replica lagging the most in terms of WAL data received. Applies to PostgreSQL. Units: Megabytes
ReadIOPS	The average number of disk read I/O operations per second. Units: Count/Second
ReadLatency	The average amount of time taken per disk I/O operation. Units: Seconds
ReadThroughput	The average number of bytes read from disk per second. Units: Bytes/Second

Metric	Description
ReplicaLag	The amount of time a Read Replica DB instance lags behind the source DB instance. Applies to MySQL, MariaDB, and PostgreSQL Read Replicas. Units: Seconds
ReplicationSlotDiskUsage	The disk space used by replication slot files. Applies to PostgreSQL. Units: Megabytes
SwapUsage	The amount of swap space used on the DB instance. This metric is not available for SQL Server. Units: Bytes
TransactionLogsDiskUsage	The disk space used by transaction logs. Applies to PostgreSQL. Units: Megabytes
TransactionLogsGeneration	The size of transaction logs generated per second. Applies to PostgreSQL. Units: Bytes/Second
WriteIOPS	The average number of disk write I/O operations per second. Units: Count/Second
WriteLatency	The average amount of time taken per disk I/O operation. Units: Seconds
WriteThroughput	The average number of bytes written to disk per second. Units: Bytes/Second

Amazon RDS Dimensions

Amazon RDS metrics data can be filtered by using any of the dimensions in the following table:

Dimension	Description
DBInstanceIdentifier	This dimension filters the data you request for a specific database instance.
DBClusterIdentifier	This dimension filters the data you request for a specific Amazon Aurora DB cluster.
DBClusterIdentifier, Role	This dimension filters the data you request for a specific Aurora DB cluster, aggregating the metric by instance role (WRITER/READER). For example, you can aggregate metrics for all READER instances that belong to a cluster.
DatabaseClass	This dimension filters the data you request for all instances in a database class. For example, you can aggregate metrics for all instances that belong to the database class db.m1.small

Dimension	Description
EngineName	This dimension filters the data you request for the identified engine name only. For example, you can aggregate metrics for all instances that have the engine name mysql.
SourceRegion	This dimension filters the data you request for the specified region only. For example, you can aggregate metrics for all instances in the region us-east-1.

Creating CloudWatch Alarms to Monitor Amazon RDS

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 you specify, and performs 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 Auto Scaling policy.

Alarms invoke actions for sustained state changes only. CloudWatch alarms will not invoke actions simply because they are in a particular state, the state must have changed and been maintained for a specified number of 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 \(p. 351\)](#).

To set alarms using the CloudWatch console

1. Sign in to the AWS Management Console and open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. Choose **Alarms** and then choose **Create Alarm**. This launches the **Create Alarm Wizard**.
3. Choose **RDS Metrics** and scroll through the Amazon RDS metrics to locate the metric you want to place an alarm on. To display just the Amazon RDS metrics in this dialog box, search for the identifier of your resource. Choose the metric to create an alarm on and then choose **Next**.
4. Enter **Name**, **Description**, and **Whenever** values for the metric.
5. If you want CloudWatch to send you an email when the alarm state is reached, for **Whenever this alarm:**, choose **State is ALARM**. For **Send notification to:**, choose an existing SNS topic. If you choose **Create topic**, you can set the name and email addresses for a new email subscription list. This list is saved and appears in the field for future alarms.

Note

If you use **Create topic** to create a new Amazon SNS topic, the email addresses must be verified before they receive notifications. Emails are only sent when the alarm enters an alarm state. If this alarm state change happens before the email addresses are verified, they don't receive a notification.

6. At this point, the **Alarm Preview** area gives you a chance to preview the alarm you're about to create. 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](#)

Publishing Database Engine Logs to Amazon CloudWatch Logs

You can configure your Amazon RDS database engine 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, which you can manage with the CloudWatch Logs Agent. For example, you can determine when to rotate log records from a host to the log service, so you can access the raw logs when you need to.

For engine-specific information, see the following topics:

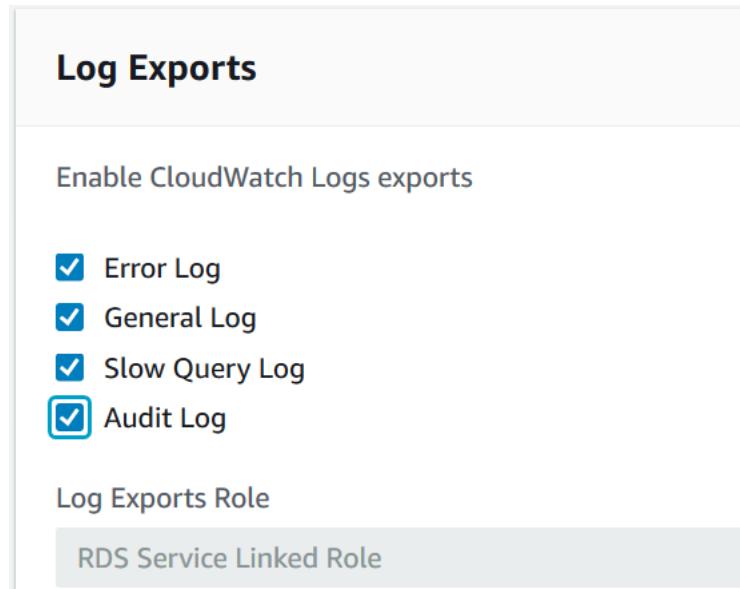
- the section called “[Publishing Aurora MySQL Logs to CloudWatch Logs](#)” (p. 759)
- the section called “[Publishing Aurora PostgreSQL Logs to CloudWatch Logs](#)” (p. 943)

Note

Before you enable log data publishing, you must 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. 235).

Configuring CloudWatch Log Integration

To publish your database log files to CloudWatch Logs, choose which logs to publish. Make this choice in the **Advanced Settings** section when you create a new DB instance. You can also modify an existing DB instance to begin publishing.



After you have enabled publishing, Amazon RDS continuously streams all of the DB instance log records to a log group. For example, you have a log group `/aws/rds/instance/log` type for each type of log that you publish. This log group is in the same AWS Region as the database instance that generates the log.

After you have published log records, you can use CloudWatch Logs to search and filter the records. For more information about searching and filtering logs, see [Searching and Filtering Log Data](#).

Viewing DB Instance Metrics

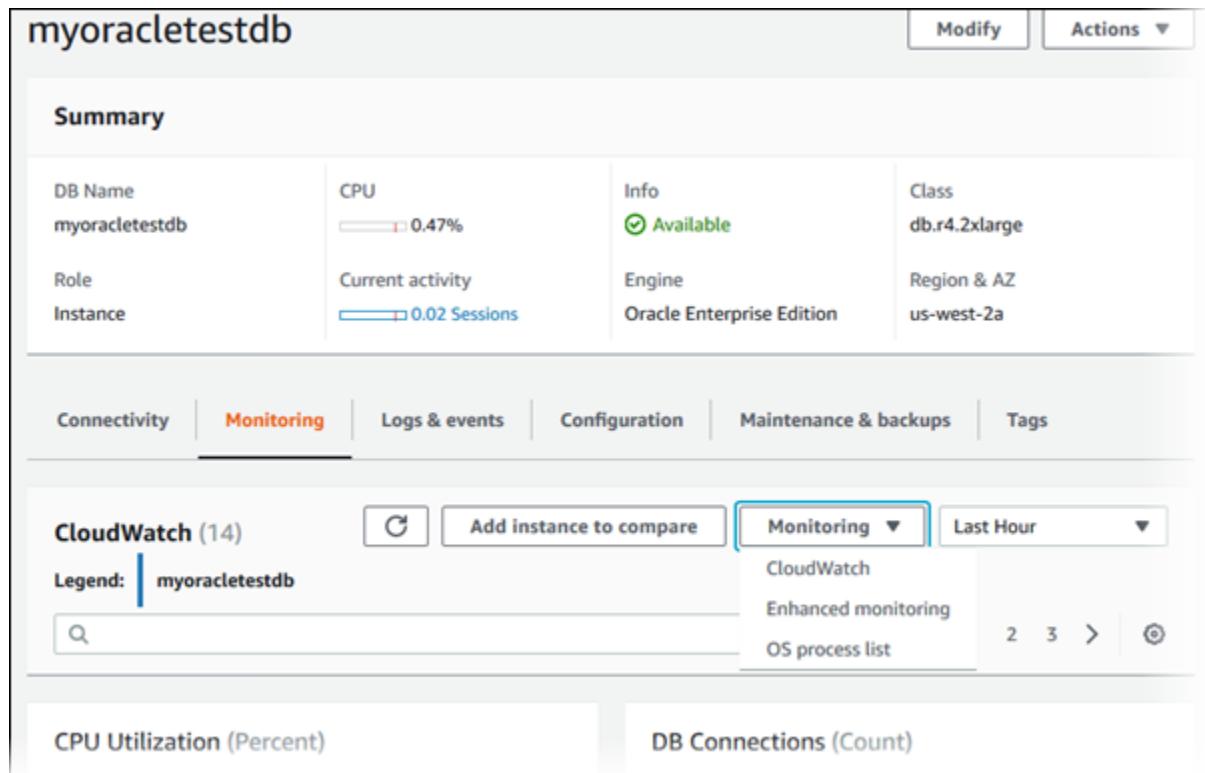
Amazon RDS provides metrics so that you can monitor the health of your DB instances. You can monitor both DB instance metrics and operating system (OS) metrics.

This section provides details on how you can view metrics for your DB instance using the RDS console and CloudWatch. For information on monitoring metrics for the operating system of your DB instance in real time using CloudWatch Logs, see [Enhanced Monitoring \(p. 469\)](#).

Viewing Metrics by Using the Console

To view DB and OS metrics 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**.
3. Choose the name of the DB instance you need information about to show its details.
4. Choose the **Monitoring** tab.
5. For **Monitoring**, choose the option for how you want to view your metrics from these:
 - **CloudWatch** – Shows a summary of DB instance metrics available from Amazon CloudWatch. Each metric includes a graph showing the metric monitored over a specific time span.
 - **Enhanced monitoring** – Shows a summary of OS metrics available for a DB instance with Enhanced Monitoring enabled. Each metric includes a graph showing the metric monitored over a specific time span.
 - **OS Process list** – Shows details for each process running in the selected instance.



Tip

You can choose the time range of the metrics represented by the graphs with the time range list.

You can choose any graph to bring up a more detailed view. You can also apply metric-specific filters to the data.

Viewing DB Instance Metrics with the CLI or API

Amazon RDS integrates with CloudWatch metrics to provide a variety of DB instance metrics. You can view CloudWatch metrics using the RDS console, AWS CLI, or API.

For a complete list of Amazon RDS metrics, go to [Amazon RDS Dimensions and Metrics](#) in the *Amazon CloudWatch User Guide*.

Viewing DB Metrics by Using the CloudWatch CLI

Note

The following CLI example requires the CloudWatch command line tools. For more information on CloudWatch and to download the developer tools, see the [Amazon CloudWatch product page](#). The `StartTime` and `EndTime` values supplied in this example are for illustrative purposes. You must substitute appropriate start and end time values for your DB instance.

To view usage and performance statistics for a DB instance

- Use the CloudWatch command `mon-get-stats` with the following parameters.

```
PROMPT>mon-get-stats FreeStorageSpace --dimensions="DBInstanceIdentifier=mydbinstance"
--statistics= Average
```

```
--namespace="AWS/RDS" --start-time 2009-10-16T00:00:00 --end-time 2009-10-16T00:02:00
```

[Viewing DB Metrics by Using the CloudWatch API](#)

The `StartTime` and `EndTime` values supplied in this example are for illustrative purposes. You must substitute appropriate start and end time values for your DB instance.

To view usage and performance statistics for a DB instance

- Call the CloudWatch API `GetMetricStatistics` with the following parameters:
 - `Statistics.member.1 = Average`
 - `Namespace = AWS/RDS`
 - `StartTime = 2009-10-16T00:00:00`
 - `EndTime = 2009-10-16T00:02:00`
 - `Period = 60`
 - `MeasureName = FreeStorageSpace`

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, you can see details about a DB cluster by choosing **Databases** from the console's navigation pane. You can also see details about DB instances that are members of an Amazon Aurora DB cluster on the **Databases** page.

The **Databases** list shows all of the DB clusters for your AWS account. When you choose a DB cluster, you see both information about the DB cluster and also a list of the DB instances that are members of that DB cluster. You can choose the identifier for a DB instance in the list to go directly to the details page for that DB instance in the RDS console.

To view the details page for a DB cluster, choose **Databases** in the navigation pane, and then choose the name of the DB cluster.

You can modify your DB cluster by choosing **Databases** from the console's navigation pane to go to the **Databases** list. To modify a DB cluster, select the DB cluster from the **Databases** list and choose **Modify**.

To modify a DB instance that is a member of a DB cluster, choose **Databases** from the console's navigation pane to go to the **Databases** list.

For example, the following image shows the details page for the DB cluster named `aurora-test`. The DB cluster has four DB instances shown in the **DB identifier** list. The writer DB instance, `dbinstance4`, is the primary DB instance for the DB cluster.

aurora-test

Related

Filter databases

DB identifier	Role	Engine	Region & AZ
aurora-test	Regional	Aurora MySQL	us-east-1
dbinstance4	Writer	Aurora MySQL	us-east-1a
dbinstance1	Reader	Aurora MySQL	us-east-1b
dbinstance2	Reader	Aurora MySQL	us-east-1b
dbinstance3	Reader	Aurora MySQL	us-east-1a

Connectivity & security **Monitoring** **Logs & events** **Configuration** **Maintenance & backups** **Tags**

Endpoints (2)

Filter endpoint

Endpoint name
aurora-test.cluster-ro- .us-east-1.rds.amazonaws.com
aurora-test.cluster- .us-east-1.rds.amazonaws.com

If you click the link for the dbinstance4 DB instance identifier, the Amazon RDS console shows the details page for the dbinstance4 DB instance, as shown in the following image.

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	

Connectivity & security	Monitoring	Logs & events	Configuration	Maintenance	Tags
---	----------------------------	-----------------------------------	-------------------------------	-----------------------------	----------------------

Connectivity & security

Endpoint & port	Network
Endpoint	Available
dbinstance4. REDACTED .us-east-1.rds.amazonaws.com	us-east-1
Port	VP
3306	vpc

AWS CLI

To view DB cluster information by using the AWS CLI, use the [describe-db-clusters](#) command. For example, the following AWS CLI command lists the DB cluster information for all of the DB clusters in the `us-east-1` region for the configured AWS account.

```
aws rds describe-db-clusters --region us-east-1
```

The command returns the following output if your AWS CLI is configured for JSON output.

```
{  
  "DBClusters": [  
    {  
      "Status": "available",  
      "Engine": "aurora",  
      "Endpoint": "sample-cluster1.cluster-123456789012.us-east-1.rds.amazonaws.com",  
      "AllocatedStorage": 1,  
      "DBClusterIdentifier": "sample-cluster1",  
      "MasterUsername": "mymasteruser",  
      "EarliestRestorableTime": "2016-03-30T03:35:42.563Z".  
    }  
  ]  
}
```

```
"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>
    </DBCluster>
  </DBClusters>
</DescribeDBClustersResult>
</DescribeDBClustersResponse>
```

```
<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>
```

DB Cluster Status

The status of a DB cluster indicates the health of the DB cluster. You can view the status of a DB cluster by using the Amazon RDS console, the AWS CLI command [describe-db-clusters](#), or the API operation [DescribeDBClusters](#).

Note

Aurora also uses another status called *maintenance status*, which is shown in the **Maintenance** column of the Amazon RDS console. This value indicates the status of any maintenance patches that need to be applied to a DB cluster. Maintenance status is independent of DB cluster status. For more information on *maintenance status*, see [Applying Updates for a DB Cluster \(p. 410\)](#).

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

DB Cluster Status	Description
available	The DB cluster is healthy and available.
backing-up	The DB cluster is currently being backed up.
backtracking	The DB cluster is currently being backtracked. This status only applies to Aurora MySQL.
cloning-failed	Cloning a DB cluster failed.
creating	The DB cluster is being created. The DB cluster is inaccessible while it is being created.
deleting	The DB cluster is being deleted.
failing-over	A failover from the primary instance to an Aurora Replica is being performed.
inaccessible-encryption-credentials	The AWS KMS key used to encrypt or decrypt the DB cluster can't be accessed.
maintenance	Amazon RDS is applying a maintenance update to the DB cluster. This status is used for DB cluster-level maintenance that RDS schedules well in advance.
migrating	A DB cluster snapshot is being restored to a DB cluster.
migration-failed	A migration failed.
modifying	The DB cluster is being modified because of a customer request to modify the DB cluster.
promoting	A Read Replica is being promoted to a standalone DB cluster.
renaming	The DB cluster is being renamed because of a customer request to rename it.
resetting-master-credentials	The master credentials for the DB cluster are being reset because of a customer request to reset them.
starting	The DB cluster is starting.
stopped	The DB cluster is stopped.
stopping	The DB cluster is being stopped.

DB Cluster Status	Description
update-iam-db-auth	IAM authorization for the DB cluster is being updated.
upgrading	The DB cluster engine version is being upgraded.

DB Instance Status

The status of a DB instance indicates the health of the DB instance. You can view the status of a DB instance by using the Amazon RDS console, the AWS CLI command [describe-db-instances](#), or the API operation [DescribeDBInstances](#).

Note

Amazon RDS also uses another status called *maintenance status*, which is shown in the **Maintenance** column of the Amazon RDS console. This value indicates the status of any maintenance patches that need to be applied to a DB instance. Maintenance status is independent of DB instance status. For more information on *maintenance status*, see [Applying Updates for a DB Cluster \(p. 410\)](#).

Find the possible status values for DB instances in the following table, which also shows how you are billed for each status. It shows if you will be billed for the DB instance and storage, billed only for storage, or not billed. For all DB instance statuses, you are always billed for backup usage.

DB Instance Status	Billed	Description
available	Billed	The DB instance is healthy and available.
backing-up	Billed	The DB instance is currently being backed up.
backtracking	Billed	The DB instance is currently being backtracked. This status only applies to Aurora MySQL.
configuring-enhanced-monitoring	Billed	Enhanced Monitoring is being enabled or disabled for this DB instance.
configuring-iam-database-auth	Billed	AWS Identity and Access Management (IAM) database authentication is being enabled or disabled for this DB instance.
configuring-log-exports	Billed	Publishing log files to Amazon CloudWatch Logs is being enabled or disabled for this DB instance.
converting-to-vpc	Billed	The DB instance is being converted from a DB instance that is not in an Amazon Virtual Private Cloud (Amazon VPC) to a DB instance that is in an Amazon VPC.
creating	Not billed	The DB instance is being created. The DB instance is inaccessible while it is being created.
deleting	Not billed	The DB instance is being deleted.
failed	Not billed	The DB instance has failed and Amazon RDS can't recover it. Perform a point-in-time restore to the latest restorable time of the DB instance to recover the data.
inaccessible-encryption-credentials	Not billed	The AWS KMS key used to encrypt or decrypt the DB instance can't be accessed.
incompatible-network	Not billed	Amazon RDS is attempting to perform a recovery action on a DB instance but can't do so because the VPC is in a state that prevents the action from being completed. This status can occur if, for example, all available IP addresses in a subnet are in use and Amazon RDS can't get an IP address for the DB instance.

DB Instance Status	Billed	Description
incompatible-option-group	Billed	Amazon RDS attempted to apply an option group change but can't do so, and Amazon RDS can't roll back to the previous option group state. For more information, check the Recent Events list for the DB instance. This status can occur if, for example, the option group contains an option such as TDE and the DB instance doesn't contain encrypted information.
incompatible-parameters	Billed	Amazon RDS can't start the DB instance because the parameters specified in the DB instance's DB parameter group aren't compatible with the DB instance. Revert the parameter changes or make them compatible with the DB instance to regain access to your DB instance. For more information about the incompatible parameters, check the Recent Events list for the DB instance.
incompatible-restore	Not billed	Amazon RDS can't do a point-in-time restore. Common causes for this status include using temp tables, using MyISAM tables with MySQL, or using Aria tables with MariaDB.
maintenance	Billed	Amazon RDS is applying a maintenance update to the DB instance. This status is used for instance-level maintenance that RDS schedules well in advance.
modifying	Billed	The DB instance is being modified because of a customer request to modify the DB instance.
moving-to-vpc	Billed	The DB instance is being moved to a new Amazon Virtual Private Cloud (Amazon VPC).
rebooting	Billed	The DB instance is being rebooted because of a customer request or an Amazon RDS process that requires the rebooting of the DB instance.
renaming	Billed	The DB instance is being renamed because of a customer request to rename it.
resetting-master-credentials	Billed	The master credentials for the DB instance are being reset because of a customer request to reset them.
restore-error	Billed	The DB instance encountered an error attempting to restore to a point-in-time or from a snapshot.
starting	Billed for storage	The DB instance is starting.
stopped	Billed for storage	The DB instance is stopped.
stopping	Billed for storage	The DB instance is being stopped.

DB Instance Status	Billed	Description
storage-full	Billed	The DB instance has reached its storage capacity allocation. This is a critical status, and we recommend that you fix this issue immediately. To do so, scale up your storage by modifying the DB instance. To avoid this situation, set Amazon CloudWatch alarms to warn you when storage space is getting low.
storage-optimization	Billed	Your DB instance is being modified to change the storage size or type. The DB instance is fully operational. However, while the status of your DB instance is storage-optimization , you can't request any changes to the storage of your DB instance. The storage optimization process is usually short, but can sometimes take up to and even beyond 24 hours.
upgrading	Billed	The database engine version is being upgraded.

Monitoring Amazon Aurora DB Cluster Metrics

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 Amazon RDS Management Console, AWS CLI, and CloudWatch API, to view Aurora metrics. For more information, see [Monitoring an Amazon Aurora DB Cluster \(p. 433\)](#).

Note

If you are using Amazon RDS Performance Insights, additional metrics are available. For more information, see [Performance Insights Metrics Published to Amazon CloudWatch \(p. 509\)](#).

Amazon Aurora Metrics

The following metrics are available from Amazon Aurora.

Amazon Aurora Metrics

The AWS/RDS namespace includes the following metrics that apply to database entities running on Amazon Aurora.

Metric	Description	Applies to
ActiveTransactions	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
AuroraBinlogReplayLag	The amount of time a replica DB cluster running on Aurora with MySQL compatibility lags behind the source DB cluster. This metric reports the value of the <code>Seconds_Behind_Master</code> field of the MySQL <code>SHOW SLAVE STATUS</code> command. This metric is useful for monitoring replica lag between Aurora DB clusters that are replicating across different AWS Regions. For more information, see Aurora MySQL Replication .	Aurora MySQL
AuroraGlobalDBReplicatedWriteIO	In a replica Aurora Global Database , the number 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.	Aurora MySQL

Metric	Description	Applies to
	Units: Bytes	
AuroraGlobalDataTransfer	The amount of redo log data transferred from the master AWS Region to a secondary AWS Region.	Aurora MySQL
	Units: Bytes	
AuroraGlobalReplicationLag	The amount of lag when replicating updates from the primary AWS Region, in milliseconds.	Aurora MySQL
	Units: Milliseconds	
AuroraReplicaLag	The amount of lag when replicating updates from the primary instance, in milliseconds.	Aurora MySQL and Aurora PostgreSQL
AuroraReplicaLagMax	The maximum amount of lag between the primary instance and each Aurora DB instance in the DB cluster, in milliseconds.	Aurora MySQL and Aurora PostgreSQL
AuroraReplicaLagMin	The minimum amount of lag between the primary instance and each Aurora DB instance in the DB cluster, in milliseconds.	Aurora MySQL and Aurora PostgreSQL
BacktrackChangeRecords	The number of backtrack change records created over five minutes for your DB cluster.	Aurora MySQL
BacktrackChangeRecordsUsed	The actual number of backtrack change records used by your DB cluster.	Aurora MySQL
BacktrackWindowDifference	The difference between the target backtrack window and the actual backtrack window.	Aurora MySQL
BacktrackWindowRatio	The number of times that the actual backtrack window is smaller than the target backtrack window for a given period of time.	Aurora MySQL
BackupRetentionRate	The total storage of backup storage in bytes used to support the point-in-time restore feature within the Aurora DB cluster's backup retention window. Included in the total reported by the <code>TotalBackupStorageBilled</code> metric. Computed separately for each Aurora cluster. For instructions, see Understanding Aurora Backup Storage Usage (p. 382) .	Aurora MySQL and Aurora PostgreSQL
BinLogDiskUsage	The amount of disk space occupied by binary logs on the master, in bytes.	Aurora MySQL
BlockedTransactions	The average number of transactions in the database that are blocked per second.	Aurora MySQL
BufferCacheHitRatio	The percentage of requests that are served by the buffer cache.	Aurora MySQL and Aurora PostgreSQL

Metric	Description	Applies to
CommitLatency	The amount of latency for commit operations, in milliseconds.	Aurora MySQL and Aurora PostgreSQL
CommitThroughput	The average number of commit operations per second.	Aurora MySQL and Aurora PostgreSQL
CPUCreditBalance	The number of CPU credits that an instance has accumulated. This metric applies only to db.t2.small and db.t2.medium instances. It is used to determine how long an Aurora MySQL DB instance can burst beyond its baseline performance level at a given rate. Note CPU credit metrics are reported at 5-minute intervals.	Aurora MySQL
CPUCreditUsage	The number of CPU credits consumed during the specified period. This metric applies only to db.t2.small and db.t2.medium instances. It identifies the amount of time during which physical CPUs have been used for processing instructions by virtual CPUs allocated to the Aurora MySQL DB instance. Note CPU credit metrics are reported at 5-minute intervals.	Aurora MySQL
CPUUtilization	The percentage of CPU used by an Aurora DB instance.	Aurora MySQL and Aurora PostgreSQL
DatabaseConnections	The number of connections to an Aurora DB instance.	Aurora MySQL and Aurora PostgreSQL
DDLLatency	The amount of latency for data definition language (DDL) requests, in milliseconds—for example, create, alter, and drop requests.	Aurora MySQL
DDLTroughput	The average number of DDL requests per second.	Aurora MySQL
Deadlocks	The average number of deadlocks in the database per second.	Aurora MySQL and Aurora PostgreSQL
DeleteLatency	The amount of latency for delete queries, in milliseconds.	Aurora MySQL
DeleteThroughput	The average number of delete queries per second.	Aurora MySQL
DiskQueueDepth	The number of outstanding read/write requests waiting to access the disk.	Aurora PostgreSQL
DMLLatency	The amount of latency for inserts, updates, and deletes, in milliseconds.	Aurora MySQL
DMLThroughput	The average number of inserts, updates, and deletes per second.	Aurora MySQL

Metric	Description	Applies to
EngineUptime	The amount of time that the instance has been running, in seconds.	Aurora MySQL and Aurora PostgreSQL
FreeableMemory	The amount of available random access memory, in bytes.	Aurora MySQL and Aurora PostgreSQL
FreeLocalStorage	The amount of local storage available, in bytes. Unlike for other DB engines, for Aurora DB instances this metric reports the amount of storage available to each DB instance. This value depends on the DB instance class (for pricing information, see the Amazon RDS product page). You can increase the amount of free storage space for an instance by choosing a larger DB instance class for your instance.	Aurora MySQL and Aurora PostgreSQL
InsertLatency	The amount of latency for insert queries, in milliseconds.	Aurora MySQL
InsertThroughput	The average number of insert queries per second.	Aurora MySQL
LoginFailures	The average number of failed login attempts per second.	Aurora MySQL
MaximumUsedTransactionId	The age of the oldest unvacuumed transaction ID, in transactions. If this value reaches 2,146,483,648 ($2^{31} - 1,000,000$), the database is forced into read-only mode, to avoid transaction ID wraparound. For more information, see Preventing Transaction ID Wraparound Failures in the PostgreSQL documentation.	Aurora PostgreSQL
NetworkReceivedThroughput	The amount of network throughput received from clients by each instance in the Aurora MySQL DB cluster, in bytes per second. This throughput doesn't include network traffic between instances in the Aurora DB cluster and the cluster volume.	Aurora MySQL and Aurora PostgreSQL
NetworkThroughput	The amount of network throughput both received from and transmitted to clients by each instance in the Aurora MySQL DB cluster, in bytes per second. This throughput doesn't include network traffic between instances in the DB cluster and the cluster volume.	Aurora MySQL and Aurora PostgreSQL
NetworkTransmittedThroughput	The amount of network throughput sent to clients by each instance in the Aurora DB cluster, in bytes per second. This throughput doesn't include network traffic between instances in the DB cluster and the cluster volume.	Aurora MySQL and Aurora PostgreSQL
Queries	The average number of queries executed per second.	Aurora MySQL

Metric	Description	Applies to
RDSToAuroraPostgreSQLLatency	The amount of time in seconds when replicating updates from the primary RDS PostgreSQL instance to other nodes in the cluster.	Aurora PostgreSQL
ReadIOPS	The average number of disk I/O operations per second. Aurora with PostgreSQL compatibility reports read and write IOPS separately, in 1-minute intervals.	Aurora PostgreSQL
ReadLatency	The average amount of time taken per disk I/O operation.	Aurora PostgreSQL
ReadThroughput	The average number of bytes read from disk per second.	Aurora PostgreSQL
ResultSetCacheHitPercentage	The percentage of requests that are served by the Resultset cache.	Aurora MySQL
SelectLatency	The amount of latency for select queries, in milliseconds.	Aurora MySQL
SelectThroughput	The average number of select queries per second.	Aurora MySQL
SnapshotStorageUsed	The total amount of backup storage in bytes consumed by all Aurora snapshots for an Aurora DB cluster outside its backup retention window. Included in the total reported by the TotalBackupStorageBilled metric. Computed separately for each Aurora cluster. For instructions, see Understanding Aurora Backup Storage Usage (p. 382) .	Aurora MySQL and Aurora PostgreSQL
SwapUsage	The amount of swap space used on the Aurora PostgreSQL DB instance.	Aurora PostgreSQL
TotalBackupStorageBilled	The total amount of backup storage in bytes for which you are billed for a given Aurora DB cluster. Includes the backup storage measured by the BackupRetentionPeriodStorageUsed and SnapshotStorageUsed metrics. Computed separately for each Aurora cluster. For instructions, see Understanding Aurora Backup Storage Usage (p. 382) .	Aurora MySQL and Aurora PostgreSQL
TransactionLogDiskUsage	The amount of disk space occupied by transaction logs on the Aurora PostgreSQL DB instance.	Aurora PostgreSQL
UpdateLatency	The amount of latency for update queries, in milliseconds.	Aurora MySQL
UpdateThroughput	The average number of update queries per second.	Aurora MySQL

Metric	Description	Applies to
AuroraVolumeBytesLeftTotal	The remaining available space for the cluster volume, measured in bytes. As the cluster volume grows, this value decreases. If it reaches zero, the cluster reports an out-of-space error. This value is simpler and more reliable to monitor than <code>VolumeBytesUsed</code> when you want to detect whether your Aurora cluster is approaching the 64 TiB size limit. <code>AuroraVolumeBytesLeftTotal</code> takes into account storage used for internal housekeeping and other allocations that don't affect your storage billing.	Aurora MySQL
VolumeBytesUsed	<p>The amount of storage used by your Aurora DB instance, in bytes.</p> <p>This value affects the cost of the Aurora DB cluster (for pricing information, see the Amazon RDS product page).</p> <p>This value doesn't reflect some internal storage allocations that don't affect storage billing. Thus, you can anticipate out-of-space issues more accurately by testing whether <code>AuroraVolumeBytesLeftTotal</code> is approaching zero instead of comparing <code>VolumeBytesUsed</code> against the 64 TiB storage limit.</p>	Aurora MySQL and Aurora PostgreSQL
VolumeReadIOPS	<p>The number of billed read I/O operations from a cluster volume, reported at 5-minute intervals.</p> <p>Billed read operations are calculated at the cluster volume level, aggregated from all instances in the Aurora DB cluster, and then reported at 5-minute intervals. The value is calculated by taking the value of the Read operations metric over a 5-minute period. You can determine the amount of billed read operations per second by taking the value of the Billed read operations metric and dividing by 300 seconds. For example, if the Billed read operations returns 13,686, then the billed read operations per second is 45 ($13,686 / 300 = 45.62$).</p> <p>You accrue billed read operations for queries that request database pages that aren't in the buffer cache and therefore 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>	Aurora MySQL and Aurora PostgreSQL
VolumeWriteIOPS	The number of write disk I/O operations to the cluster volume, reported at 5-minute intervals. See the description of <code>VolumeReadIOPS</code> above for a detailed description of how billed write operations are calculated.	Aurora MySQL and Aurora PostgreSQL

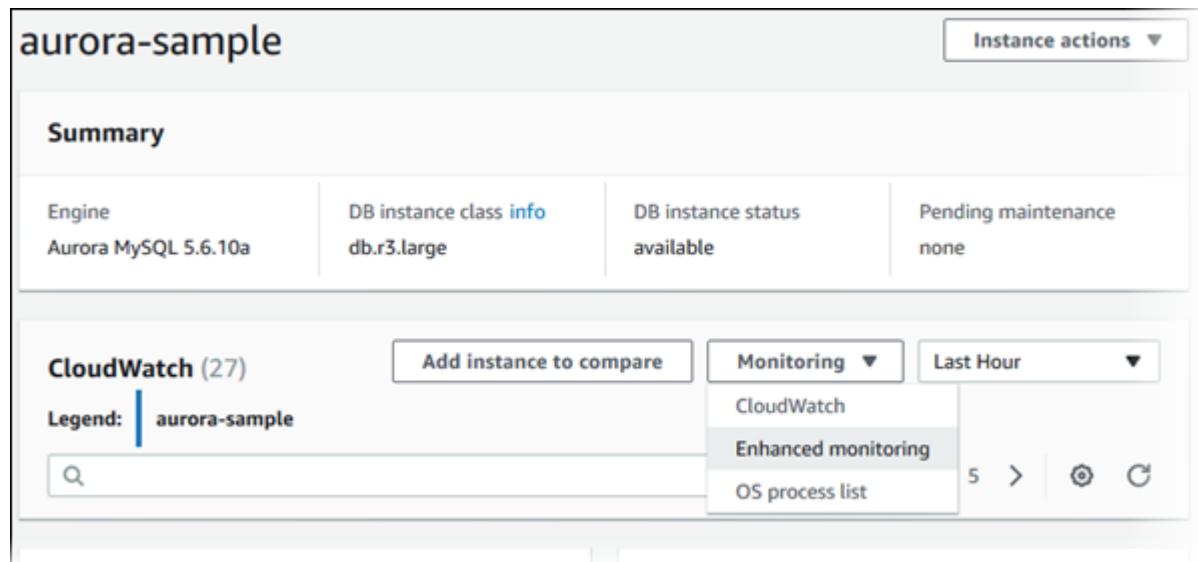
Metric	Description	Applies to
WriteIOPS	The average number of disk I/O operations per second. Aurora PostgreSQL reports read and write IOPS separately, on 1-minute intervals.	Aurora PostgreSQL
WriteLatency	The average amount of time taken per disk I/O operation.	Aurora PostgreSQL
WriteThroughput	The average number of bytes written to disk per second.	Aurora PostgreSQL

Viewing Aurora Metrics in the Amazon RDS Console

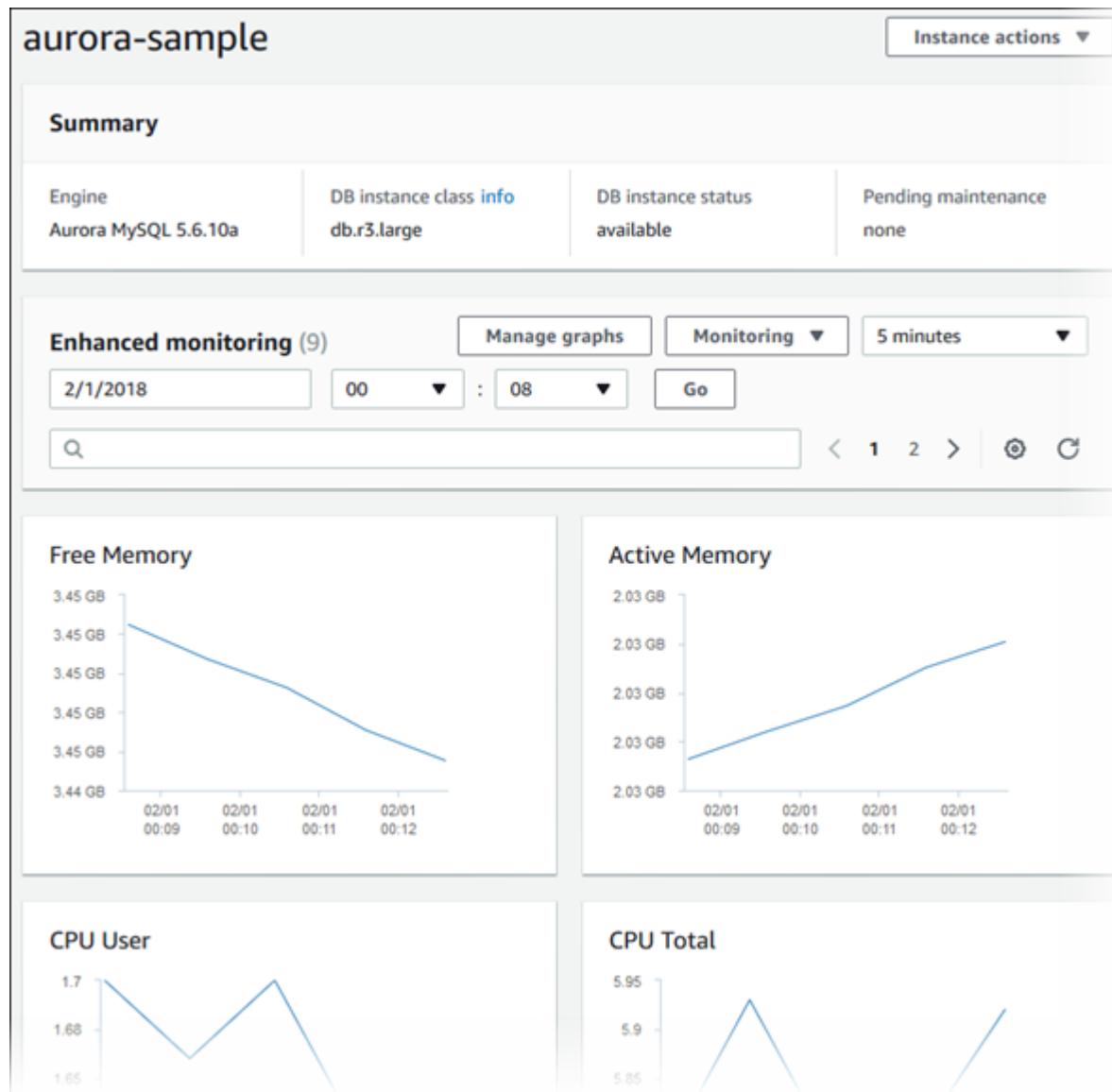
To monitor the health and performance of your Aurora DB cluster, you can view some, but not all, of the metrics provided by Amazon Aurora in the Amazon RDS console. For a detailed list of Aurora metrics available to the Amazon RDS console, see [Aurora Metrics Available in the Amazon RDS Console \(p. 465\)](#).

To view Aurora metrics 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 **Instances**.
3. Choose the name of the DB instance you want to monitor to see its details.
4. In the Cloudwatch section, choose one of the following options from **Monitoring** for how you want to view your metrics:
 - **Cloudwatch** – Shows a summary of CloudWatch metrics. Each metric includes a graph showing the metric monitored over a specific time span. For more information, see [Monitoring with Amazon CloudWatch \(p. 436\)](#).
 - **Enhanced monitoring** – Shows a summary of OS metrics available to an Aurora DB instance with Enhanced Monitoring enabled. Each metric includes a graph showing the metric monitored over a specific time span. For more information, see [Enhanced Monitoring \(p. 469\)](#).
 - **OS process list** – Shows the processes running on the DB instance or DB cluster and their related metrics including CPU percentage, memory usage, and so on.



5. The following image shows the metrics view with **Enhanced monitoring** selected.



Aurora Metrics Available in the Amazon RDS Console

Not all of the metrics provided by Amazon Aurora are available to you in the Amazon RDS console. You can view them using other tools, however, such as the AWS CLI and CloudWatch API. In addition, some of the metrics that are available in the Amazon RDS console are either shown only for specific instance classes, or with different names and different units of measurement.

The following Aurora metrics are not available in the Amazon RDS console:

- `AuroraBinlogReplicaLag`
- `DeleteLatency`
- `DeleteThroughput`
- `EngineUptime`
- `InsertLatency`

- InsertThroughput
- NetworkThroughput
- Queries
- UpdateLatency
- UpdateThroughput

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 db.t2.small and db.t2.medium instances
- The following metrics that are displayed with different names, as listed:

Metric	Display Name
AuroraReplicaLagMax	Replica lag maximum
AuroraReplicaLagMin	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

Latest Metrics View

You can view a subset of categorized Aurora metrics in the Latest Metrics view of 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

Category	Metrics
	CommitLatency CommitThroughput DatabaseConnections DDLLatency DDLThroughput Deadlocks DMLLatency DMLThroughput LoginFailures ResultSetCacheHitRatio SelectLatency SelectThroughput
System	AuroraReplicaLag AuroraReplicaLagMaximum AuroraReplicaLagMinimum CPUCreditBalance CPUCreditUsage CPUUtilization FreeableMemory FreeLocalStorage NetworkReceiveThroughput
Deployment	AuroraReplicaLag BufferCacheHitRatio ResultSetCacheHitRatio SelectThroughput

Note

The **Failed SQL Statements** metric, displayed under the **SQL** category of the Latest Metrics view in the Amazon RDS console, does not apply to Amazon Aurora.

Related Topics

- [Managing an Amazon Aurora DB Cluster \(p. 260\)](#)

Enhanced Monitoring

Amazon RDS provides metrics in real time for the operating system (OS) that your DB instance runs on. You can view the metrics for your DB instance using the console, or consume the Enhanced Monitoring JSON output from Amazon CloudWatch Logs in a monitoring system of your choice.

By default, Enhanced Monitoring metrics are stored in the CloudWatch Logs for 30 days. 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*.

The cost for using Enhanced Monitoring depends on several factors:

- You are only charged for Enhanced Monitoring that exceeds the free tier provided by Amazon CloudWatch Logs.

For more information about pricing, see [Amazon CloudWatch Pricing](#).
- A smaller monitoring interval results in more frequent reporting of OS metrics and increases your monitoring cost.
- 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.

Differences Between CloudWatch and Enhanced Monitoring Metrics

CloudWatch gathers metrics about CPU utilization from the hypervisor for a DB instance, and Enhanced Monitoring gathers its metrics from an agent on the instance. As a result, you might find differences between the measurements, because the hypervisor layer performs a small amount of work. The differences can be greater if your DB instances use smaller instance classes, because then there are likely more virtual machines (VMs) that are managed by the hypervisor layer on a single physical instance. Enhanced Monitoring metrics are useful when you want to see how different processes or threads on a DB instance use the CPU.

Setting Up for and Enabling Enhanced Monitoring

To set up for and enable Enhanced Monitoring, take the steps listed following.

Before You Begin

Enhanced Monitoring requires permission to act on your behalf to send OS metric information to CloudWatch Logs. You grant Enhanced Monitoring the required permissions using an AWS Identity and Access Management (IAM) role.

The first time that you enable Enhanced Monitoring in the console, you can select the **Default** option for the **Monitoring Role** property to have RDS create the required IAM role. RDS then automatically creates a role named `rds-monitoring-role` for you, and uses it for the specified DB instance or Read Replica.

You can also create the required role before you enable Enhanced Monitoring, and then specify your new role's name when you enable Enhanced Monitoring. You must create this required role if you enable Enhanced Monitoring using the AWS CLI or the RDS API.

To create the appropriate IAM role to permit Amazon RDS to communicate with the Amazon CloudWatch Logs service on your behalf, take the following steps.

The user that enables Enhanced Monitoring must be granted the `PassRole` permission. For more information, see Example 2 in [Granting a User Permissions to Pass a Role to an AWS Service](#) in the *IAM User Guide*.

To create an IAM role for Amazon RDS Enhanced Monitoring

1. Open the [IAM Console](#) at <https://console.aws.amazon.com>.
2. In the navigation pane, choose **Roles**.
3. Choose **Create role**.
4. Choose the **AWS service** tab, and then choose **RDS** from the list of services.
5. Choose **RDS - Enhanced Monitoring**, and then choose **Next: Permissions**.
6. Ensure that the **Attached permissions policy** page shows **AmazonRDSEnhancedMonitoringRole**, and then choose **Next: Tags**.
7. On the **Add tags** page, choose **Next: Review**.
8. For **Role Name**, enter a name for your role, for example **emaccess**, and then choose **Create role**.

Enabling and Disabling Enhanced Monitoring

You can enable Enhanced Monitoring when you create a DB cluster or Read Replica, or when you modify a DB cluster. If you modify a DB instance to enable Enhanced Monitoring, you don't need to reboot your DB instance for the change to take effect.

You can enable Enhanced Monitoring in the RDS console when you do one of the following actions:

- **Create a DB Cluster** – You can enable Enhanced Monitoring in the [Configure Advanced Settings](#) page.
- **Create a Read Replica** – You can enable Enhanced Monitoring in the [Configure Advanced Settings](#) page.
- **Modify a DB Instance** – You can enable Enhanced Monitoring in the [Modify DB Instance](#) page.

To enable Enhanced Monitoring by using the RDS console, scroll to the **Monitoring** section and do the following:

1. Choose **Enable enhanced monitoring** for your DB instance or Read Replica.
2. 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`.
3. 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.

To disable Enhanced Monitoring, choose **Disable enhanced monitoring**.

Monitoring

Enhanced monitoring

Enable enhanced monitoring
Enhanced monitoring metrics are useful when you want to see how different processes or threads use the CPU.

Disable enhanced monitoring

Monitoring Role	Granularity
rds-monitoring-role	60 seco... ▾

Enabling Enhanced Monitoring doesn't require your DB instance to restart.

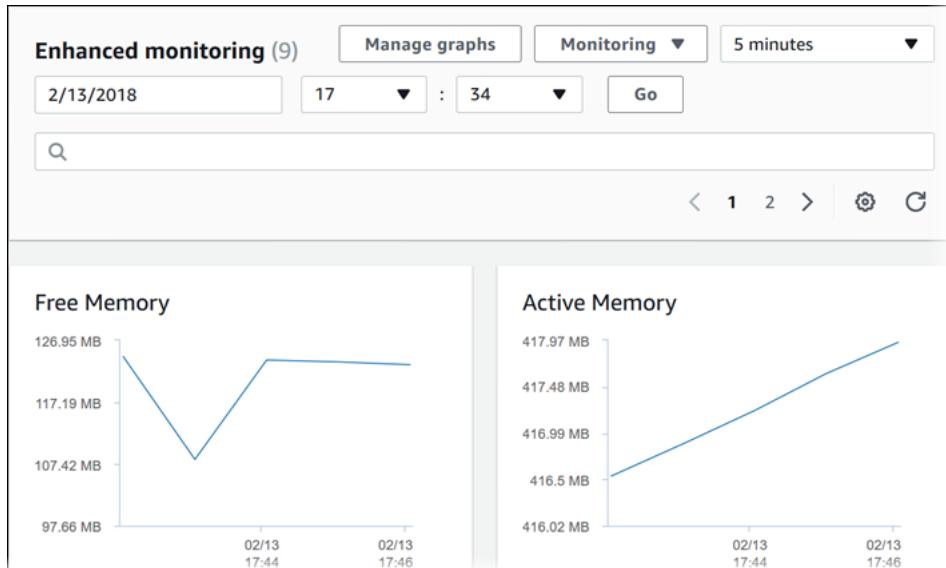
Note

The fastest that the RDS console refreshes is every 5 seconds. If you set the granularity to 1 second in the RDS console, you still see updated metrics only every 5 seconds. You can retrieve 1-second metric updates by using CloudWatch Logs.

Viewing Enhanced Monitoring

You can view OS metrics reported by Enhanced Monitoring in the RDS console by choosing **Enhanced monitoring for Monitoring**.

The Enhanced Monitoring page is shown following.



If you want to see details for the processes running on your DB instance, choose **OS process list for Monitoring**.

The **Process List** view is shown following.

NAME	VIRT	RES	CPU%	MEM%	VMLIMIT
postgres [3181] ^t	283.55 MB	17.11 MB	0.02	1.72	
postgres: rdsadmin	384.7 MB	9.51 MB	0.02	0.95	

The Enhanced Monitoring metrics shown in the **Process list** view are organized as follows:

- **RDS child processes** – Shows a summary of the RDS processes that support the DB instance, for example `aurora` for Amazon Aurora DB clusters. Process threads appear nested beneath the parent process. Process threads show CPU utilization only as other metrics are the same for all threads for the process. The console displays a maximum of 100 processes and threads. The results are a combination of the top CPU consuming and memory consuming processes and threads. If there are more than 50 processes and more than 50 threads, the console displays the top 50 consumers in each category. This display helps you identify which processes are having the greatest impact on performance.
- **RDS processes** – Shows a summary of the resources used by the RDS management agent, diagnostics monitoring processes, and other AWS processes that are required to support RDS DB instances.
- **OS processes** – Shows a summary of the kernel and system processes, which generally have minimal impact on performance.

The items listed for each process are:

- **VIRT** – Displays the virtual size of the process.
- **RES** – Displays the actual physical memory being used by the process.
- **CPU%** – Displays the percentage of the CPU bandwidth consumed by the process.
- **MEM%** – Displays the percentage of the total memory consumed 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 Enhanced Monitoring by Using CloudWatch Logs \(p. 473\)](#).

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 Enhanced Monitoring by Using CloudWatch Logs

After you have enabled Enhanced Monitoring for your DB instance, you can view the metrics for your DB instance using CloudWatch Logs, with each log stream representing a single DB instance being monitored. The log stream identifier is the resource identifier (`DbiResourceId`) for the DB instance.

To view Enhanced Monitoring log data

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. If necessary, choose the region that your DB instance is in. For more information, see [Regions and Endpoints](#) in the *Amazon Web Services General Reference*.
3. Choose **Logs** in the navigation pane.
4. Choose **RDSOSMetrics** from the list of log groups.
5. Choose the log stream that you want to view from the list of log streams.

Available OS Metrics

The following tables list the OS metrics available using Amazon CloudWatch Logs.

Metrics for Aurora

Group	Metrics	Description
General	engine	The database engine for the DB instance.
	instanceID	The DB instance identifier.
	instanceResourceIdentifier	A region-unique, immutable identifier for the DB instance, also used as the log stream identifier.
	numVCpus	The number of virtual CPUs for the DB instance.
	timestamp	The time at which the metrics were taken.
	uptime	The amount of time that the DB instance has been active.
	version	The version of the OS metrics' stream JSON format.
cpuUtilization	guest	The percentage of CPU in use by guest programs.
	idle	The percentage of CPU that is idle.
	irq	The percentage of CPU in use by software interrupts.
	nice	The percentage of CPU in use by programs running at lowest priority.
	steal	The percentage of CPU in use by other virtual machines.
	system	The percentage of CPU in use by the kernel.
	total	The total percentage of the CPU in use. This value includes the nice value.
	user	The percentage of CPU in use by user programs.

Group	Metrics	Description
	wait	The percentage of CPU unused while waiting for I/O access.
fileSys	maxFiles	The maximum number of files that can be created for the file system.
	mountPoint	The path to the file system.
	name	The name of the file system.
	total	The total number of disk space available for the file system, in kilobytes.
	used	The amount of disk space used by files in the file system, in kilobytes.
	usedFilePercent	The percentage of available files in use.
	usedFiles	The number of files in the file system.
	usedPercent	The percentage of the file-system disk space in use.
loadAverageMinutes	fifteen	The number of processes requesting CPU time over the last 15 minutes.
	five	The number of processes requesting CPU time over the last 5 minutes.
	one	The number of processes requesting CPU time over the last minute.
memory	active	The amount of assigned memory, in kilobytes.
	buffers	The amount of memory used for buffering I/O requests prior to writing to the storage device, in kilobytes.
	cached	The amount of memory used for caching file system-based I/O.
	dirty	The amount of memory pages in RAM that have been modified but not written to their related data block in storage, in kilobytes.
	free	The amount of unassigned memory, in kilobytes.
	hugePagesFree	The number of free huge pages. Huge pages are a feature of the Linux kernel.
	hugePagesRsvd	The number of committed huge pages.
	hugePagesSize	The size for each huge pages unit, in kilobytes.
	hugePagesSurp	The number of available surplus huge pages over the total.
	hugePagesTotal	The total number of huge pages for the system.
	inactive	The amount of least-frequently used memory pages, in kilobytes.
	mapped	The total amount of file-system contents that is memory mapped inside a process address space, in kilobytes.

Group	Metrics	Description
	pageTables	The amount of memory used by page tables, in kilobytes.
	slab	The amount of reusable kernel data structures, in kilobytes.
	total	The total amount of memory, in kilobytes.
	writeback	The amount of dirty pages in RAM that are still being written to the backing storage, in kilobytes.
network	interface	The identifier for the network interface being used for the DB instance.
	rx	The number of bytes received per second.
	tx	The number of bytes uploaded per second.
processList	cpuUsedPc	The percentage of CPU used by the process.
	id	The identifier of the process.
	memoryUsedPc	The amount of memory used by the process, in kilobytes.
	name	The name of the process.
	parentID	The process identifier for the parent process of the process.
	rss	The amount of RAM allocated to the process, in kilobytes.
	tgid	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.
	VIRT	The amount of virtual memory allocated to the process, in kilobytes.
swap	swap	The amount of swap memory available, in kilobytes.
	swap_in	The amount of memory, in kilobytes, swapped in from disk.
	swap_out	The amount of memory, in kilobytes, swapped out to disk.
	free	The amount of swap memory free, in kilobytes.
	committed	The amount of swap memory, in kilobytes, used as cache memory.
tasks	blocked	The number of tasks that are blocked.
	running	The number of tasks that are running.
	sleeping	The number of tasks that are sleeping.
	stopped	The number of tasks that are stopped.
	total	The total number of tasks.
	zombie	The number of child tasks that are inactive with an active parent task.

Using Amazon RDS Performance Insights

Amazon RDS Performance Insights monitors your Amazon RDS DB instance load so that you can analyze and troubleshoot your database performance. Amazon RDS Performance Insights is currently available for use with the following DB engines:

- Amazon Aurora with MySQL compatibility version 2.04.2 and higher 2.x versions (compatible with MySQL 5.7)
- Amazon Aurora with MySQL compatibility version 1.17.3 and higher 1.x versions (compatible with MySQL 5.6)
- Amazon Aurora with PostgreSQL compatibility
- Amazon RDS for MariaDB version 10.2.21 and higher 10.2 versions
- Amazon RDS for MySQL version 5.7.22 and higher 5.7 versions, and version 5.6.41 and higher 5.6 versions
- Amazon RDS for Microsoft SQL Server (all versions except SQL Server 2008)
- Amazon RDS for PostgreSQL version 10 and 11
- Amazon RDS for Oracle (all versions)

Note

Amazon RDS Performance Insights is not supported for MariaDB version 10.0, 10.1, or 10.3, or for MySQL version 5.5 or 8.0.

For Amazon RDS for MariaDB and MySQL, Performance Insights is not supported on the following DB instance classes: db.t2.micro, db.t2.small, db.t3.micro, and db.t3.small.

On Aurora MySQL, Performance Insights is not supported on db.t2 or db.t3 DB instance classes. Performance Insights is not supported for Aurora MySQL DB clusters enabled for parallel query.

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.

Performance Insights is on by default in the console create wizard for all DB engines that work with Amazon RDS. If you have more than one database on a DB instance, performance data for all of the databases is aggregated for the DB instance.

The central metric for Performance Insights is `DB_Load`, which represents the average number of active sessions for the DB engine. The `DB_Load` metric is collected every second. An *active session* is a connection that has submitted work to the DB engine and is waiting for a response from it. For example, if you submit a SQL query to the DB engine, the database session is active while the DB engine is processing that query.

By combining `DB_Load` with wait event data, you can get a complete picture of the state for an active session. A *wait event* is a condition that causes the execution of a SQL statement to wait for a specific event to happen before it can continue. For example, SQL statement execution might wait until a locked resource is unlocked. Wait events vary by DB engine:

- For a list of the most commonly used wait events for Aurora MySQL, see [Aurora MySQL Events \(p. 785\)](#).
- For information about all MySQL wait events, see [Wait Event Summary Tables](#) in the MySQL documentation.
- For a list of the most commonly used wait events for Aurora PostgreSQL, see [Amazon Aurora PostgreSQL Events \(p. 969\)](#).
- For information about all PostgreSQL wait events, see [PostgreSQL Wait Events](#) in the PostgreSQL documentation.

Session information is collected, aggregated, and displayed in the dashboard as the **Average Active Sessions** chart. The **Average Active Sessions** chart displays the **Max CPU** value as a line, so you can see if active sessions are exceeding it or not. The **Max CPU** value is determined by the number of **vCPU** (virtual CPU) cores for your DB instance.

If the load in the **Average Active Sessions** chart is often above the **Max CPU** line and the primary wait state is CPU, the system CPU is overloaded. In these cases, 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 load doesn't cross the **Max CPU** line.

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

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

Topics

- [Enabling Performance Insights \(p. 477\)](#)
- [Access Control for Performance Insights \(p. 481\)](#)
- [Using the Performance Insights Dashboard \(p. 482\)](#)
- [Additional User Interface Features \(p. 495\)](#)
- [Performance Insights API \(p. 496\)](#)
- [Performance Insights Metrics Published to Amazon CloudWatch \(p. 509\)](#)
- [Performance Insights Counters \(p. 510\)](#)
- [Logging Performance Insights Calls by Using AWS CloudTrail \(p. 517\)](#)

Enabling Performance Insights

To use Performance Insights, you must enable it on your DB instance.

If you use Performance Insights together with Aurora Global Database, you must enable Performance Insights individually for the DB instances in each AWS Region. For details, see [Performance Insights for Aurora Global Database \(p. 45\)](#).

Console

You can use the console to enable Performance Insights when you create a new DB instance. You can also modify a DB instance to enable Performance Insights.

Topics

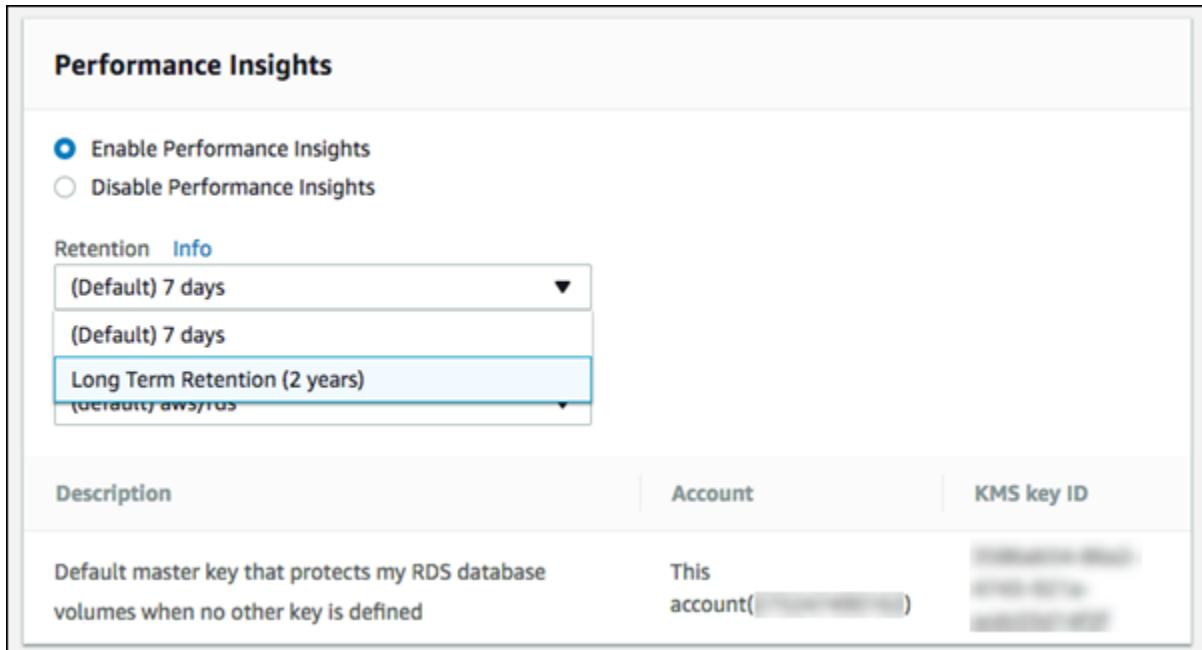
- [Enabling Performance Insights with the Console When Creating a DB Instance \(p. 477\)](#)
- [Enabling Performance Insights with the Console When Modifying a DB Instance \(p. 478\)](#)

[Enabling Performance Insights with the Console When Creating a DB Instance](#)

When you create a new DB instance, Performance Insights is enabled when you choose **Enable Performance Insights** in the **Performance Insights** section.

To create a DB instance, follow the instructions for your DB engine in [Creating an Amazon Aurora DB Cluster \(p. 100\)](#).

The following screenshot shows the **Performance Insights** section.



You have the following options when you choose **Enable Performance Insights**:

- **Retention** – The amount of time to retain Performance Insights data. Choose either 7 days (the default) or 2 years.
- **Master key** – Specify your AWS Key Management Service (AWS KMS) key. Performance Insights encrypts all potentially sensitive data using your AWS KMS key. Data is encrypted in flight and at rest. For more information, see [Encrypting Amazon Aurora Resources \(p. 175\)](#).

[Enabling Performance Insights with the Console When Modifying a DB Instance](#)

You can modify a DB instance to enable Performance Insights using the console.

To enable Performance Insights for a DB instance using the console

1. Sign in to the AWS Management Console and open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. Choose **Databases**.
3. Choose the DB instance that you want to modify, and choose **Modify**.
4. In the **Performance Insights** section, choose **Enable Performance Insights**.

You have the following options when you choose **Enable Performance Insights**:

- **Retention** – The amount of time to retain Performance Insights data. Choose either 7 days (the default) or 2 years.
 - **Master key** – Specify your AWS Key Management Service (AWS KMS) key. Performance Insights encrypts all potentially sensitive data using your AWS KMS key. Data is encrypted in flight and at rest. For more information, see [Encrypting Amazon Aurora Resources \(p. 175\)](#).
5. Choose **Continue**.
 6. For **Scheduling of Modifications**, choose one of the following:
 - **Apply during the next scheduled maintenance window** – Wait to apply the **Performance Insights** modification until the next maintenance window.

- **Apply immediately** – Apply the **Performance Insights** modification as soon as possible.
7. Choose **Modify instance**.

AWS CLI

When you create a new DB instance using the [create-db-instance](#) AWS CLI command, Performance Insights is enabled when you specify `--enable-performance-insights`.

You can also specify the `--enable-performance-insights` value using the following AWS CLI commands:

- [create-db-instance-read-replica](#)
- [modify-db-instance](#)
- [restore-db-instance-from-s3](#)

The following procedure describes how to enable Performance Insights for a DB instance using the AWS CLI.

To enable Performance Insights for a DB instance using the AWS CLI

- Call the [modify-db-instance](#) AWS CLI command and supply the following values:
 - `--db-instance-identifier` – The name of the DB instance.
 - `--enable-performance-insights`

The following example enables Performance Insights for `sample-db-instance`.

For Linux, OS X, or Unix:

```
aws rds modify-db-instance \
  --db-instance-identifier sample-db-instance \
  --enable-performance-insights
```

For Windows:

```
aws rds modify-db-instance ^
  --db-instance-identifier sample-db-instance ^
  --enable-performance-insights
```

When you enable Performance Insights, you can optionally specify the amount of time, in days, to retain Performance Insights data with the `--performance-insights-retention-period` option. Valid values are 7 (the default) or 731 (2 years).

The following example enables Performance Insights for `sample-db-instance` and specifies that Performance Insights data is retained for two years.

For Linux, OS X, or Unix:

```
aws rds modify-db-instance \
  --db-instance-identifier sample-db-instance \
```

```
--enable-performance-insights \
--performance-insights-retention-period 731
```

For Windows:

```
aws rds modify-db-instance ^
--db-instance-identifier sample-db-instance ^
--enable-performance-insights ^
--performance-insights-retention-period 731
```

RDS API

When you create a new DB instance using the [CreateDBInstance](#) operation Amazon RDS API operation, Performance Insights is enabled when you set `EnablePerformanceInsights` to `True`.

You can also specify the `EnablePerformanceInsights` value using the following API operations:

- [ModifyDBInstance](#)
- [CreateDBInstanceReadReplica](#)
- [RestoreDBInstanceFromS3](#)

When you enable Performance Insights, you can optionally specify the amount of time, in days, to retain Performance Insights data with the `PerformanceInsightsRetentionPeriod` parameter. Valid values are 7 (the default) or 731 (2 years).

Enabling the Performance Schema for Performance Insights on Aurora MySQL

For Aurora MySQL, Performance Insights provides more detailed information when the Performance Schema feature is enabled. The Performance Schema is enabled automatically when you create an Aurora MySQL DB instance with Performance Insights enabled. When you create the DB instance with Performance Insights enabled, the following subset of Performance Schema parameters is set to the specified values automatically:

Parameter Name	Parameter Value
<code>performance_schema</code>	1
<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

Performance Schema is enabled automatically only if your parameter group doesn't have an explicitly set value for the `performance_schema` parameter. You can examine the `performance_schema` parameter, and if the value of source is `user`, then you set a value. If you want the Performance Schema parameters to be set automatically, then unset the value for the `performance_schema` parameter. You

can view the source of a parameter value by viewing the parameter in the AWS Management Console or by running the AWS CLI [describe-db-parameters](#) command.

When you change the value of the `performance_schema` parameter, a DB instance reboot is required. If you're creating a new DB instance with Performance Insights enabled, the `performance_schema` parameter is set to 1 (enabled) by default.

Without the Performance Schema enabled, Performance Insights displays database load broken down by the list state of the MySQL process. With Performance Schema enabled, Performance Insights displays database load broken down by detailed wait events.

For more information, see [Using the Performance Insights Dashboard \(p. 482\)](#).

Access Control for Performance Insights

To access Performance Insights, you must have the appropriate permissions from AWS Identity and Access Management (IAM). There are two options available for granting access:

1. Attach the `AmazonRDSFullAccess` managed policy to an IAM user or role.
2. Create a custom IAM policy and attach it to an IAM user or role.

AmazonRDSFullAccess Managed Policy

`AmazonRDSFullAccess` is an AWS-managed policy that grants access to all of the Amazon RDS API operations. The policy also grants access to related services that are used by the Amazon RDS console—for example, event notifications using Amazon SNS.

In addition, `AmazonRDSFullAccess` contains all the permissions needed for using Performance Insights. If you attach this policy to an IAM user or role, the recipient can use Performance Insights, along with other console features.

Using a Custom IAM Policy

For users who don't have full access with the `AmazonRDSFullAccess` policy, you can grant access to Performance Insights by creating or modifying a user-managed IAM policy. When you attach the policy to an IAM user or role, the recipient can use Performance Insights.

To create a custom policy

1. Open the IAM console at <https://console.aws.amazon.com/iam/>.
2. In the navigation pane, choose **Policies**.
3. Choose **Create policy**.
4. On the **Create Policy** page, choose the JSON tab.
5. Copy and paste the following.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": "pi:*",  
            "Resource": "arn:aws:pi:*:metrics/rds/*"  
        }  
    ]  
}
```

6. Choose **Review policy**

Note

Currently, when you enter this policy, the **Visual editor** tab displays a warning that the **pi** resource is not recognized. You can ignore this warning.

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.

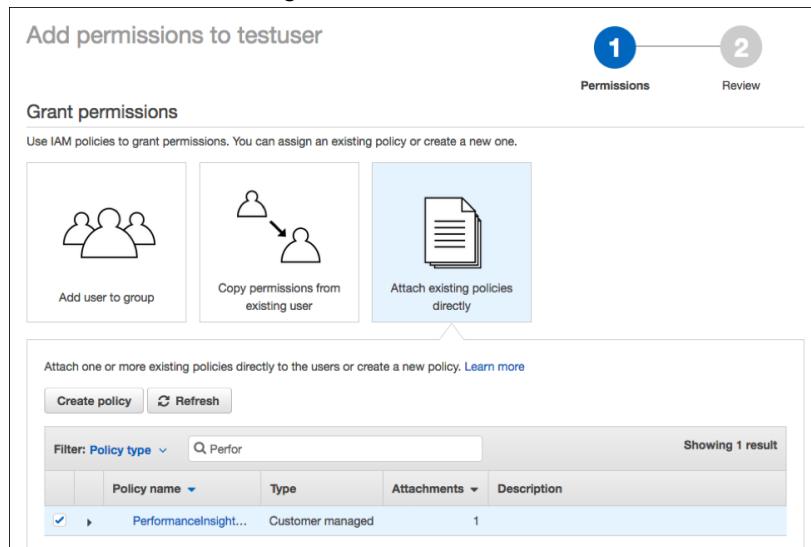
To attach the policy to an IAM user

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, the user must have access to Amazon RDS in addition to the custom policy. For example, the `AmazonRDSReadOnlyAccess` predefined policy provides read-only access to Amazon RDS. For more information, see [Managing Access Using Policies \(p. 193\)](#).

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**.

Using 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 also drill into details for a particular wait state, SQL query, host, or user.

Topics

- [Opening the Performance Insights Dashboard \(p. 483\)](#)
- [Performance Insights Dashboard Components \(p. 485\)](#)

- [Analyzing Database Load Using the Performance Insights Dashboard \(p. 489\)](#)
- [Analyzing Statistics for Running Queries \(p. 490\)](#)
- [Viewing More SQL Text in the Performance Insights Dashboard \(p. 493\)](#)

Opening the Performance Insights Dashboard

To see the Performance Insights dashboard, use the following procedure.

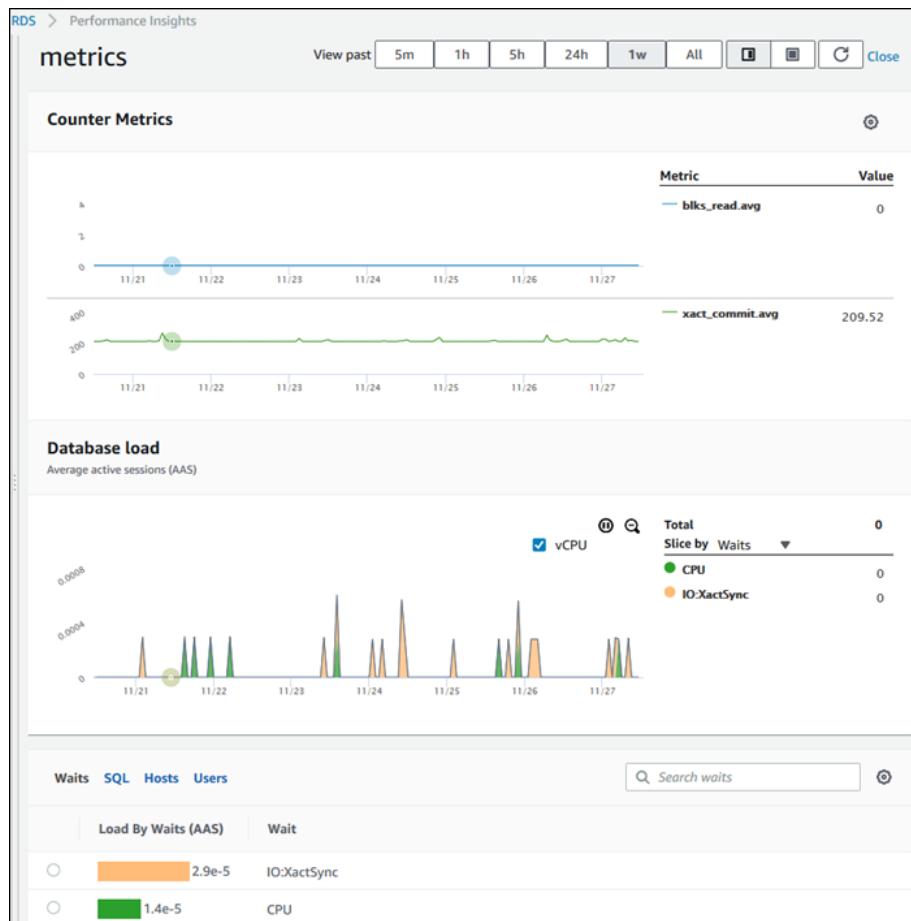
To view the Performance Insights dashboard in the AWS Management Console

1. Open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. In the navigation pane, choose **Performance Insights**.
3. Choose a DB instance. The Performance Insights dashboard is displayed for that DB instance.

For DB instances with Performance Insights enabled, you can also reach the dashboard by choosing the **Sessions** item in the list of DB instances. Under **Current activity**, the **Sessions** item shows the database load in average active sessions over the last five minutes. The bar graphically shows the load. When the bar is empty, the DB instance is idle. As the load increases, the bar fills with blue. When the load passes the number of virtual CPUs (vCPUs) on the DB instance class, the bar turns red, indicating a potential bottleneck.

Instances (11)						
DB instance	Engine	Status	CPU	Current activity	Mail	...
Aurora MySQL	available	1.23%	2 Selects/Sec	0 Sessions	none	
Oracle Enterprise Edition	available	1.33%	0 Connections	0 Sessions	requi	
Aurora PostgreSQL	available	23.25%	0.31 Sessions	0 Sessions	none	
Aurora MySQL	available	44.58%	3.86 Sessions	0 Sessions	none	
Aurora MySQL	available	51.93%	15.05 Sessions	0 Sessions	none	
MySQL	available	0.37%	0 Sessions	0 Sessions	none	
MySQL	available	0.33%	0 Sessions	0 Sessions	none	
MySQL	available	0.38%	0 Sessions	0 Sessions	none	
MySQL	available	0.83%	0 Connections	0 Sessions	none	
PostgreSQL	available	2.50%	0 Connections	0 Sessions	none	
PostgreSQL	available	1.83%	0 Connections	0 Sessions	none	

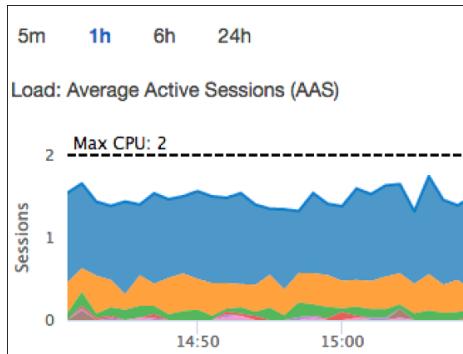
The following screenshot shows the dashboard for a DB instance.



By default, the Performance Insights dashboard shows data for the last 60 minutes. You can modify it to display data for the last 5 minutes, 60 minutes, 5 hours, 24 hours, or 1 week. You can also show all of the data available.

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 and 5 hours both refresh every minute.
- 24 hours refreshes every 5 minutes.
- 1 week refreshes every hour.



Performance Insights Dashboard Components

The dashboard is divided into three parts:

1. **Counter Metrics chart** – Shows data for specific performance counter metrics.
2. **Average Active Sessions chart** – Shows how the database load compares to DB instance capacity as represented by the **Max CPU** line.
3. **Top load items table** – Shows the top items contributing to database load.

Counter Metrics Chart

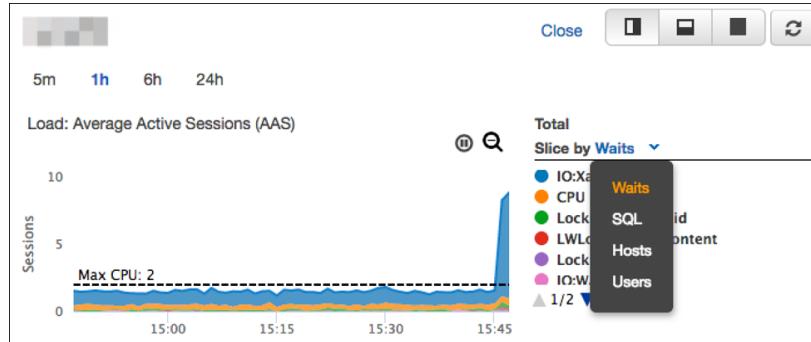
The **Counter Metrics** chart displays data for performance counters. The default metrics shown are `blk.read.avg` and `xact.commit.avg`. Choose which performance counters to display by selecting the gear icon in the upper-right corner of the chart.



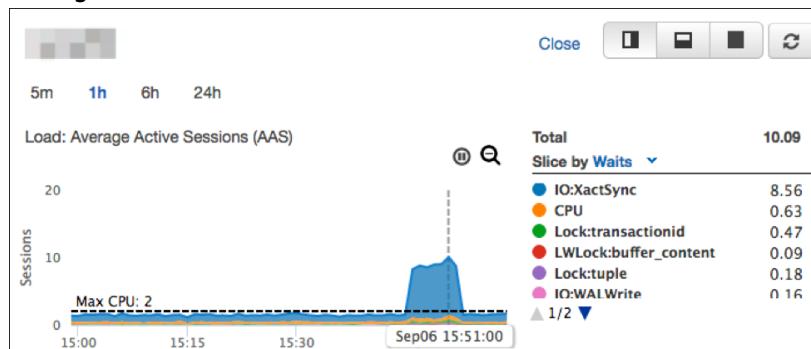
For more information, see [Performance Insights Counters \(p. 510\)](#).

Average Active Sessions Chart

The **Average Active Sessions** chart shows how the database load compares to DB instance capacity as represented by the **Max CPU** line. By default, load is shown as active sessions grouped by wait states. You can also choose instead to display load as active sessions grouped by SQL queries, hosts, or users.



To see details for any item for the selected time period in the legend, hover over that item on the **Average Active Sessions** chart.



Top Load Items Table

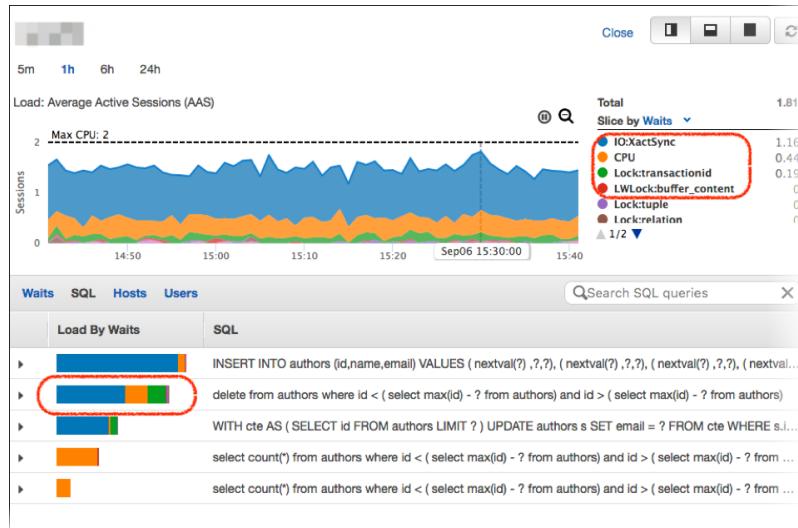
The **Top Load Items** table shows the top items contributing to database load. By default, the top SQL queries that are contributing to the database load are shown. Queries are displayed as digests of multiple actual queries that are structurally similar but that possibly have different parameters. You can choose to display top wait states, hosts, or users instead.

Waits	SQL	Hosts	Users	
Load By Waits				Search SQL queries
	INSERT INTO authors (id,name,email) VALUES (nextval(?) ,? ,?), (nextval(?) ,? ,?), (nextval(?) ,? ,?), (nextval(?) ,? ,?)			
	delete from authors where id < (select max(id) - ? from authors) and id > (select max(id) - ? from authors)			
	WITH cte AS (SELECT id FROM authors LIMIT ?) UPDATE authors s SET email = ? FROM cte WHERE s.id = cte.id			
	select count(*) from authors where id < (select max(id) - ? from authors) and id > (select max(id) - ? from authors)			
	select count(*) from authors where id < (select max(id) - ? from authors) and id > (select max(id) - ? from authors)			

The percentage of the database load associated with each top load item is illustrated in the **DB Load by Waits** column. This column reflects the load for that item by whatever grouping is currently selected in the **Average Active Sessions** chart. Take the case where the **Average Active Sessions** chart is grouping by hosts and you are looking at SQL queries in the top load items table. In this case, the **DB Load by Waits** bar reflects the load that query represents on the related host. Here it's colored-coded to map to the representation of that host in the **Average Active Sessions** chart.

For another example, suppose that the **Average Active Sessions** chart is grouping by wait states and you are looking at 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 what wait states are affecting that query.



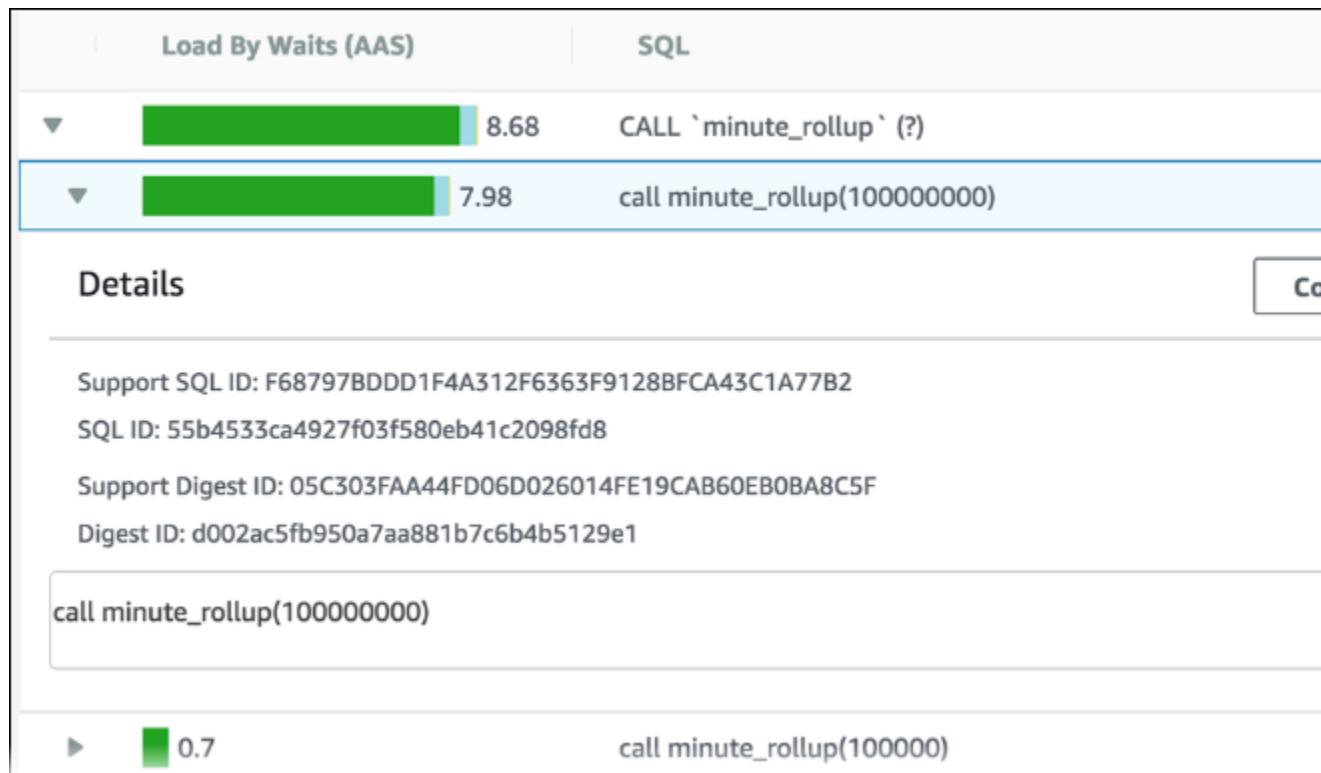
In the **Top Load Items** table, you can view the following types of identifiers (IDs) that are associated with SQL statements:

- **SQL ID** – An ID that the database uses to uniquely identify a SQL statement.
- **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.
- **Digest ID** – An ID that the database uses to uniquely identify a SQL digest. A SQL digest can contain one or more SQL statements with literals removed and white space standardized. The literals are replaced with question marks (?).

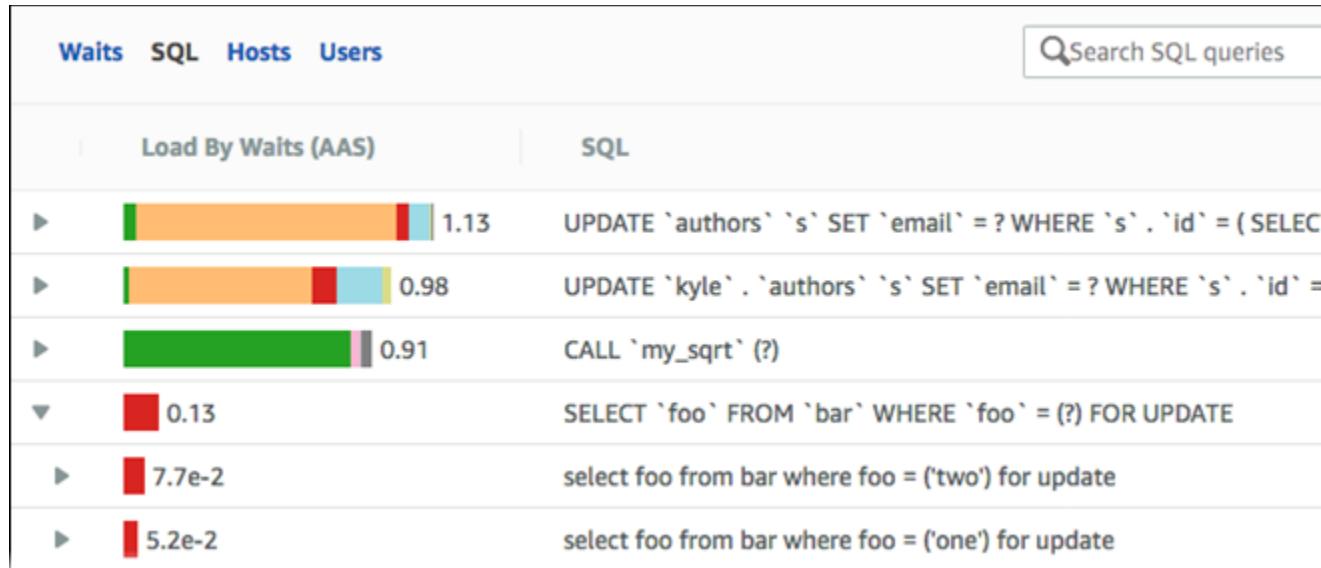
For Aurora MySQL and Aurora PostgreSQL DB instances, you can use a digest ID to find a specific SQL digest.

- **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.

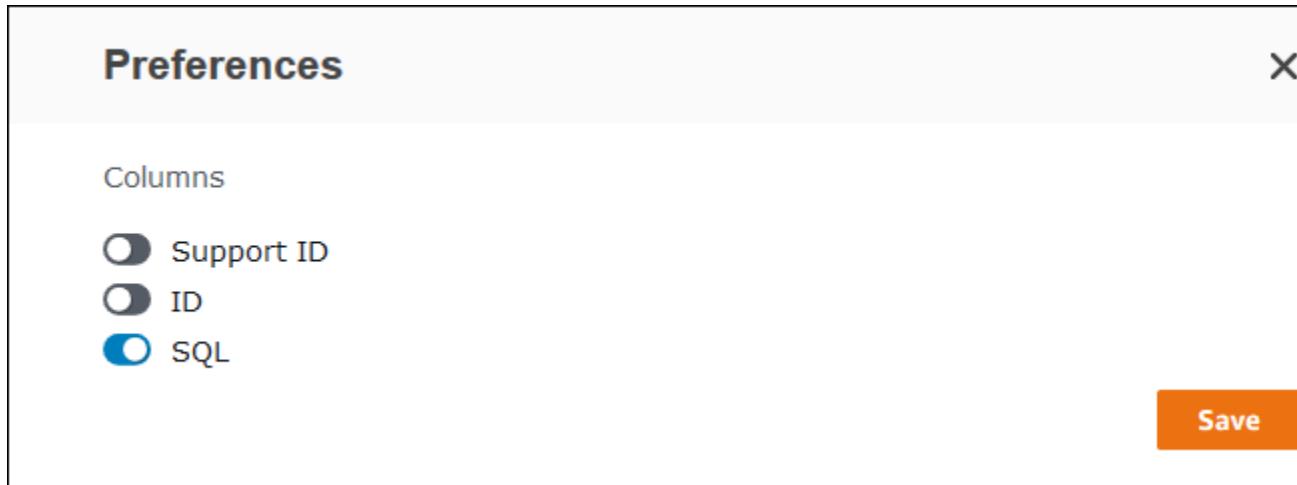
In the **Top Load Items** table, you can open a top statement to view its IDs. The following screenshot shows an open top statement.



You can control the IDs that the **Top Load Items** table shows by choosing the **Preferences** icon.



When you choose the **Preferences** icon, the **Preferences** window opens.



Enable the IDs that you want to have visible in the **Top Load Items** table, and choose **Save**.

Analyzing Database Load Using the Performance Insights Dashboard

If the **Average Active Sessions** 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 **Average Active Sessions** 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.

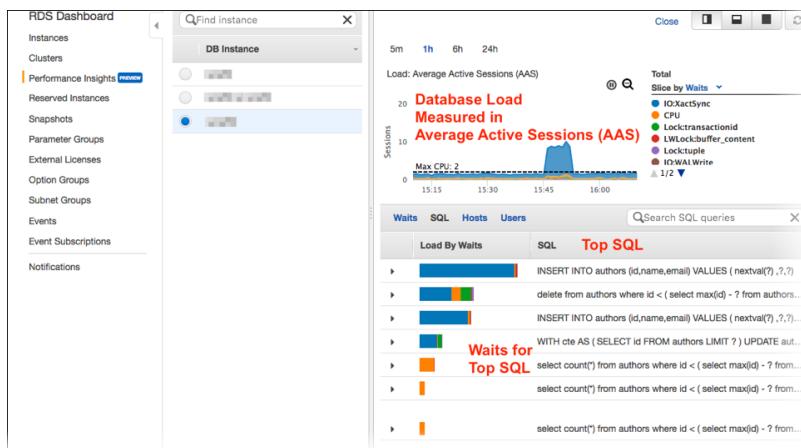
Your typical workflow for diagnosing performance issues is as follows:

1. Review the **Average Active Sessions** chart and see if there are any incidents of database load exceeding the **Max CPU** line.
2. If there is, look at the **Average Active Sessions** 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, **IO:XactSync** waits are a frequent issue. **CPU** wait is less, but it still contributes to load.

The first four roll-up queries in the **SQL** tab of the top load items table correlate strongly to the first state. Thus, those are the ones to drill into and examine the child queries of. You do so to determine how they are contributing to the performance issue.

The last three roll-up queries are the major contributors to CPU. These are the queries to investigate if CPU load is an issue.



Analyzing Statistics for Running Queries

In Amazon RDS Performance Insights, you can find statistics on running queries in the **Top Load Items** section. To view these statistics, view top SQL. Performance Insights collects statistics only for the most common queries, and these usually match the top queries by load shown in the Performance Insights dashboard.

Topics

- [SQL Digest Statistics for Aurora PostgreSQL \(p. 490\)](#)
- [Analyzing Aurora Metrics for SQL Statements That Are Running \(p. 492\)](#)

SQL Digest Statistics for Aurora PostgreSQL

To view SQL Digest statistics, the `pg_stat_statements` library must be loaded. This library is loaded by default for Aurora PostgreSQL DB clusters that are compatible with PostgreSQL 10. However, you must enable this library manually for Aurora PostgreSQL DB clusters that are compatible with PostgreSQL 9.6. To enable it manually, add `pg_stat_statements` to `shared_preload_libraries` in the DB parameter group associated with the DB instance. Then reboot your DB instance. For more information, see [Working with DB Parameter Groups and DB Cluster Parameter Groups \(p. 286\)](#).

Note

Performance Insights can only collect statistics for non-truncated queries in `pg_stat_activity`. By default, Aurora PostgreSQL databases truncate queries longer than 1,024 bytes. You can increase the query size by changing the `track_activity_query_size` parameter in the DB parameter group associated to your DB instance. When you change this parameter, a DB instance reboot is required. Performance Insights also has a limit of 5,120 bytes when collecting queries, which also impacts statistics collection. The limit of 5,120 bytes is required due to memory limitations.

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>	Bulk hits per second

Metric	Unit
db.sql_tokenized.stats.shared_blk_per_sec	Bulk reads per second
db.sql_tokenized.stats.shared_blk_dirtied_per_sec	Bulk dirty per second
db.sql_tokenized.stats.shared_blk_written_per_sec	Bulk writes per second
db.sql_tokenized.stats.local_blk_hit_per_sec	Local bulk hits per second
db.sql_tokenized.stats.local_blk_read_per_sec	Local bulk reads per second
db.sql_tokenized.stats.local_blk_dirtied_per_sec	Local bulk dirty per second
db.sql_tokenized.stats.local_blk_written_per_sec	Local bulk writes per second
db.sql_tokenized.stats.temp_blk_written_per_sec	Temporary writes per second
db.sql_tokenized.stats.temp_blk_read_per_sec	Temporary reads per second
db.sql_tokenized.stats.blk_read_time_per_sec	Average concurrent reads per second
db.sql_tokenized.stats.blk_write_time_per_sec	Average concurrent writes per second

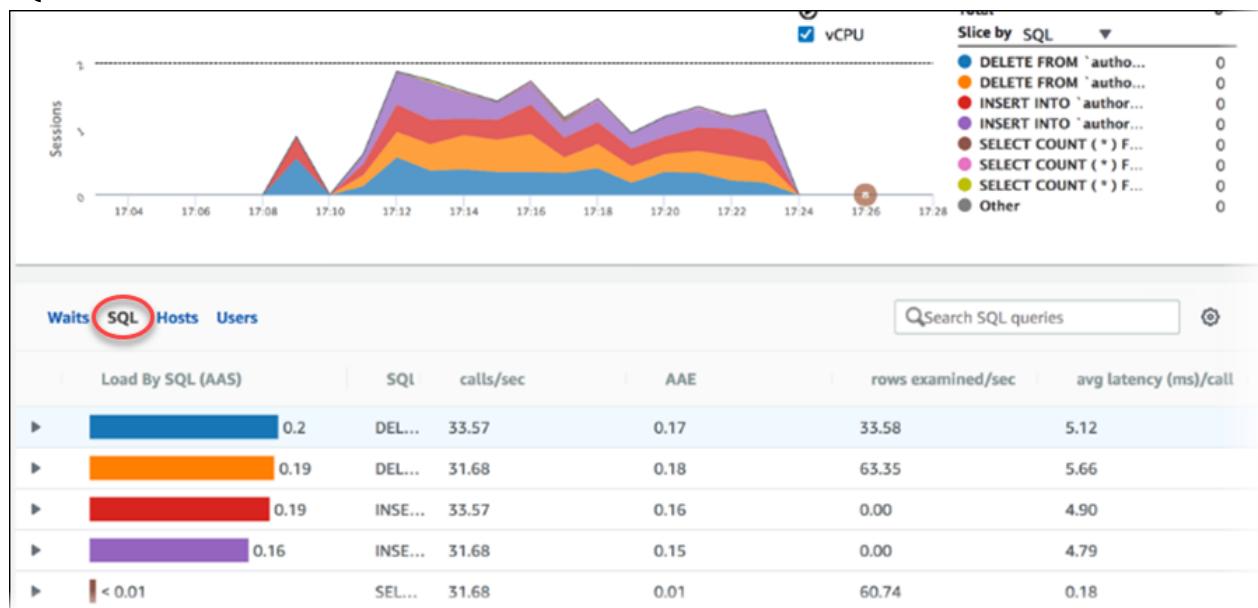
The following metrics provide per call statistics for a SQL statement.

Metric	Unit
db.sql_tokenized.stats.rows_per_call	Rows per call
db.sql_tokenized.stats.avg_latency_per_call	Average latency per call (in ms)
db.sql_tokenized.stats.shared_blk_hit_per_call	Bulk hits per call
db.sql_tokenized.stats.shared_blk_read_per_call	Bulk reads per call
db.sql_tokenized.stats.shared_blk_written_per_call	Bulk writes per call
db.sql_tokenized.stats.shared_blk_dirtied_per_call	Bulk dirty per call
db.sql_tokenized.stats.local_blk_hit_per_call	Local bulk hits per call
db.sql_tokenized.stats.local_blk_read_per_call	Local bulk reads per call
db.sql_tokenized.stats.local_blk_dirtied_per_call	Local bulk dirty per call
db.sql_tokenized.stats.local_blk_written_per_call	Local bulk writes per call
db.sql_tokenized.stats.temp_blk_written_per_call	Temporary bulk writes per call
db.sql_tokenized.stats.temp_blk_read_per_call	Temporary bulk 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.

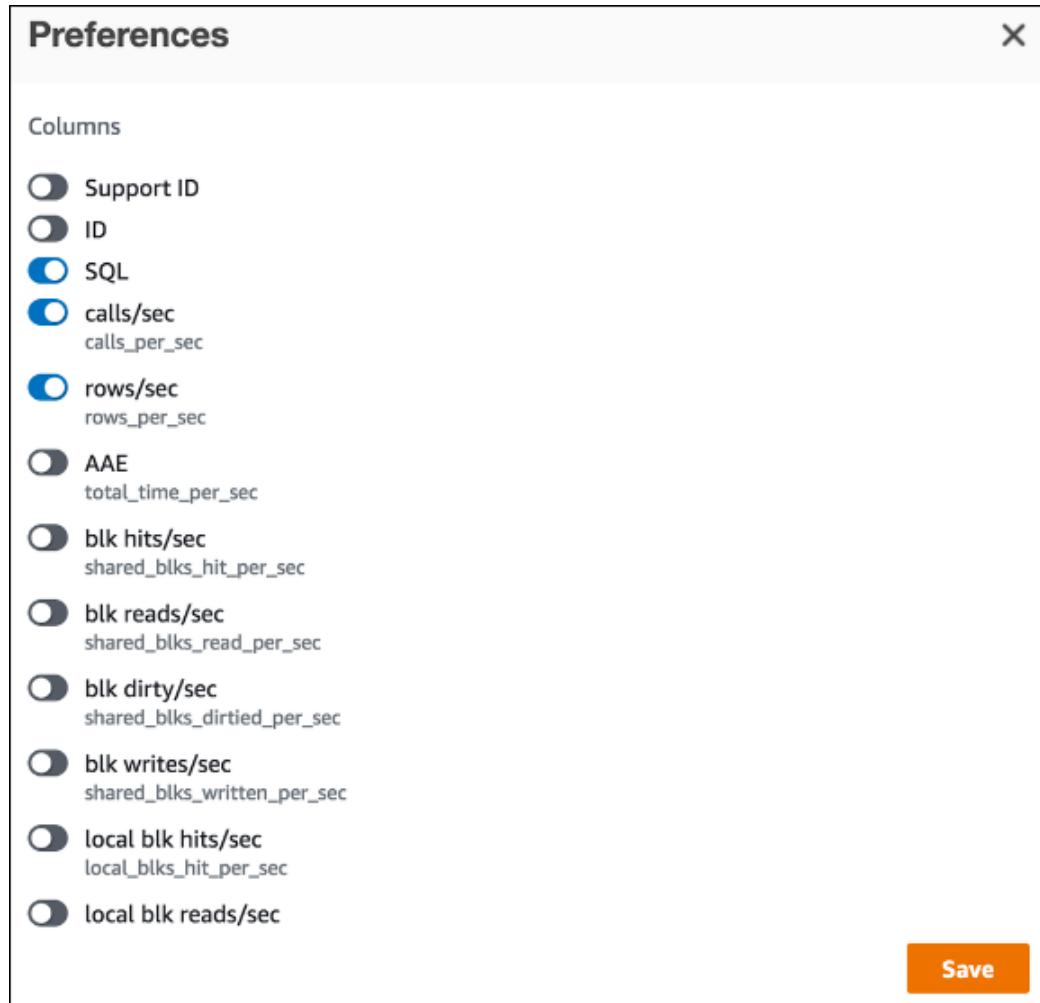
Analyzing Aurora Metrics for SQL Statements That Are Running

Using the AWS Management Console, you can view the metrics for a running SQL query by choosing the **SQL** tab.



Choose which statistics to display by choosing the gear icon in the upper-right corner of the chart.

The following screenshot shows the preferences for Aurora PostgreSQL.



Viewing More SQL Text in the Performance Insights Dashboard

By default, each row in the **Top Load Items** table shows 500 bytes of SQL text for each SQL statement. When a SQL statement is larger than 500 bytes, you can view more of the SQL statement by opening the statement in the Performance Insights dashboard. The Performance Insights dashboard can display up to 4,096 bytes for a SQL statement. You can copy the displayed SQL statement. To view more than 4,096 bytes, choose **Download full SQL** to view the SQL text up to the DB engine limit.

The limit for SQL text depends on the DB engine. The following limits apply:

- Aurora MySQL 5.7 – 4,096 bytes
- Aurora MySQL 5.6 – 1,024 bytes
- Aurora PostgreSQL – Set by the `track_activity_query_size` DB instance parameter

For Aurora PostgreSQL DB instances, you can control the limit of the SQL text size by setting the `track_activity_query_size` DB instance parameter, up to 102,400 bytes. You can use the AWS Management Console to download SQL text up to the limit you set with this parameter. For more information, see [Setting the SQL Text Limit for Aurora PostgreSQL DB Instances \(p. 495\)](#).

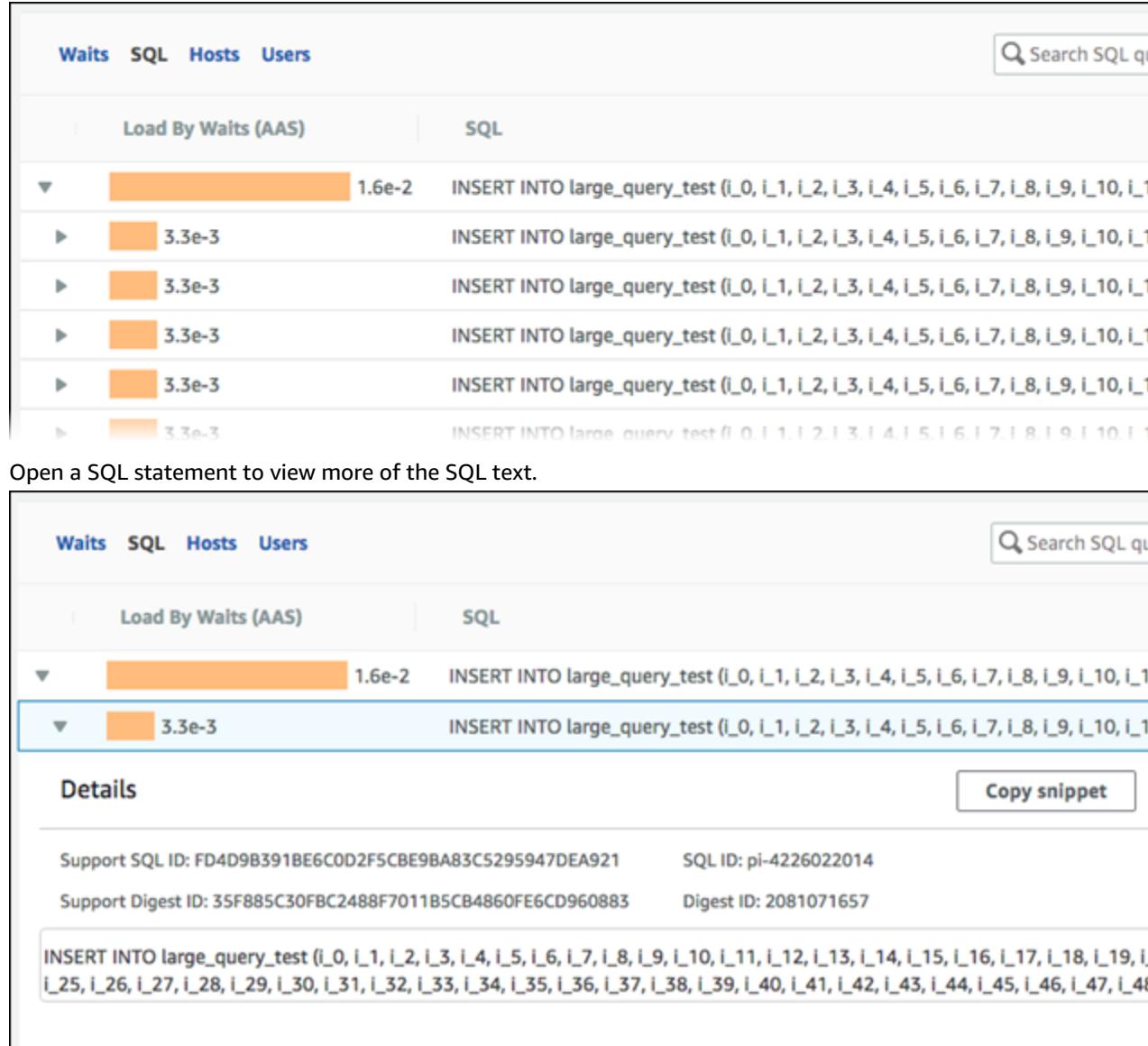
Important

Currently, you can only view and download more SQL text with the AWS Management Console. The AWS Performance Insights CLI and API can return a maximum of 500 bytes of text.

To view more SQL text in the Performance Insights dashboard

1. Open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. In the navigation pane, choose **Performance Insights**.
3. Choose a DB instance. The Performance Insights dashboard is displayed for that DB instance.

SQL statements with text larger than 500 bytes look similar to the following image.



The Performance Insights dashboard can display up to 4,096 bytes for each SQL statement.

5. (Optional) Choose **Copy snippet** to copy the displayed SQL statement, or choose **Download full SQL** 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.

Setting the SQL Text Limit for Aurora PostgreSQL DB Instances

For Aurora PostgreSQL DB instances, you can control the limit for the SQL text that can be shown on the Performance Insights dashboard.

To do so, modify the `track_activity_query_size` DB instance parameter. 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 setting for the `track_activity_query_size` parameter is 4,096 bytes.

You can increase the number of bytes to increase the SQL text size visible in the Performance Insights dashboard. The limit for the parameter is 10,240 bytes. For more information about the `track_activity_query_size` DB instance parameter, see [Run-time Statistics](#) in the PostgreSQL documentation.

To modify the parameter, change the parameter setting in the parameter group that is associated with the Aurora PostgreSQL DB instance.

If the Aurora PostgreSQL DB instance is using the default parameter group, complete the following steps:

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.

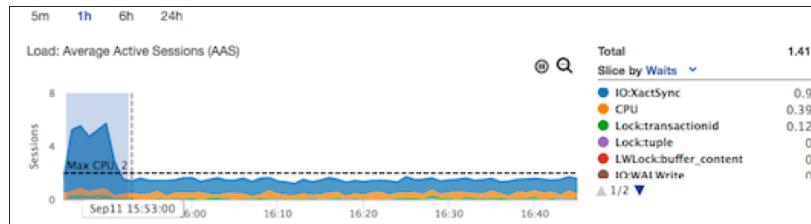
For information about setting a DB instance parameter, see [Modifying Parameters in a DB Parameter Group \(p. 291\)](#).

Additional User Interface Features

You can use other features of the Performance Insights user interface to help analyze performance data.

Click-and-Drag Zoom In

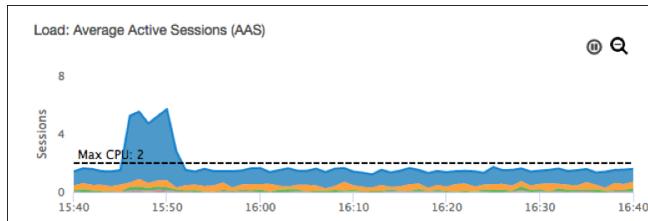
In the Performance Insights interface, you can choose a small portion of the load chart and zoom in on the detail.



To zoom in on a portion of the load chart, choose the start time and drag to the end of the time period you want. When you do this, the selected area is highlighted. When you release the mouse, the load chart zooms in on the selected region, and the **Top N** table is recalculated.

Pause and Zoom Out

In the upper-right corner of the load chart, you can find the **Pause** and **Zoom out** tools.



When you choose **Pause**, the load chart stops autorefreshing. When you choose **Pause** again, the chart resumes autorefreshing.

When you choose **Zoom out**, the load chart zooms out to the next largest time interval.

Performance Insights API

The Amazon RDS Performance Insights API provides visibility into the performance of your RDS instance, when Performance Insights is enabled for supported engine types. 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 and provided to API consumers as a two-dimensional time-series dataset. The time dimension of the data provides database 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 RDS DB instance 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. The API can be used to offload data into a database, add Performance Insights data to existing monitoring dashboards, or to build monitoring tools. To use the Performance Insights API, enable Performance Insights on one of your Amazon RDS DB instances. For information about enabling Performance Insights, see [Enabling Performance Insights \(p. 477\)](#).

The Performance Insights API provides the following operations.

Performance Insights Operation	AWS CLI Command	Description
<code>DescribeDimensionKeys</code>	<code>aws rds describe-dimension-keys</code>	Retrieves the top N dimension keys for a metric for a specific time period.
<code>GetResourceMetrics</code>	<code>aws rds 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.

For more information about the Performance Insights API, see the [Amazon RDS Performance Insights API Reference](#).

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.

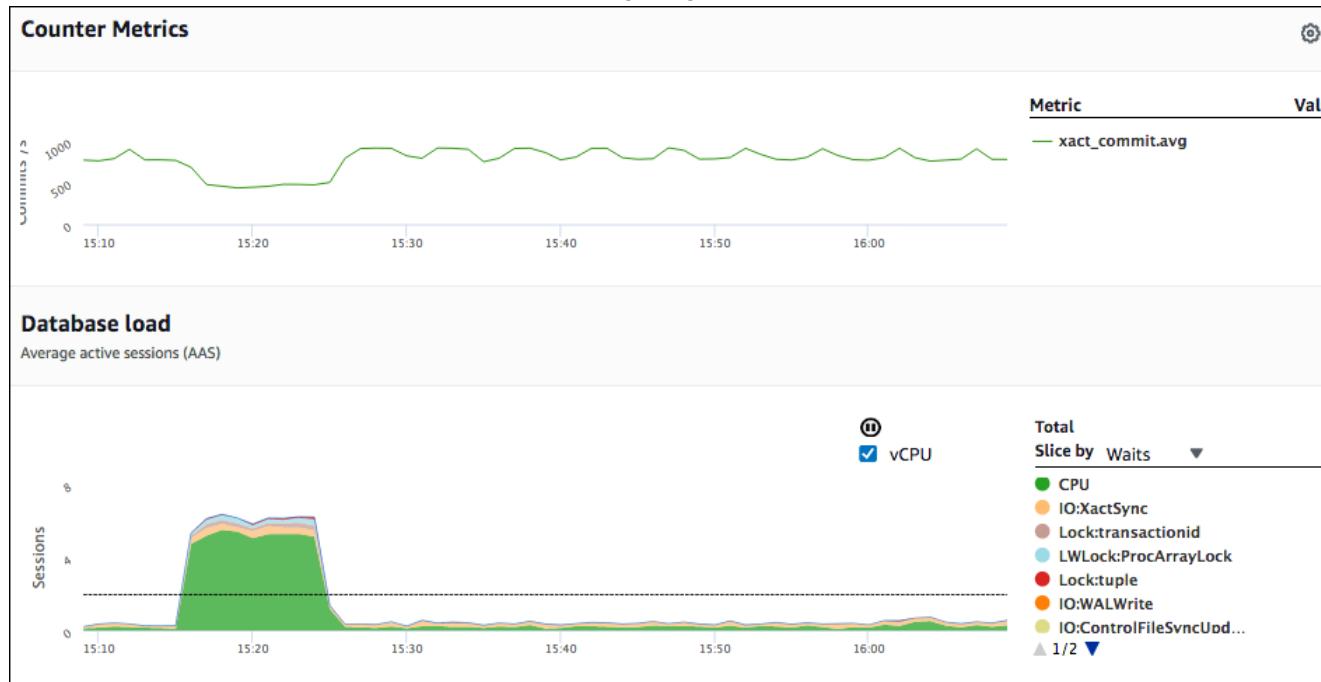
```
aws pi help
```

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` in two places in the Performance Insights dashboard. `GetResourceMetrics` is used to populate the **Counter Metrics** chart and in the **Database Load** chart, as seen in the following image.



All the metrics returned by `GetResourceMetrics` are standard time-series metrics with one exception. The exception is `db.load`, which is the core metric in Performance Insights. 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 Counters \(p. 510\)](#).

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

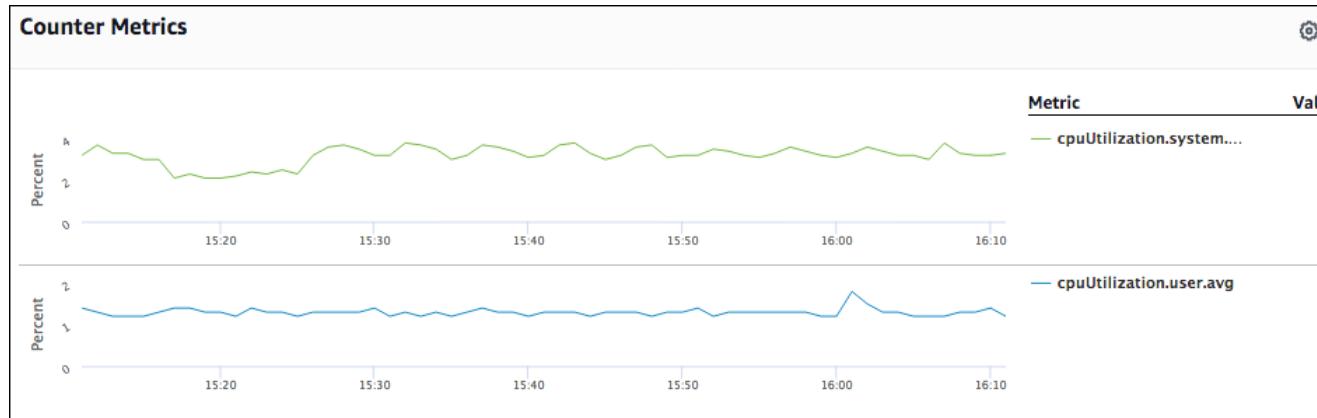
The following are several examples that use the AWS CLI for Performance Insights.

Topics

- [Retrieving Counter Metrics \(p. 498\)](#)
- [Retrieving the DB Load Average for Top Wait Events \(p. 501\)](#)
- [Retrieving the DB Load Average for Top SQL \(p. 503\)](#)
- [Retrieving the DB Load Average Filtered by SQL \(p. 506\)](#)

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, OS X, 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, OS X, 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 \(p. 498\)](#). 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, OS X, 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 are 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 \(p. 501\)](#). 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, OS X, 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.statement` – 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 key. 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 \(p. 503\)](#). 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, OS X, or Unix:

```
aws pi get-resource-metrics \  
  --service-type RDS \  
  --identifier db-ID \  
  --start-time 2018-10-30T00:00:00Z \  
  --end-time 2018-10-30T01:00:00Z \  
  --period-in-seconds 60 \  
  --metric-queries file://query.json
```

For Windows:

```
aws pi get-resource-metrics ^  
  --service-type RDS ^  
  --identifier db-ID ^
```

```
--start-time 2018-10-30T00:00:00Z ^
--end-time 2018-10-30T01:00:00Z ^
--period-in-seconds 60 ^
--metric-queries file://query.json
```

The response looks similar to the following.

```
{
  "Identifier": "db-XXX",
  "AlignedStartTime": 1556215200.0,
  "MetricList": [
    {
      "Key": {
        "Metric": "db.load.avg"
      },
      "DataPoints": [
        {
          "Timestamp": 1556218800.0,
          "Value": 1.4878117913832196
        },
        {
          "Timestamp": 1556222400.0,
          "Value": 1.192823803967328
        }
      ]
    },
    {
      "Key": {
        "Metric": "db.load.avg",
        "Dimensions": {
          "db.wait_event.type": "io",
          "db.wait_event.name": "wait/io/aurora_redo_log_flush"
        }
      },
      "DataPoints": [
        {
          "Timestamp": 1556218800.0,
          "Value": 1.1360544217687074
        },
        {
          "Timestamp": 1556222400.0,
          "Value": 1.058051341890315
        }
      ]
    },
    {
      "Key": {
        "Metric": "db.load.avg",
        "Dimensions": {
          "db.wait_event.type": "io",
          "db.wait_event.name": "wait/io/table/sql/handler"
        }
      },
      "DataPoints": [
        {
          "Timestamp": 1556218800.0,
          "Value": 0.16241496598639457
        },
        {
          "Timestamp": 1556222400.0,
          "Value": 0.05163360560093349
        }
      ]
    }
  ]
}
```

```

        },
        {
            "Key": {
                "Metric": "db.load.avg",
                "Dimensions": {
                    "db.wait_event.type": "synch",
                    "db.wait_event.name": "wait/synch/mutex/innodb/
aurora_lock_thread_slot_futex"
                }
            },
            "DataPoints": [
                {
                    "Timestamp": 1556218800.0,
                    "Value": 0.11479591836734694
                },
                {
                    "Timestamp": 1556222400.0,
                    "Value": 0.013127187864644107
                }
            ]
        },
        {
            "Key": {
                "Metric": "db.load.avg",
                "Dimensions": {
                    "db.wait_event.type": "CPU",
                    "db.wait_event.name": "CPU"
                }
            },
            "DataPoints": [
                {
                    "Timestamp": 1556218800.0,
                    "Value": 0.05215419501133787
                },
                {
                    "Timestamp": 1556222400.0,
                    "Value": 0.05805134189031505
                }
            ]
        },
        {
            "Key": {
                "Metric": "db.load.avg",
                "Dimensions": {
                    "db.wait_event.type": "synch",
                    "db.wait_event.name": "wait/synch/mutex/innodb/lock_wait_mutex"
                }
            },
            "DataPoints": [
                {
                    "Timestamp": 1556218800.0,
                    "Value": 0.017573696145124718
                },
                {
                    "Timestamp": 1556222400.0,
                    "Value": 0.002333722287047841
                }
            ]
        },
        ],
        "AlignedEndTime": 1556222400.0
    } //end of response
}

```

In this response, all values are filtered according to the contribution of tokenized SQL AKIAIOSFODNN7EXAMPLE specified in the query.json file. The keys also might follow a different order than a query without a filter, because it's the top five wait events that affected the filtered SQL.

Performance Insights Metrics Published to Amazon CloudWatch

Performance Insights automatically publishes metrics to Amazon CloudWatch. The same data can be queried from Performance Insights, but having the metrics in CloudWatch makes it easy to add CloudWatch alarms. It also makes it easy to add the metrics to existing CloudWatch Dashboards.

Metric	Description
DBLoad	The number of active sessions for the DB engine. Typically, you want the data for the average number of active sessions. In Performance Insights, this data is queried as 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": "2018-07-19T21:30:00Z",
      "Unit": "None",
      "Average": 2.1
    },
    {
      "Timestamp": "2018-07-19T21:34:00Z",
      "Unit": "None",
      "Average": 1.7
    },
    {
      "Timestamp": "2018-07-19T21:35:00Z",
      "Unit": "None",
      "Average": 1.7
    }
  ]
}
```

```

    "Unit": "None",
    "Average": 2.8
  },
  {
    "Timestamp": "2018-07-19T21:31:00Z",
    "Unit": "None",
    "Average": 1.5
  },
  {
    "Timestamp": "2018-07-19T21:32:00Z",
    "Unit": "None",
    "Average": 1.8
  },
  {
    "Timestamp": "2018-07-19T21:29:00Z",
    "Unit": "None",
    "Average": 3.0
  },
  {
    "Timestamp": "2018-07-19T21:33:00Z",
    "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 Counters

With counter metrics, you can customize the Performance Insights dashboard to include up to 10 additional graphs. These graphs that show a selection of dozens of operating system and database performance metrics. This information can be correlated with database load to help identify and analyze performance problems.

Topics

- [Performance Insights Operating System Counters \(p. 510\)](#)
- [Performance Insights Counters for Aurora MySQL \(p. 512\)](#)
- [Performance Insights Counters for Aurora PostgreSQL \(p. 515\)](#)

Performance Insights Operating System Counters

The following operating system counters are available with Performance Insights for Aurora PostgreSQL. You can find definitions for these metrics in [Viewing Enhanced Monitoring by Using CloudWatch Logs \(p. 473\)](#).

Counter	Type	Metric
active	memory	os.memory.active
buffers	memory	os.memory.buffers
cached	memory	os.memory.cached
dirty	memory	os.memory.dirty

Counter	Type	Metric
free	memory	os.memory.free
hugePagesFree	memory	os.memory.hugePagesFree
hugePagesRsvd	memory	os.memory.hugePagesRsvd
hugePageSize	memory	os.memory.hugePageSize
hugePagesSurp	memory	os.memory.hugePagesSurp
hugePagesTotal	memory	os.memory.hugePagesTotal
inactive	memory	os.memory.inactive
mapped	memory	os.memory.mapped
pageTables	memory	os.memory.pageTables
slab	memory	os.memory.slab
total	memory	os.memory.total
writeback	memory	os.memory.writeback
guest	cpuUtilization	os.cpuUtilization.guest
idle	cpuUtilization	os.cpuUtilization.idle
irq	cpuUtilization	os.cpuUtilization.irq
nice	cpuUtilization	os.cpuUtilization.nice
steal	cpuUtilization	os.cpuUtilization.steal
system	cpuUtilization	os.cpuUtilization.system
total	cpuUtilization	os.cpuUtilization.total
user	cpuUtilization	os.cpuUtilization.user
wait	cpuUtilization	os.cpuUtilization.wait
avgQueueLen	diskIO	os.diskIO.avgQueueLen
avgReqSz	diskIO	os.diskIO.avgReqSz
await	diskIO	os.diskIO.await
readIOsPS	diskIO	os.diskIO.readIOsPS
readKb	diskIO	os.diskIO.readKb
readKbPS	diskIO	os.diskIO.readKbPS
rrqmPS	diskIO	os.diskIO.rrqmPS
tps	diskIO	os.diskIO.tps
util	diskIO	os.diskIO.util
writelOsPS	diskIO	os.diskIO.writelOsPS

Counter	Type	Metric
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
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
numVCpus	general	os.general.numVCpus

Performance Insights Counters for Aurora MySQL

The following database counters are available with Performance Insights for Aurora MySQL.

Topics

- [Native Counters for Aurora MySQL \(p. 513\)](#)
- [Non-Native Counters for Aurora MySQL \(p. 514\)](#)

Native Counters for Aurora MySQL

You can find definitions for these native metrics in [Server Status Variables](#) in the MySQL documentation.

Counter	Type	Unit	Metric
Com_analyze	SQL	Queries per second	db.SQL.Com_analyze
Com_optimize	SQL	Queries per second	db.SQL.Com_optimize
Com_select	SQL	Queries per second	db.SQL.Com_select
Innodb_rows_deleted	SQL	Rows per second	db.SQL.Innodb_rows_deleted
Innodb_rows_inserted	SQL	Rows per second	db.SQL.Innodb_rows_inserted
Innodb_rows_read	SQL	Rows per second	db.SQL.Innodb_rows_read
Innodb_rows_updated	SQL	Rows per second	db.SQL.Innodb_rows_updated
Questions	SQL	Queries per second	db.SQL.Questions
Select_full_join	SQL	Queries per second	db.SQL.Select_full_join
Select_full_range_join	SQL	Queries per second	db.SQL.Select_full_range_join
Select_range	SQL	Queries per second	db.SQL.Select_range
Select_range_check	SQL	Queries per second	db.SQL.Select_range_check
Select_scan	SQL	Queries per second	db.SQL.Select_scan
Slow_queries	SQL	Queries per second	db.SQL.Slow_queries
Sort_merge_passes	SQL	Queries per second	db.SQL.Sort_merge_passes
Sort_range	SQL	Queries per second	db.SQL.Sort_range
Sort_rows	SQL	Queries per second	db.SQL.Sort_rows
Sort_scan	SQL	Queries per second	db.SQL.Sort_scan
Table_locks_immediate	Locks	Requests per second	db.Locks.Table_locks_immediate
Table_locks_waited	Locks	Requests per second	db.Locks.Table_locks_waited
Innodb_row_lock_time	Locks	Milliseconds (average)	db.Locks.Innodb_row_lock_time
Aborted_clients	Users	Connections	db.Users.Aborted_clients
Aborted_connects	Users	Connections	db.Users.Aborted_connects
Threads_created	Users	Connections	db.Users.Threads_created
Threads_running	Users	Connections	db.Users.Threads_running
Created_tmp_disk_tables	Temp	Tables per second	db.Temp.Created_tmp_disk_tables
Created_tmp_tables	Temp	Tables per second	db.Temp.Created_tmp_tables
Innodb_buffer_pool_pages _{Cached}	Cached	Pages	db.Cache.Innodb_buffer_pool_pages_c
Innodb_buffer_pool_pages _{Uncached}	Uncached	Pages	db.Cache.Innodb_buffer_pool_pages_u

Counter	Type	Unit	Metric
Innodb_buffer_pool_read_Cache	Cache	Pages per second	db.Cache.Innodb_buffer_pool_read_re
Innodb_buffer_pool_readsCache		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		db.Cache.innodb_buffer_pool_hits	The number of hits that InnoDB could satisfy from the buffer pool.	innodb_buffer_pool_read_requests - innodb_buffer_pool_reads
innodb_buffer_poolhit_rate		db.Cache.innodb_buffer_poolhit_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_poolusage		db.Cache.innodb_buffer_poolusage	The percentage of the InnoDB buffer pool that contains data (pages). Note When using compressed tables, this value can vary. For more information, see the information about Innodb_buffer_pool_pages_data and Innodb_buffer_pool_pages_total in Server Status Variables in the MySQL documentation.	Innodb_buffer_pool_pages_data / Innodb_buffer_pool_pages_total * 100.0
query_cache_hits		db.Cache.query_cache_hits	The hit ratio for the MySQL result set cache (query cache).	Qcache_hits / (QCache_hits + Com_select) * 100
innodb_rows_changed		db.SQL.innodb_rows_changed	The total number of DB row operations.	db.SQL.Innodb_rows_inserted + db.SQL.Innodb_rows_deleted

Counter	Type	Metric	Description	Definition
				+ db.SQL.Innodb_rows_updated
active_transactions	Transactions	db.Transactions	The total active transactions.	SELECT COUNT(1) AS active_transactions FROM INFORMATION_SCHEMA.INNODB_TRX
innodb_deadlocks	Locks	db.Lock.innodb	The total number of deadlocks.	SELECT COUNT AS innodb_deadlocks FROM INFORMATION_SCHEMA.INNODB_METRIC WHERE NAME='lock_deadlocks'
innodb_lock_timeouts	Locks	db.Lock.innodb	The total number of deadlocks that timed out.	SELECT COUNT AS innodb_lock_timeouts FROM INFORMATION_SCHEMA.INNODB_METRIC WHERE NAME='lock_timeouts'
innodb_row_lock_waits	Locks	db.Lock.innodb	The total number of row locks that resulted in a wait.	SELECT COUNT AS innodb_row_lock_waits FROM INFORMATION_SCHEMA.INNODB_METRIC 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 \(p. 515\)](#)
- [Non-Native Counters for Aurora PostgreSQL \(p. 516\)](#)

Native Counters for Aurora PostgreSQL

You can find definitions for these native metrics in [Viewing Statistics](#) in the PostgreSQL documentation.

Counter	Type	Unit	Metric
tup_deleted	SQL	Tuples per second	db.SQL.tup_deleted
tup_fetched	SQL	Tuples per second	db.SQL.tup_fetched
tup_inserted	SQL	Tuples per second	db.SQL.tup_inserted
tup_returned	SQL	Tuples per second	db.SQL.tup_returned
tup_updated	SQL	Tuples per second	db.SQL.tup_updated
buffers_checkpoint	Checkpoint	Blocks per second	db.Checkpoint.buffers_checkpoint

Counter	Type	Unit	Metric
checkpoints_req	Checkpoint	Checkpoints per minute	db.Checkpoint.checkpoints_req
checkpoint_sync_time	Checkpoint	Milliseconds per checkpoint	db.Checkpoint.checkpoint_sync_time
checkpoints_timed	Checkpoint	Checkpoints per minute	db.Checkpoint.checkpoints_timed
checkpoint_write_time	Checkpoint	Milliseconds per checkpoint	db.Checkpoint.checkpoint_write_time
maxwritten_clean	Checkpoint	Bgwriter clean stops per minute	db.Checkpoint.maxwritten_clean
active_transactions	Transactions	Transactions	db.Transactions.active_transactions
blocked_transactions	Transactions	Transactions	db.Transactions.blocked_transactions
max_used_xact_ids	Transactions	Transactions	db.Transactions.max_used_xact_ids
xact_commit	Transactions	Commits per second	db.Transactions.xact_commit
xact_rollback	Transactions	Rollbacks per second	db.Transactions.xact_rollback
blk_read_time	IO	Milliseconds	db.IO.blk_read_time
blks_read	IO	Blocks per second	db.IO.blks_read
buffers_backend	IO	Blocks per second	db.IO.buffers_backend
buffers_backend_fsync	IO	Blocks per second	db.IO.buffers_backend_fsync
buffers_clean	IO	Blocks per second	db.IO.buffers_clean
blks_hit	Cache	Blocks per second	db.Cache.blks_hit
buffers_alloc	Cache	Blocks per second	db.Cache.buffers_alloc
temp_files	Temp	Files per minute	db.Temp.temp_files
numbackends	User	Connections	db.User.numbackends
deadlocks	Concurrency	Deadlocks per minute	db.Concurrency.deadlocks
archived_count	WAL	Files per minute	db.WAL.archived_count
archive_failed_count	WAL	Files per minute	db.WAL.archive_failed_count

Non-Native Counters for Aurora PostgreSQL

Non-native counter metrics are counters defined by Amazon 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
checkpoint_sync_time	Checkpoint	db.Checkpoint.checkpoint_sync_time	The total 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	Checkpoint	db.Checkpoint. checkpoint_time checkpoint_time_left	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	IO	db.IO.read_latency	The time spent reading data file blocks by backends in this instance.	blk_read_time / blks_read

Logging Performance Insights Calls by Using AWS CloudTrail

Performance Insights is integrated 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 from which the request was made, who made the request, when it was made, and 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](#).

Understanding Performance Insights Log File Entries

A *trail* is a configuration that enables delivery of events as log files to an Amazon S3 bucket that you specify. CloudTrail log files contain one or more log entries. An *event* represents a single request from any source. Each event includes information about the requested operation, the date and time of the operation, request parameters, and so on. CloudTrail log files aren't an ordered stack trace of the public API calls, so they don't appear in any specific order.

The following example shows a CloudTrail log entry that demonstrates the `GetResourceMetrics` operation.

```
{  
    "eventVersion": "1.05",  
    "userIdentity": {  
        "type": "IAMUser",  
        "principalId": "AKIAIOSFODNN7EXAMPLE",  
        "arn": "arn:aws:iam::123456789012:user/johndoe",  
        "accountId": "123456789012",  
        "accessKeyId": "AKIAI44QH8DHBEXAMPLE",  
        "userName": "johndoe"  
    },  
    "eventTime": "2019-12-18T19:28:46Z",  
    "eventSource": "pi.amazonaws.com",  
    "eventName": "GetResourceMetrics",  
    "awsRegion": "us-east-1",  
    "sourceIPAddress": "72.21.198.67",  
    "userAgent": "aws-cli/1.16.240 Python/3.7.4 Darwin/18.7.0 botocore/1.12.230",  
    "requestParameters": {  
        "identifier": "db-YTDU5J5V66X7CXSCVDFD2V3SZM",  
        "metricQueries": [  
            {  
                "metric": "os.cpuUtilization.user.avg"  
            },  
            {  
                "metric": "os.cpuUtilization.idle.avg"  
            }  
        ],  
        "startTime": "Dec 18, 2019 5:28:46 PM",  
        "periodInSeconds": 60,  
        "endTime": "Dec 18, 2019 7:28:46 PM",  
        "serviceType": "RDS"  
    },  
    "responseElements": null,  
    "requestID": "9fffbe15c-96b5-4fe6-bed9-9fccff1a0525",  
    "eventID": "08908de0-2431-4e2e-ba7b-f5424f908433",  
    "eventType": "AwsApiCall",  
    "recipientAccountId": "123456789012"
```

}

Using Amazon Aurora Recommendations

Amazon Aurora provides automated recommendations for database resources, such as DB instances, DB clusters, and DB cluster parameter groups. These recommendations provide best practice guidance by analyzing DB cluster configuration, DB instance configuration, usage, and performance data.

You can find examples of these recommendations in the following table.

Type	Description	Recommendation	Additional Information
Nondefault custom memory parameters	Your DB parameter group sets memory parameters that diverge too much from the default values.	Settings that diverge too much from the default values can cause poor performance and errors. We recommend setting custom memory parameters to their default values in the DB parameter group used by the DB instance.	Working with DB Parameter Groups and DB Cluster Parameter Groups (p. 286)
Change buffering enabled for a MySQL DB instance	Your DB parameter group has change buffering enabled.	Change buffering allows a MySQL DB instance to defer some writes necessary to maintain secondary indexes. This configuration can improve performance slightly, but it can create a large delay in crash recovery. During crash recovery, the secondary index must be brought up to date. So, the benefits of change buffering are outweighed by the potentially very long crash recovery events. We recommend disabling change buffering.	Best practices for configuring parameters for Amazon RDS for MySQL, part 1: Parameters related to performance on the AWS Database Blog
Logging to table	Your DB parameter group sets logging output to <code>TABLE</code> .	Setting logging output to <code>TABLE</code> uses more storage than setting this parameter to <code>FILE</code> . To avoid reaching the storage limit, we recommend setting the logging output parameter to <code>FILE</code> .	MySQL Database Log Files (p. 567)
DB cluster with one DB instance	Your DB cluster only contains one DB instance.	For improved performance and availability, we recommend adding another DB instance with the same DB instance class in a different Availability Zone.	High Availability for Aurora (p. 27)
DB cluster in one Availability Zone	Your DB cluster has all of its DB instances in the same Availability Zone.	For improved availability, we recommend adding another DB instance with the same DB instance class in a different Availability Zone.	High Availability for Aurora (p. 27)
DB cluster outdated	Your DB cluster is running an older engine version.	We recommend that you keep your DB cluster at the most current minor version because it includes the latest security and functionality fixes. Unlike major version upgrades, minor version upgrades include only changes	Maintaining an Amazon Aurora DB Cluster (p. 408)

Type	Description	Recommendation	Additional Information
		that are backward-compatible with previous minor versions (of the same major version) of the DB engine. We recommend that you upgrade to a recent engine version	
DB cluster with different parameter groups	Your DB cluster has different DB parameter groups assigned to its DB instances.	Using different parameter groups can cause incompatibilities between the DB instances. To avoid problems and for easier maintenance, we recommend using the same parameter group for all of the DB instances in the DB cluster.	Working with DB Parameter Groups and DB Cluster Parameter Groups (p. 286)
DB cluster with different DB instance classes	Your DB cluster has DB instances that use different DB instance classes.	Using different DB instance classes for DB instances can cause problems. For example, performance might suffer if a less powerful DB instance class is promoted to replace a more powerful DB instance class. To avoid problems and for easier maintenance, we recommend using the same DB instance class for all of the DB instances in the DB cluster.	Aurora Replicas (p. 47)

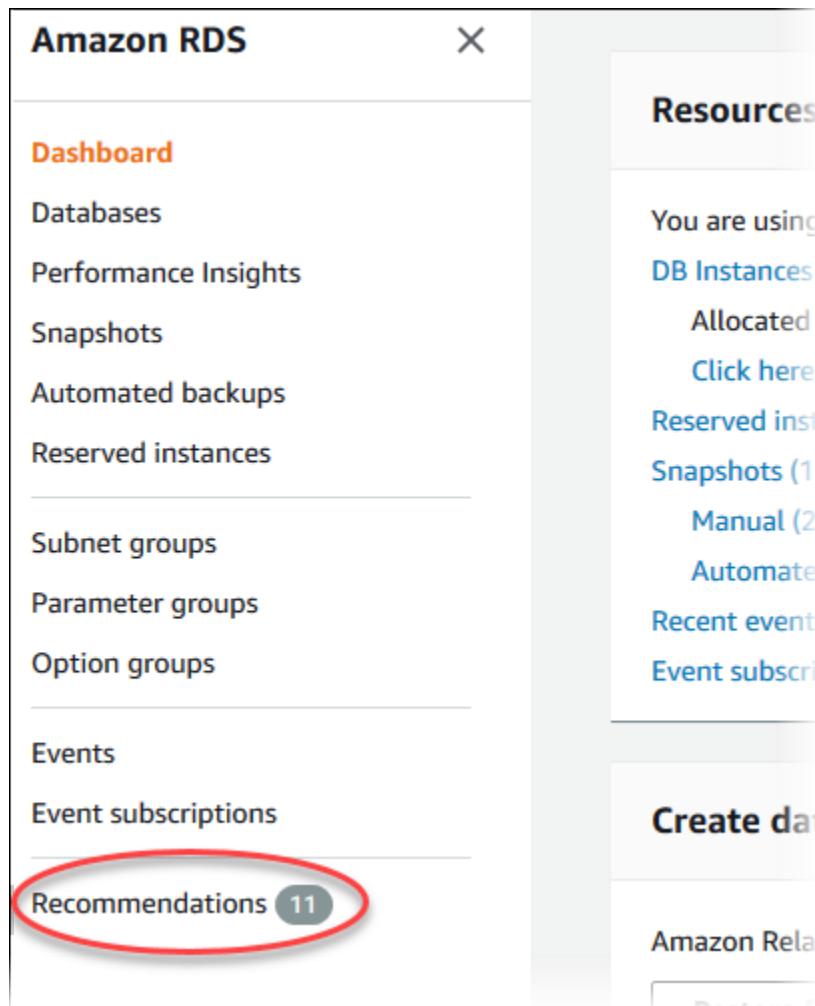
Amazon Aurora generates recommendations for a resource when the resource is created or modified. Amazon Aurora also periodically scans your resources and generates recommendations.

Responding to Amazon Aurora Recommendations

You can find recommendations in the AWS Management Console. You can perform the recommended action immediately, schedule it for the next maintenance window, or dismiss it.

To respond to Amazon Aurora recommendations

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



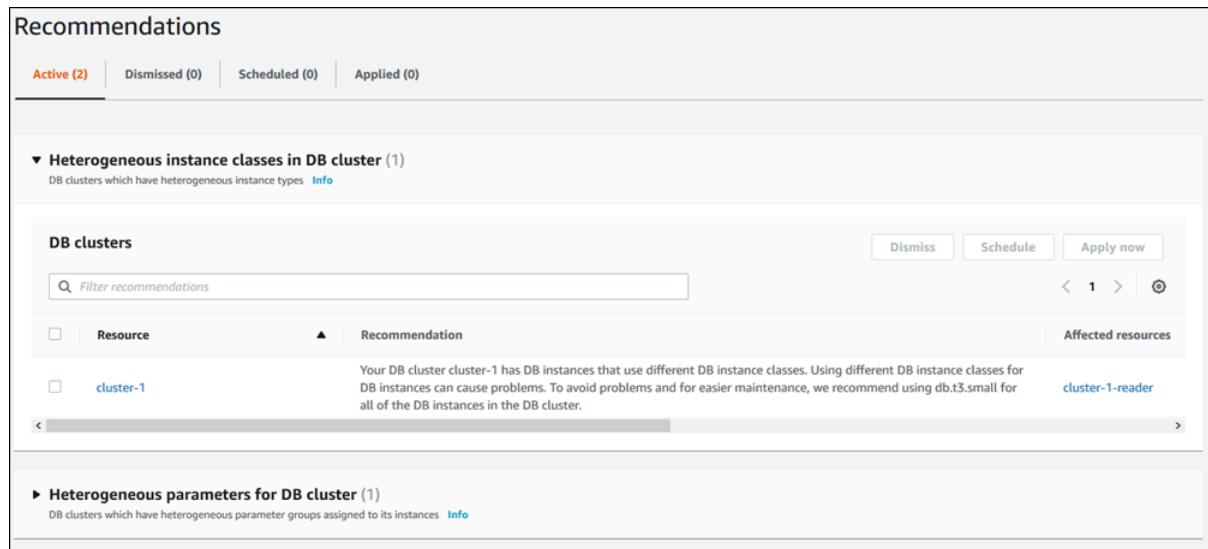
The Recommendations page appears.

This screenshot shows the 'Recommendations' page. At the top, there are tabs for Active (2), Dismissed (0), Scheduled (0), and Applied (0). Below the tabs, there are two recommendations listed in a table format. Each recommendation has a 'Details' link next to it.

Recommendation	Details
Heterogeneous instance classes in DB cluster (1)	DB clusters which have heterogeneous instance types Info
Heterogeneous parameters for DB cluster (1)	DB clusters which have heterogeneous parameter groups assigned to its instances Info

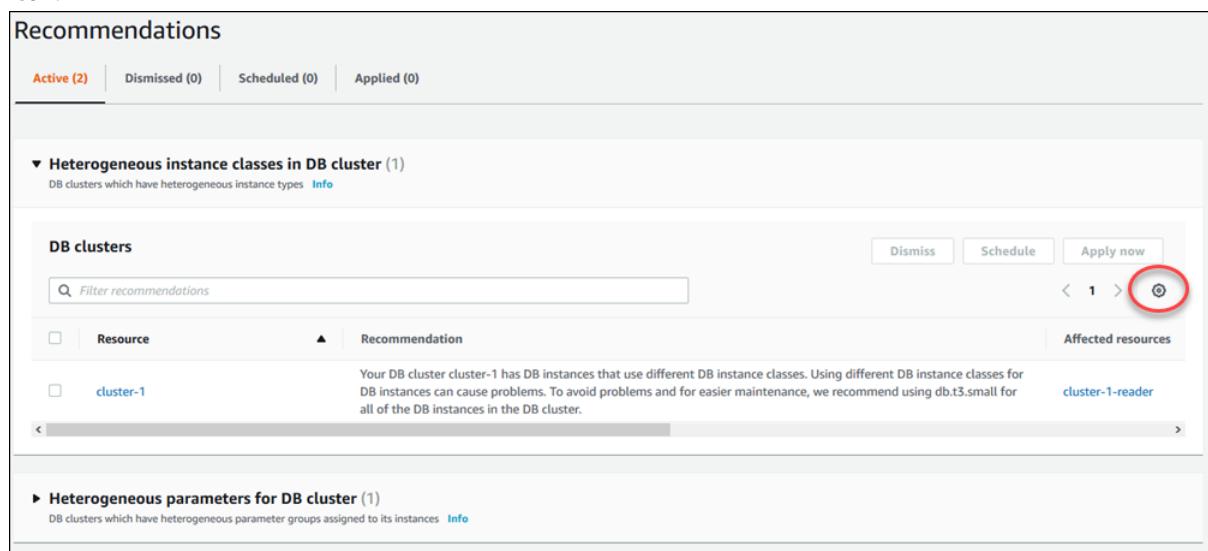
3. On the **Recommendations** page, choose one of the following:
 - **Active** – Shows the current recommendations that you can apply, dismiss, or schedule.
 - **Dismissed** – Shows the recommendations that have been dismissed. When you choose **Dismissed**, you can apply these dismissed recommendations.
 - **Scheduled** – Shows the recommendations that are scheduled but not yet applied. These recommendations will be applied in the next scheduled maintenance window.
 - **Applied** – Shows the recommendations that are currently applied.

From any list of recommendations, you can open a section to view the recommendations in that section.



The screenshot shows the 'Recommendations' interface. At the top, there are four tabs: 'Active (2)', 'Dismissed (0)', 'Scheduled (0)', and 'Applied (0)'. The 'Active (2)' tab is selected. Below the tabs, there is a section titled '▼ Heterogeneous instance classes in DB cluster (1)'. This section contains a link 'DB clusters which have heterogeneous instance types' and an 'Info' button. Underneath this, there is a table with columns 'Resource' and 'Recommendation'. A single row is visible for 'cluster-1', which has a note: 'Your DB cluster cluster-1 has DB instances that use different DB instance classes. Using different DB instance classes for DB instances can cause problems. To avoid problems and for easier maintenance, we recommend using db.t3.small for all of the DB instances in the DB cluster.' To the right of the table, there are buttons for 'Dismiss', 'Schedule', and 'Apply now'. Below the table, there is another section titled '► Heterogeneous parameters for DB cluster (1)' with a similar structure.

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



This screenshot is identical to the one above, but it includes a red circle around the small circular icon located to the right of the 'Apply now' button in the 'DB clusters' section of the 'Heterogeneous instance classes in DB cluster' section. This icon typically represents a 'Preferences' or 'Settings' window.

From the **Preferences** window that appears, you can set display options. These options include the visible columns and the number of recommendations to display on the page.

4. Manage your active recommendations:

- Choose **Active** and open one or more sections to view the recommendations in them.
- Choose one or more recommendations and choose **Apply now** (to apply them immediately), **Schedule** (to apply them in next maintenance window), or **Dismiss**.

If the **Apply now** button appears for a recommendation but is unavailable (grayed out), the DB instance is not available. You can apply recommendations immediately only if the DB instance status is **available**. For example, you can't apply recommendations immediately to the DB

instance if its status is **modifying**. In this case, wait for the DB instance to be available and then apply the recommendation.

If the **Active** 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 \(p. 264\)](#).

Note

When you choose **Apply now**, a brief DB instance outage might result.

Using Database Activity Streams with Aurora PostgreSQL

Monitoring your database activity can help you provide safeguards for your database and meet compliance and regulatory requirements. One way of monitoring database activity with Amazon Aurora with PostgreSQL compatibility is to use *Database Activity Streams*. Database Activity Streams provide a near real-time data stream of the database activity in your relational database. When you integrate Database Activity Streams with third-party monitoring tools, you can monitor and audit database activity.

Beyond external security threats, managed databases need to provide protection against insider risks from database administrators (DBAs). Database Activity Streams 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 stream of database activity is beyond the access of the DBAs that manage the database.

A stream of database activity is pushed from Aurora PostgreSQL to an Amazon Kinesis data stream that is created on behalf of your database. From Kinesis, the activity stream can then be consumed by Amazon CloudWatch or by applications for compliance management. These compliance applications include Imperva's SecureSphere Database Audit and Protection, McAfee's Data Center Security Suite, or IBM's InfoSphere Guardium. These applications can use the activity stream information to generate alerts and provide auditing of all activity on your Amazon Aurora databases.

Database Activity Streams have the following limits and requirements:

- Currently, these streams are supported only with Aurora with PostgreSQL compatibility version 2.3, which is compatible with PostgreSQL version 10.7.
- Database Activity Streams support the DB instance classes listed for Aurora PostgreSQL in [Hardware Specifications for All Available DB Instance Classes for Aurora \(p. 77\)](#), except they don't support the t3.medium instance class.
- They're not supported in the following AWS Regions:
 - China (Beijing) Region, `cn-north-1`
 - China (Ningxia) Region, `cn-northwest-1`
 - AWS GovCloud (US-East), `us-gov-east-1`
 - AWS GovCloud (US-West), `us-gov-west-1`
- They require use of AWS Key Management Service (AWS KMS) because the activity streams are always encrypted.

Topics

- [Starting an Activity Stream \(p. 525\)](#)
- [Getting the Status of an Activity Stream \(p. 527\)](#)
- [Stopping an Activity Stream \(p. 528\)](#)
- [Monitoring Database Activity Streams \(p. 528\)](#)
- [Managing Access to Database Activity Streams \(p. 540\)](#)

Starting an Activity Stream

You start an activity stream at the DB cluster level to monitor database activity for all DB instances of the cluster. Any DB instances added to the cluster are also automatically monitored.

When you start an activity stream, each database activity event, such as a change or access, generates an activity stream event. Access events are generated from SQL commands such as `CONNECT` and `SELECT`. Change events are generated from SQL commands such as `CREATE` and `INSERT`. To make each activity stream event durable, you encrypt and store it. You can choose to have the database session handle database activity events either synchronously or asynchronously:

- *Synchronous mode* – In synchronous mode, when a database session generates an activity stream event, the session blocks 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.

- *Asynchronous mode* – In 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.

Console

To start 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 the DB cluster that you want to use.
4. For **Actions**, choose **Start activity stream**. The **Database Activity Stream** window appears.
5. Enter the following settings in the **Database Activity Stream** window:
 - For **Master key**, choose a key from the list of AWS KMS keys.

The master key is used to encrypt the key that in turn encrypts the database activity logged. You must choose a master 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**.
- Choose **Apply immediately**.

If you choose **Schedule for the next maintenance window**, the database doesn't restart right away. Instead, it remains in the PENDING REBOOT state. In this case, the database activity stream will not start until either the next maintenance window or a manual restart.

When you're done entering settings, choose **Continue**.

The cluster's DB instance status shows that the database activity stream is being configured.

AWS CLI

To start Database Activity Streams for a DB cluster, configure the DB cluster using the `start-activity-stream` AWS CLI command. Identify the AWS Region for the DB cluster with the `--region` parameter. The `--apply-immediately` parameter is optional.

For Linux, OS X, or Unix:

```
aws rds --region us-west-2 \
  start-activity-stream \
  --mode sync \
  --kms-key-id MY_KMS_KEY_ARN \
  --resource-arn MY_CLUSTER_ARN \
  --apply-immediately \
  --profile MY_PROFILE_CREDENTIALS
```

For Windows:

```
aws rds --region us-west-2 ^
  start-activity-stream ^
  --mode sync ^
  --kms-key-id MY_KMS_KEY_ARN ^
  --resource-arn MY_CLUSTER_ARN ^
  --apply-immediately ^
  --profile MY_PROFILE_CREDENTIALS
```

Getting the Status of an Activity Stream

You can get the status of an activity stream using the console or AWS CLI.

Console

To get the status of a DB cluster's 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.
3. Choose the **Configuration** tab and check **Database activity stream** for status.

AWS CLI

You can get a DB cluster's activity stream configuration as the response to a `describe-db-clusters` CLI request. In the following example, see the values for `ActivityStreamKinesisStreamName`, `ActivityStreamStatus`, `ActivityStreamKmsKeyId`, and `ActivityStreamMode`.

The request is as follows:

```
aws rds --region us-west-2 describe-db-clusters --db-cluster-identifier my-cluster --
  profile MY_PROFILE_CREDENTIALS
```

The response includes the following items for a database activity stream:

```
{
  "DBClusters": [
    {
      "DBClusterIdentifier": "my-cluster",
      . .
      "ActivityStreamKinesisStreamName": "aws-rds-das-cluster-A6TSYXITZCZXJHIRVFUBZ5LTWY",
      "ActivityStreamStatus": "starting",
      "ActivityStreamKmsKeyId": "12345678-abcd-efgh-ijkl-bd041f170262",
      "ActivityStreamMode": "sync",
      "DbClusterResourceId": "cluster-ABCD123456"
      . .
    }
  ]
}
```

Stopping an Activity Stream

You can stop an activity stream using the console or AWS CLI.

Note, if you delete a DB cluster, the activity stream is stopped 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 the 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 **Apply immediately**.

If you choose **Schedule for the next maintenance window**, the database doesn't restart right away. Instead, it remains in the PENDING REBOOT state. In this case, the data activity stream is not disabled until either the next maintenance window or a manual restart.

- b. Choose **Continue**.

AWS CLI

To stop Database Activity Streams for a 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, OS X, or Unix:

```
aws rds --region us-west-2 \
    stop-activity-stream \
    --resource-arn MY_CLUSTER_ARN \
    --apply-immediately \
    --profile MY_PROFILE_CREDENTIALS
```

For Windows:

```
aws rds --region us-west-2 ^
    stop-activity-stream ^
    --resource-arn MY_CLUSTER_ARN ^
    --apply-immediately ^
    --profile MY_PROFILE_CREDENTIALS
```

Monitoring Database Activity Streams

Database Activity Streams monitor and report all activities on the database. The stream of activity is collected and transmitted to a secure server by using Amazon Kinesis. From Kinesis, the activity stream can be monitored or later consumed by other services and applications for further analysis.

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, PostgreSQL functions, and functions in Procedural Language for SQL (PL/SQL).

- **Other database information** – Activity monitored includes the full SQL statement, parameters, bind variables, the row count of affected rows from DML commands, accessed objects, and the unique database name.
- **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 a DB instance, you are notified by using RDS events. If a failure occurs, you can decide to shut down the DB instance or let it continue.

Topics

- [Accessing an Activity Stream from Kinesis \(p. 529\)](#)
- [Audit Log Contents and Examples \(p. 529\)](#)
- [Processing an Activity Stream using the AWS SDK \(p. 534\)](#)

Accessing an Activity Stream from Kinesis

When you enable an activity streams 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 such as Amazon CloudWatch. You can also connect it to compliance management applications from Imperva, McAfee, or IBM.

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-` followed by the DB cluster's resource ID. The following is an example.

```
aws-rds-das-cluster-NHVOV4PCLWHGF52NP
```

To use the Amazon RDS console to find your DB cluster's resource ID, 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 `DBActivityStreamKinesisStreamName` 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

The database activity events that are monitored are represented in the Kinesis 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

- [Audit Log Examples \(p. 530\)](#)
- [Activity Event Record \(p. 531\)](#)
- [databaseActivityEventList JSON Array \(p. 532\)](#)

Audit Log Examples

Following are sample decrypted JSON audit logs of activity event records.

Example Activity Event Record of a 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": "DatabaseActivityMonitoringRecord",  
    "clusterId": "cluster-4HNY5V4RRNPKKYB7ICFKE5JBQQ",  
    "instanceId": "db-FZJTMYKCXQBUUZ6VLU7NW3ITCM",  
    "databaseActivityEventList": [  
        {  
            "logTime": "2019-05-23 01:31:28.610198+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",  
            "netProtocol": "TCP",  
            "dbProtocol": "Postgres 3.0",  
            "type": "record"  
        }  
    ]  
}
```

Example Activity Event Record of a CREATE TABLE Statement

Following is an example of a CREATE TABLE event.

```
{  
    "type": "DatabaseActivityMonitoringRecord",  
    "clusterId": "cluster-4HNY5V4RRNPKKYB7ICFKE5JBQQ",  
    "instanceId": "db-FZJTMYKCXQBUUZ6VLU7NW3ITCM",  
    "databaseActivityEventList": [  
        {  
            "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": "49804",  
            "sessionId": "5ce5f7f0.474b",  
            "rowCount": null,  
            "commandText": "CREATE TABLE public.t1 (id integer);",  
            "paramList": [{"name": "schema", "value": "public"}, {"name": "table", "value": "t1"}],  
            "pid": 18251,  
            "clientApplication": "psql",  
            "exitCode": null,  
            "class": "DML",  
            "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"  
        }  
    ]  
}
```

```

        "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": "pgsql",
        "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"
    }
]
}

```

Example Activity Event Record of a SELECT Statement

Following is an example of a SELECT event.

```

{
    "type": "DatabaseActivityMonitoringRecord",
    "clusterId": "cluster-4HNY5V4RRNPKKYB7ICFKE5JBQQ",
    "instanceId": "db-FZJTMYKXQBUUZ6VLU7NW3ITCM",
    "databaseActivityEventList": [
        {
            "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": "pgsql",
            "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"
        }
    ]
}

```

Activity Event Record

The audit log activity event record is a JSON object that contains the following information.

JSON Field	Data Type	Description
type	string	The type of JSON record. The value is <code>DatabaseActivityMonitoringRecord</code> .
clusterId	string	The DB cluster ID
instanceId	string	The DB instance ID.
databaseActivityEventList	string	A JSON object encrypted as a base64 byte array. Take the following steps to decrypt this content: <ol style="list-style-type: none"> 1. Decrypt the value in the key JSON field using the database activity stream's master key. 2. Decrypt the <code>databaseActivityEventList</code> object using the decrypted key from step 1.

databaseActivityEventList JSON Array

The audit log payload is an encrypted `databaseActivityEventList` JSON array. The following table lists alphabetically the fields for each activity event in the decrypted `DatabaseActivityEventList` array of an audit log.

Field	Data Type	Description
class	string	The class of activity event. Valid values are the following: <ul style="list-style-type: none"> • ALL • CONNECT – A connect or disconnect event. • DDL – A DDL statement that is not included in the list of statements for the <code>ROLE</code> class. • FUNCTION – A function call or a DO block. • MISC – A miscellaneous command such as <code>DISCARD</code>, <code>FETCH</code>, <code>CHECKPOINT</code>, or <code>VACUUM</code>. • NONE • READ – A <code>SELECT</code> or <code>COPY</code> statement when the source is a relation or a query. • ROLE – A statement related to roles and privileges including <code>GRANT</code>, <code>REVOKE</code>, and <code>CREATE/ALTER/DROP ROLE</code>. • WRITE – An <code>INSERT</code>, <code>UPDATE</code>, <code>DELETE</code>, <code>TRUNCATE</code>, or <code>COPY</code> statement when the destination is a relation.
clientApplication	string	The application the client used to connect as reported by the client. The client doesn't have to provide this information, so the value can be null.
command	string	The name of the SQL command without any command details.
commandText	string	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. Sensitive Data

Field	Data Type	Description
		<p>The full SQL text is visible including any sensitive data. However, database user passwords are redacted if they can be determined from the context, such as in the following SQL statement.</p> <pre>ALTER ROLE role-name WITH password</pre>
<code>databaseName</code>	<code>string</code>	The database to which the user connected.
<code>dbProtocol</code>	<code>string</code>	The database protocol.
<code>dbUserName</code>	<code>string</code>	The database user with which the client authenticated.
<code>exitCode</code>	<code>int</code>	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> .
<code>logTime</code>	<code>string</code>	A timestamp as recorded in the auditing code path.
<code>netProtocol</code>	<code>string</code>	The network communication protocol.
<code>objectName</code>	<code>string</code>	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>	<code>string</code>	<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"> • COMPOSITE TYPE • FOREIGN TABLE • FUNCTION • INDEX • MATERIALIZED VIEW • SEQUENCE • TABLE • TOAST TABLE • VIEW • UNKNOWN
<code>paramList</code>	<code>string</code>	An array of comma-separated parameters passed to the SQL statement. If the SQL statement has no parameters, this value is an empty array.
<code>pid</code>	<code>int</code>	The process ID of the backend process that is allocated for serving the client connection.
<code>remoteHost</code>	<code>string</code>	Either the client IP address or hostname, depending on the database's <code>log_hostname</code> parameter setting.
<code>remotePort</code>	<code>string</code>	The client port number.

Field	Data Type	Description
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 host IP address.
serverType	string	The database server type, for example PostgreSQL.
serverVersion	string	The database server version.
serviceName	string	The name of the service, for example Amazon Aurora PostgreSQL-Compatible edition.
sessionId	int	A pseudo-unique session identifier.
statementId	int	An ID 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 ID for a SQL substatement. This value counts the contained substatements for each SQL statement identified by the statementId field.
type	string	The event type. Valid values are record or heartbeat.

Processing an 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 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 statementId;
        String substatementId;
        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,
getBytesArray(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;
        }
    }
}

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.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

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 = aws_encryption_sdk.decrypt(
        source=payload,
        materials_manager=aws_encryption_sdk.DefaultCryptoMaterialsManager(master_key_provider=my_key_provider)
    )
    return decrypted_plaintext

def decrypt_decompress(payload, key):
    decrypted = decrypt_payload(payload, key)
    return zlib.decompress(decrypted, zlib.MAX_WBITS + 1)

def main():
    session = boto3.session.Session()
    kms = session.client('kms', region_name=REGION_NAME)
    kinesis = session.client('kinesis', region_name=REGION_NAME)

    response = kinesis.describe_stream(StreamName=STREAM_NAME)
    shard_iters = []

```

```

        for shard in response['StreamDescription']['Shards']:
            shard_iter_response = kinesis.get_shard_iterator(StreamName=STREAM_NAME,
ShardId=shard['ShardId'],
                                                ShardIteratorType='LATEST')
            shard_iters.append(shard_iter_response['ShardIterator'])

        while len(shard_iters) > 0:
            next_shard_iters = []
            for shard_iter in shard_iters:
                response = kinesis.get_records(ShardIterator=shard_iter, Limit=10000)
                for record in response['Records']:
                    record_data = record['Data']
                    record_data = json.loads(record_data)
                    payload_decoded =
base64.b64decode(record_data['databaseActivityEvents'])
                    data_key_decoded = base64.b64decode(record_data['key'])
                    data_key_decrypt_result = kms.decrypt(CiphertextBlob=data_key_decoded,
                                                EncryptionContext={'aws:rds:dbc-
id': RESOURCE_ID})
                    print decrypt_decompress(payload_decoded,
data_key_decrypt_result['Plaintext'])
                    if 'NextShardIterator' in response:
                        next_shard_iters.append(response['NextShardIterator'])
            shard_iters = next_shard_iters

if __name__ == '__main__':
    main()

```

Managing Access to Database Activity Streams

Any user with appropriate AWS Identity and Access Management (IAM) role privileges for Database Activity Streams can create, start, stop, and modify the activity stream settings for a DB cluster. These actions are included in the audit log of the stream. For best compliance practices, we recommend that you don't provide these privileges to DBAs.

You set access to Database Activity Streams using IAM policies. For more information about Aurora authentication, see [Identity and Access Management in Amazon Aurora \(p. 191\)](#). For more information about creating IAM policies, see [Creating and Using an IAM Policy for IAM Database Access \(p. 209\)](#).

Example Policy to Allow Configuring Database Activity Streams

To give users fine-grained access to modify activity streams, use the service-specific operation context key `rds:ConfigureDBActivityStreams` 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 denies 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 denies a user or role from stopping activity streams.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "DenyStopActivityStreams",  
            "Effect": "Deny",  
            "Action": "rds:StopActivityStream",  
            "Resource": "*"  
        }  
    ]  
}
```



Using Amazon RDS Event Notification

Topics

- [Amazon RDS Event Categories and Event Messages \(p. 544\)](#)
- [Subscribing to Amazon RDS Event Notification \(p. 550\)](#)
- [Listing Your Amazon RDS Event Notification Subscriptions \(p. 553\)](#)
- [Modifying an Amazon RDS Event Notification Subscription \(p. 555\)](#)
- [Adding a Source Identifier to an Amazon RDS Event Notification Subscription \(p. 557\)](#)
- [Removing a Source Identifier from an Amazon RDS Event Notification Subscription \(p. 558\)](#)
- [Listing the Amazon RDS Event Notification Categories \(p. 559\)](#)
- [Deleting an Amazon RDS Event Notification Subscription \(p. 560\)](#)

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.

Amazon RDS groups these events into categories that you can subscribe to so that you can be notified when an event in that category occurs. You can subscribe to an event category for a DB instance, DB cluster, DB cluster snapshot, DB parameter group, or DB security group. For example, if you subscribe to the Backup category for a given DB instance, you are 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 are notified when the DB security group is changed. You also receive notification when an event notification subscription changes.

For Amazon Aurora, events occur at both the DB cluster and the DB instance level, so you can receive events if you subscribe to an Aurora DB cluster or an Aurora DB instance.

Event notifications are sent to the addresses that you provide when you create the subscription. You might want to create several different subscriptions, such as one subscription receiving all event notifications and another subscription that includes only critical events for your production DB instances. You can easily turn off notification without deleting a subscription by choosing **No** for **Enabled** in the Amazon RDS console or by setting the `Enabled` parameter to `false` using the AWS CLI or Amazon RDS API.

Important

Amazon RDS doesn't guarantee the order of events sent in an event stream. The event order is subject to change.

Note

Amazon RDS event notifications using SMS text messages are currently available for topic Amazon Resource Names (ARNs) and Amazon RDS resources in the US-East (Northern Virginia) Region. For more information on using text messages with SNS, see [Sending and Receiving SMS Notifications Using Amazon SNS](#) in the *Amazon Simple Notification Service Developer Guide*.

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. If you use the CLI or API, you create the ARN by using the Amazon SNS console or the Amazon SNS API when you create a subscription.

Billing for Amazon RDS event notification is through the Amazon Simple Notification Service (Amazon SNS). Amazon SNS fees apply when using event notification. For more information on Amazon SNS billing, see [Amazon Simple Notification Service Pricing](#).

The process for subscribing to Amazon RDS event notification is as follows:

1. Create an Amazon RDS event notification subscription by using the Amazon RDS console, AWS CLI, or API.

2. Amazon RDS sends an approval email or SMS message to the addresses you submitted with your subscription. To confirm your subscription, choose the link in the notification you were sent.
3. When you have confirmed the subscription, the status of your subscription is updated in the Amazon RDS console's **My Event Subscriptions** section.
4. You then begin to receive event notifications.

Note

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*.

The following section lists all categories and events that you can be notified of. It also provides information about subscribing to and working with Amazon RDS event subscriptions.

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, which can be a DB instance, DB snapshot, DB security group, or DB parameter group.

The following table shows the event category and a list of events when a DB instance is the source type.

Category	Amazon RDS Event ID	Description
availability	RDS-EVENT-0006	The DB instance restarted.
availability	RDS-EVENT-0004	DB instance shutdown.
availability	RDS-EVENT-0022	An error has occurred while restarting MySQL or MariaDB.
backtrack	RDS-EVENT-0131	The actual Backtrack window is smaller than the target Backtrack window you specified. Consider reducing the number of hours in your target Backtrack window. For more information about backtracking, see Backtracking an Aurora DB Cluster (p. 625).
backtrack	RDS-EVENT-0132	The actual Backtrack window is the same as the target Backtrack window.
backup	RDS-EVENT-0001	Backing up DB instance.
backup	RDS-EVENT-0002	Finished DB Instance backup.
configuration change	RDS-EVENT-0009	The DB instance has been added to a security group.
configuration change	RDS-EVENT-0024	The DB instance is being converted to a Multi-AZ DB instance.
configuration change	RDS-EVENT-0030	The DB instance is being converted to a Single-AZ DB instance.
configuration change	RDS-EVENT-0012	Applying modification to database instance class.

Category	Amazon RDS Event ID	Description
configuration change	RDS-EVENT-0018	The current storage settings for this DB instance are being changed.
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-0028	Automatic backups for this DB instance have been disabled.
configuration change	RDS-EVENT-0032	Automatic backups for this DB instance have been enabled.
configuration change	RDS-EVENT-0033	There are [count] users that match the master user name. Users not tied to a specific host have been reset.
configuration change	RDS-EVENT-0025	The DB instance has been converted to a Multi-AZ DB instance.
configuration change	RDS-EVENT-0029	The DB instance has been converted to a Single-AZ DB instance.
configuration change	RDS-EVENT-0014	The DB instance class for this DB instance has changed.
configuration change	RDS-EVENT-0017	The storage settings for this DB instance have changed.
configuration change	RDS-EVENT-0010	The DB instance has been removed from a security group.
configuration change	RDS-EVENT-0016	The master password for the DB instance has been reset.
configuration change	RDS-EVENT-0067	An attempt to reset the master password for the DB instance has failed.
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.
failover	RDS-EVENT-0034	Amazon RDS is not attempting a requested failover because a failover recently occurred on the DB instance.
failover	RDS-EVENT-0013	A Multi-AZ failover that resulted in the promotion of a standby instance has started.
failover	RDS-EVENT-0015	A Multi-AZ failover that resulted in the promotion of a standby instance is complete. It may take several minutes for the DNS to transfer to the new primary DB instance.

Category	Amazon RDS Event ID	Description
failover	RDS-EVENT-0065	The instance has recovered from a partial failover.
failover	RDS-EVENT-0049	A Multi-AZ failover has completed.
failover	RDS-EVENT-0050	A Multi-AZ activation has started after a successful instance recovery.
failover	RDS-EVENT-0051	A Multi-AZ activation is complete. Your database should be accessible now.
failure	RDS-EVENT-0031	The DB instance has failed due to an incompatible configuration or an underlying storage issue. Begin a point-in-time-restore for 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-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-0058	Error while creating Statspack user account PERFSTAT. Please drop the account before adding the Statspack option.
failure	RDS-EVENT-0079	Enhanced Monitoring cannot be enabled without the enhanced monitoring IAM role. For information on creating the enhanced monitoring IAM role, see To create an IAM role for Amazon RDS Enhanced Monitoring (p. 470) .
failure	RDS-EVENT-0080	Enhanced Monitoring was disabled due to an error making the configuration change. It is likely that the enhanced monitoring IAM role is configured incorrectly. For information on creating the enhanced monitoring IAM role, see To create an IAM role for Amazon RDS Enhanced Monitoring (p. 470) .
failure	RDS-EVENT-0082	Aurora was unable to copy backup data from an Amazon S3 bucket. It is likely that the permissions for Aurora to access the Amazon S3 bucket are configured incorrectly. For more information, see Migrating Data from MySQL by Using an Amazon S3 Bucket (p. 590) .
low storage	RDS-EVENT-0089	The DB instance has consumed more than 90% of its allocated storage. You can monitor the storage space for a DB instance using the Free Storage Space metric. For more information, see Viewing DB Instance Metrics (p. 443) .

Category	Amazon RDS Event ID	Description
low storage	RDS-EVENT-0007	The allocated storage for the DB instance has been exhausted. To resolve this issue, you should allocate additional storage for the DB instance. For more information, see the RDS FAQ . You can monitor the storage space for a DB instance using the Free Storage Space metric. For more information, see Viewing DB Instance Metrics (p. 443) .
maintenance	RDS-EVENT-0026	Offline maintenance of the DB instance is taking place. The DB instance is currently unavailable.
maintenance	RDS-EVENT-0027	Offline maintenance of the DB instance is complete. The DB instance is now available.
maintenance	RDS-EVENT-0047	Patching of the DB instance has completed.
maintenance	RDS-EVENT-0155	The DB instance has a DB engine minor version upgrade available.
notification	RDS-EVENT-0044	Operator-issued notification. For more information, see the event message.
notification	RDS-EVENT-0048	Patching of the DB instance has been delayed.
notification	RDS-EVENT-0054	The MySQL storage engine you are using is not InnoDB, which is the recommended MySQL storage engine for Amazon RDS. For information about MySQL storage engines, see Supported Storage Engines for MySQL on Amazon RDS .
notification	RDS-EVENT-0055	<p>The number of tables you have for your DB instance exceeds the recommended best practices for Amazon RDS. Please reduce the number of tables on your DB instance.</p> <p>For information about recommended best practices, see Best Practices with Amazon Aurora (p. 987).</p>
notification	RDS-EVENT-0056	<p>The number of databases you have for your DB instance exceeds the recommended best practices for Amazon RDS. Please reduce the number of databases on your DB instance.</p> <p>For information about recommended best practices, see Best Practices with Amazon Aurora (p. 987).</p>
notification	RDS-EVENT-0087	The DB instance has been stopped.
notification	RDS-EVENT-0088	The DB instance has been started.
notification	RDS-EVENT-0154	The DB instance is being started due to it exceeding the maximum allowed time being stopped.

Category	Amazon RDS Event ID	Description
notification	RDS-EVENT-0157	RDS can't modify the DB instance class because the target instance class can't support the number of databases that exist on the source DB instance. The error message appears as: "The instance has <i>N</i> databases, but after conversion it would only support <i>N</i> ".
notification	RDS-EVENT-0158	DB instance is in a state that can't be upgraded.
notification	RDS-EVENT-0159	DB snapshot export task failed.
notification	RDS-EVENT-0160	DB snapshot export task canceled.
notification	RDS-EVENT-0161	DB snapshot export task completed.
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.
read replica	RDS-EVENT-0045	An error has occurred in the read replication process. For more information, see the event message. For information on troubleshooting Read Replica errors, see Troubleshooting a MySQL or MariaDB Read Replica Problem .
read replica	RDS-EVENT-0046	The Read Replica has resumed replication. This message appears when you first create a Read Replica, or as a monitoring message confirming that replication is functioning properly. If this message follows an RDS-EVENT-0045 notification, then replication has resumed following an error or after replication was stopped.
read replica	RDS-EVENT-0057	Replication on the Read Replica was terminated.
read replica	RDS-EVENT-0062	Replication on the Read Replica was manually stopped.
read replica	RDS-EVENT-0063	Replication on the Read Replica was reset.
recovery	RDS-EVENT-0020	Recovery of the DB instance has started. Recovery time will vary with the amount of data to be recovered.
recovery	RDS-EVENT-0021	Recovery of the DB instance is complete.
recovery	RDS-EVENT-0023	A manual backup has been requested but Amazon RDS is currently in the process of creating a DB snapshot. Submit the request again after Amazon RDS has completed the DB snapshot.
recovery	RDS-EVENT-0052	Recovery of the Multi-AZ instance has started. Recovery time will vary with the amount of data to be recovered.

Category	Amazon RDS Event ID	Description
recovery	RDS-EVENT-0053	Recovery of the Multi-AZ instance is complete.
restoration	RDS-EVENT-0008	The DB instance has been restored from a DB snapshot.
restoration	RDS-EVENT-0019	The DB instance has been restored from a point-in-time backup.

The following table shows the event category and a list of events when a DB parameter group is the source type.

Category	RDS Event ID	Description
configuration change	RDS-EVENT-0037	The parameter group was modified.

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 Amazon EC2 security group owned by [user] does not exist; authorization for the security group has been revoked.

The following table shows the event category and a list of events when an Aurora DB cluster is the source type.

Category	RDS Event ID	Description
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 Migrating Data from MySQL by Using an Amazon S3 Bucket (p. 590) .

Category	RDS Event ID	Description
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-0141	Scaling initiated for the Aurora Serverless DB cluster.
notification	RDS-EVENT-0142	Scaling completed for the Aurora Serverless DB cluster.
notification	RDS-EVENT-0143	Scaling failed 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.
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.

The following table shows the event category and a list of events when an Aurora DB cluster snapshot is the source type.

Category	RDS Event ID	Description
backup	RDS-EVENT-0074	Creation of a manual DB cluster snapshot has started.
backup	RDS-EVENT-0075	A manual DB cluster snapshot has been created.

Subscribing to Amazon RDS Event Notification

You can create an Amazon RDS event notification subscription so you can be notified when an event occurs for a given DB instance, DB snapshot, DB security group, or DB parameter group. The simplest way to create a subscription is with the RDS console. If you choose to create event notification subscriptions

using the CLI or API, you must create an Amazon Simple Notification Service topic and subscribe to that topic with the Amazon SNS console or Amazon SNS API. You will also need to retain the Amazon Resource Name (ARN) of the topic because it is used when submitting CLI commands or API operations. For information on creating an SNS topic and subscribing to it, see [Getting Started with Amazon SNS](#) in the *Amazon Simple Notification Service Developer Guide*.

You can specify the type of source you want to be notified of and the Amazon RDS source that triggers the event. These are defined by the **SourceType** (type of source) and the **SourceIdentifier** (the Amazon RDS source generating the event). If you specify both the **SourceType** and **SourceIdentifier**, such as `SourceType = db-instance` and `SourceIdentifier = myDBInstance1`, you receive all the DB instance events for the specified source. If you specify a **SourceType** but don't specify a **SourceIdentifier**, you receive notice of the events for that source type for all your Amazon RDS sources. If you don't specify either the **SourceType** or the **SourceIdentifier**, you are notified of events generated from all Amazon RDS sources belonging to your customer account.

Note

Event notifications might take up to five minutes to be delivered.

Amazon RDS event notification is only available for unencrypted SNS topics. If you specify an encrypted SNS topic, event notifications aren't sent for the topic.

Console

To subscribe to RDS event notification

1. Sign in to the AWS Management Console and open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. In navigation pane, choose **Event subscriptions**.
3. In the **Event subscriptions** pane, choose **Create event subscription**.
4. In the **Create event subscription** dialog box, do the following:
 - a. For **Name**, enter a name for the event notification subscription.
 - b. For **Send notifications to**, choose an existing Amazon SNS ARN for an Amazon SNS topic, or choose **create topic** to enter the name of a topic and a list of recipients.
 - c. For **Source type**, choose a source type.
 - d. Choose **Yes** to enable the subscription. If you want to create the subscription but to not have notifications sent yet, choose **No**.
 - e. Depending on the source type you selected, choose the event categories and sources that you want to receive event notifications for.
 - f. Choose **Create**.

The Amazon RDS console indicates that the subscription is being created.

Event subscriptions (2)			
		Edit	Delete
Name		Status	Source Type
Configchancerdpgres		active	Instances
Test		creating	Instances

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, OS X, or Unix:

```
aws rds create-event-subscription \
--subscription-name myeventsubscription \
--sns-topic-arn arn:aws:sns:us-east-1:802#####:myawsuser-RDS \
--enabled
```

For Windows:

```
aws rds create-event-subscription ^
--subscription-name myeventsubscription ^
--sns-topic-arn arn:aws:sns:us-east-1:802#####:myawsuser-RDS ^
--enabled
```

API

To subscribe to Amazon RDS event notification, call the Amazon RDS API function [CreateEventSubscription](#). Include the following required parameters:

- **SubscriptionName**
- **SnsTopicArn**

Listing Your 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"/>	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.

Example

The following code example lists up to 100 event subscriptions.

```
https://rds.us-east-1.amazonaws.com/?Action=DescribeEventSubscriptions
```

```
&MaxRecords=100
&SignatureMethod=HmacSHA256
&SignatureVersion=4
&Version=2014-10-31
&X-Amz-Algorithm=AWS4-HMAC-SHA256
&X-Amz-Credential=AKIADQKE4SARGYLE/20140428/us-east-1/rds/aws4_request
&X-Amz-Date=20140428T161907Z
&X-Amz-SignedHeaders=content-type;host;user-agent;x-amz-content-sha256;x-amz-date
&X-Amz-Signature=4208679fe967783a1a149c826199080a066085d5a88227a80c6c0cadb3e8c0d4
```

The following example describes the `myfirsteventsubscription`.

```
https://rds.us-east-1.amazonaws.com/
?Action=DescribeEventSubscriptions
&SignatureMethod=HmacSHA256
&SignatureVersion=4
&SubscriptionName=myfirsteventsubscription
&Version=2014-10-31
&X-Amz-Algorithm=AWS4-HMAC-SHA256
&X-Amz-Credential=AKIADQKE4SARGYLE/20140428/us-east-1/rds/aws4_request
&X-Amz-Date=20140428T161907Z
&X-Amz-SignedHeaders=content-type;host;user-agent;x-amz-content-sha256;x-amz-date
&X-Amz-Signature=4208679fe967783a1a149c826199080a066085d5a88227a80c6c0cadb3e8c0d4
```

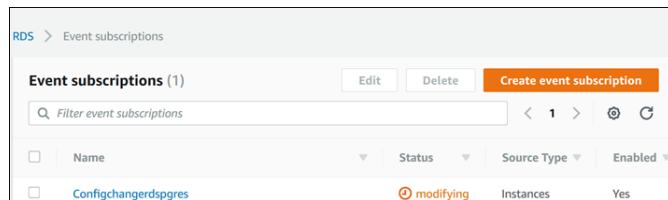
Modifying an Amazon RDS Event Notification Subscription

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, OS X, 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 \(p. 555\)](#).

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, OS X, 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 \(p. 555\)](#).

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, OS X, or Unix:

```
aws rds remove-source-identifier-from-subscription \
--subscription-name myrdseventsSubscription \
--source-identifier mysqldb
```

For Windows:

```
aws rds remove-source-identifier-from-subscription ^
--subscription-name myrdseventsSubscription ^
--source-identifier mysqldb
```

API

To remove a source identifier from an Amazon RDS event notification subscription, use the Amazon RDS API `RemoveSourceIdentifierFromSubscription` command. Include the following required parameters:

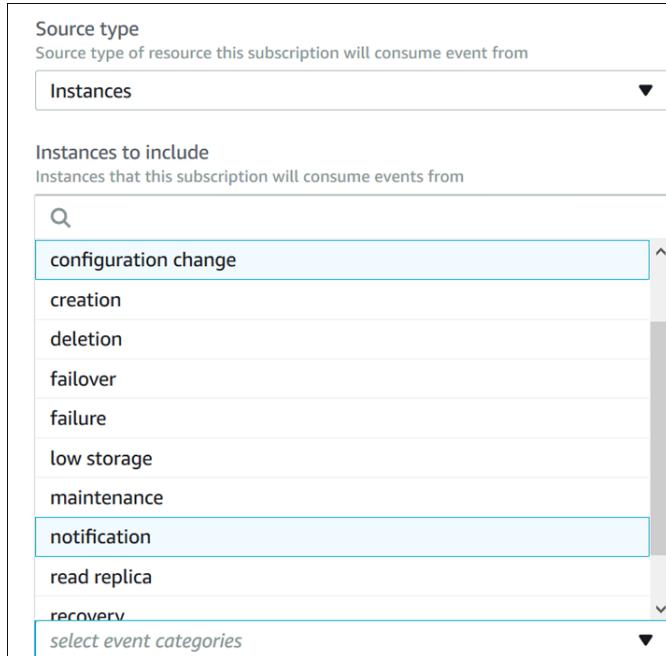
- `SubscriptionName`
- `SourceIdentifier`

Listing the Amazon RDS Event Notification Categories

All events for a resource type are grouped into categories. To view the list of categories available, use the following procedures.

Console

When you create or modify an event notification subscription, the event categories are displayed in the Amazon RDS console. For more information, see [Modifying an Amazon RDS Event Notification Subscription \(p. 555\)](#).



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.

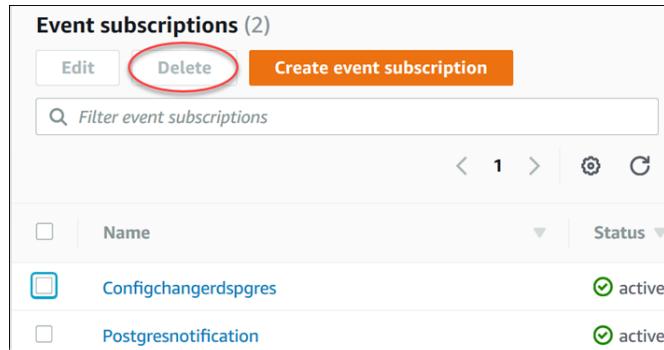
Deleting an Amazon RDS Event Notification Subscription

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.



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`

Viewing Amazon RDS Events

Amazon RDS keeps a record of events that relate to your DB instances, DB snapshots, DB security groups, and DB parameter groups. This information includes the date and time of the event, the source name and source type of the event, and a message associated with the event.

You can retrieve events for your RDS resources through the AWS Management Console, which shows events from the past 24 hours. You can also retrieve events for your RDS resources by using the [describe-events](#) AWS CLI command, or the [DescribeEvents](#) RDS API operation. If you use the AWS CLI or the RDS API to view events, you can retrieve events for up to the past 14 days.

Console

To view all Amazon RDS instance events for the past 24 hours

1. Sign in to the AWS Management Console and open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. In the navigation pane, choose **Events**. The available events appear in a list.
3. Use the **Filter** list to filter the events by type, and use the text box to the right of the **Filter** list to further filter your results. For example, the following screenshot shows a list of events filtered by the DB instance event type and containing the characters **1318**.

Identifier	Type
feb1318	DB cluster snapshots
feb1318	DB cluster snapshots

AWS CLI

To view all Amazon RDS instance events for the past 7 days

You can view all Amazon RDS instance events for the past 7 days by calling the [describe-events](#) AWS CLI command and setting the `--duration` parameter to 10080.

```
aws rds describe-events --duration 10080
```

API

To view all Amazon RDS instance events for the past 14 days

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.

```
https://rds.us-west-2.amazonaws.com/  
?Action=DescribeEvents
```

```
&Duration=20160
&MaxRecords=100
&SignatureMethod=HmacSHA256
&SignatureVersion=4
&Version=2014-10-31
&X-Amz-Algorithm=AWS4-HMAC-SHA256
&X-Amz-Credential=AKIADQKE4SARGYLE/20140421/us-west-2/rds/aws4_request
&X-Amz-Date=20140421T194733Z
&X-Amz-SignedHeaders=content-type;host;user-agent;x-amz-content-sha256;x-amz-date
&X-Amz-Signature=8e313cabcd9766c56a2886b5b298fd944e0b7cfa248953c82705fdd0374f27
```

Amazon Aurora Database Log Files

You can view, download, and watch database logs using the Amazon RDS console, the AWS Command Line Interface (AWS CLI), or the Amazon RDS API. Viewing, downloading, or watching transaction logs is not supported.

Note

In some cases, logs contain hidden data. Therefore, the AWS Management Console might show content in a log file, but the log file might be empty when you download it.

Viewing and Listing Database Log Files

You can view database log files for your DB engine by using the Amazon RDS console. You can list what log files are available for download or monitoring by using the AWS CLI or Amazon RDS API.

Console

To view a database log file

1. Open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. In the navigation pane, choose **Databases**.
3. Choose the name of the DB instance that has the log file that you want to view.
4. Choose the **Logs & events** tab.
5. Scroll down to the **Logs** section.
6. In the **Logs** section, choose the log that you want to view, and then choose **View**.

AWS CLI

To list the available database log files for a DB instance, use the AWS CLI `describe-db-log-files` command.

The following example returns a list of log files for a DB instance named `my-db-instance`.

Example

```
aws rds describe-db-log-files --db-instance-identifier my-db-instance
```

RDS API

To list the available database log files for a DB instance, use the Amazon RDS API `DescribeDBLogFiles` action.

Downloading a Database Log File

You can use the Amazon RDS console, AWS CLI or API to download a database log file.

Console

To download a database log file

1. Open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. In the navigation pane, choose **Databases**.

3. Choose the name of the DB instance that has the log file that you want to view.
4. Choose the **Logs & events** tab.
5. Scroll down to the **Logs** section.
6. In the **Logs** section, choose the button next to the log that you want to download, and then choose **Download**.
7. Open the context (right-click) menu for the link provided, and then choose **Save Link As**. Enter the location where you want the log file to be saved, and then choose **Save**.



AWS CLI

To download a database log file, use the AWS CLI command [download-db-log-file-portion](#). By default, this command downloads only the latest portion of a log file. However, you can download an entire file by specifying the parameter `--starting-token 0`.

The following example shows how to download the entire contents of a log file called `log/ERROR.4` and store it in a local file called `errorlog.txt`.

Example

For Linux, OS X, or Unix:

```
aws rds download-db-log-file-portion \
    --db-instance-identifier myexampledb \
    --starting-token 0 --output text \
    --log-file-name log/ERROR.4 > errorlog.txt
```

For Windows:

```
aws rds download-db-log-file-portion ^
    --db-instance-identifier myexampledb ^
    --starting-token 0 --output text ^
    --log-file-name log/ERROR.4 > errorlog.txt
```

RDS API

To download a database log file, use the Amazon RDS API [DownloadDBLogFilePortion](#) action.

Watching a Database Log File

You can monitor the contents of a log file by using the Amazon RDS console.

Console

To watch a database log file

1. Open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. In the navigation pane, choose **Databases**.
3. Choose the name of the DB instance that has the log file that you want to view.
4. Choose the **Logs & events** tab.
5. In the **Logs** section, choose a log file, and then choose **Watch**.

Publishing Database Logs to Amazon CloudWatch Logs

In addition to viewing and downloading DB instance logs, you can publish logs to Amazon CloudWatch Logs. CloudWatch Logs lets you perform real-time analysis of the log data, store the data in highly durable storage, and manage the data with the CloudWatch Logs Agent. AWS retains log data published to CloudWatch Logs for an indefinite time period unless you specify a retention period. For more information, see [Change Log Data Retention in CloudWatch Logs](#).

For engine-specific information, see the following:

- the section called “[Publishing Aurora MySQL Logs to CloudWatch Logs](#)” (p. 759)
- the section called “[Publishing Aurora PostgreSQL Logs to CloudWatch Logs](#)” (p. 943)

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: AQCnDYZdzEIH///////////
wEa0AIXLhngC5zp9CyB1R6abwKrXHVR5efnAVN3XvR7IwqKYalFSn6UyJuEFTft9nObglx4QJ+GXV9cpACKETq=
X-Amz-Date: 20140903T233749Z
X-Amz-Algorithm: AWS4-HMAC-SHA256
```

```
X-Amz-Credential: AKIADQKE4SARGYLE/20140903/us-west-2/rds/aws4_request
X-Amz-SignedHeaders: host
X-Amz-Content-SHA256: e3b0c44298fc1c229afbf4c8996fb92427ae41e4649b934de495991b7852b855
X-Amz-Expires: 86400
X-Amz-Signature: 353a4f14b3f250142d9afc34f9f9948154d46ce7d4ec091d0cdabbcf8b40c558
```

If you specify a nonexistent DB instance, the response consists of the following error:

- `DBInstanceNotFound`—*DB Instance Identifier* does not refer to an existing DB instance. (HTTP status code: 404)

MySQL Database Log Files

You can monitor the MySQL error log, slow query log, and the general log. The MySQL error log is generated by default; you can generate the slow query and general logs by setting parameters in your DB parameter group. Amazon RDS rotates all of the MySQL log files; the intervals for each type are given following.

You can monitor the 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 [Amazon Aurora Database Log Files \(p. 563\)](#).

Accessing MySQL Error Logs

The MySQL error log is written to the `mysql-error.log` file. You can view `mysql-error.log` by using the Amazon RDS console or by retrieving the log using the Amazon RDS API, Amazon RDS CLI, or AWS SDKs. `mysql-error.log` is flushed every 5 minutes, and its contents are appended to `mysql-error-running.log`. The `mysql-error-running.log` file is then rotated every hour and the hourly files generated during the last 30 days are retained. Note that the retention period is different between Amazon RDS and Aurora.

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.

MySQL writes to the error log only on startup, shutdown, and when it encounters errors. A DB instance can go hours or days without new entries being written to the error log. If you see no recent entries, it's because the server did not encounter an error that would result in a log entry.

Accessing the MySQL Slow Query and General Logs

The MySQL slow query log and the general log can be written to a file or a database table by setting parameters in your DB parameter group. For information about creating and modifying a DB parameter group, see [Working with DB Parameter Groups and DB Cluster Parameter Groups \(p. 286\)](#). 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 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 execution 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 execution time exceeds the `long_query_time` value are logged. For example, setting `long_query_time` to 0.1 prevents any query that runs for less than 100 milliseconds from being logged.
- `log_queries_not_using_indexes`: To log all queries that do not use an index to the slow query log, set to 1. The default is 0. Queries that do not use an index are logged even if their execution time is less than the value of the `long_query_time` parameter.
- `log_output option`: You can specify one of the following options for the `log_output` parameter.
 - **TABLE** (default)– Write general queries to the `mysql.general_log` table, and slow queries to the `mysql.slow_log` table.
 - **FILE**– Write both general and slow query logs to the file system. Log files are rotated hourly.

- **NONE**—Disable logging.

When logging is enabled, Amazon RDS rotates table logs or deletes log files at regular intervals. This measure is a precaution to reduce the possibility of a large log file either blocking database use or affecting performance. **FILE** and **TABLE** logging approach rotation and deletion as follows:

- When **FILE** logging is enabled, log files are examined every hour and log files older than 24 hours are deleted. In some cases, the remaining combined log file size after the deletion might exceed the threshold of 2 percent of a DB instance's allocated space. In these cases, the largest log files are deleted until the log file size no longer exceeds the threshold.
- When **TABLE** logging is enabled, in some cases log tables are rotated every 24 hours. This rotation occurs if the space used by the table logs is more than 20 percent of the allocated storage space or the size of all logs combined is greater than 10 GB. If the amount of space used for a DB instance is greater than 90 percent of the DB instance's allocated storage space, then the thresholds for log rotation are reduced. Log tables are then rotated if the space used by the table logs is more than 10 percent of the allocated storage space or the size of all logs combined is greater than 5 GB. You can subscribe to the `low_free_storage` event to be notified when log tables are rotated to free up space. For more information, see [Using Amazon RDS Event Notification \(p. 543\)](#).

When log tables are rotated, the current log table is copied to a backup log table and the entries in the current log table are removed. If the backup log table already exists, then it is deleted before the current log table is copied to the backup. You can query the backup log table if needed. The backup log table for the `mysql.general_log` table is named `mysql.general_log_backup`. The backup log table for the `mysql.slow_log` table is named `mysql.slow_log_backup`.

You can rotate the `mysql.general_log` table by calling the `mysql.rds_rotate_general_log` procedure. You can rotate the `mysql.slow_log` table by calling the `mysql.rds_rotate_slow_log` procedure.

Table logs are rotated during a database version upgrade.

To work with the logs from the Amazon RDS console, Amazon RDS API, Amazon RDS CLI, or AWS SDKs, set the `log_output` parameter to **FILE**. Like the MySQL error log, these log files are rotated hourly. The log files that were generated during the previous 72 hours are retained. Note that the retention period is different between Amazon RDS and Aurora.

For more information about the slow query and general logs, go to the following topics in the MySQL documentation:

- [The Slow Query Log](#)
- [The General Query Log](#)

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 \(p. 759\)](#).

Log File Size

The MySQL slow query log, error log, and the general log file sizes are constrained to no more than 2 percent of the allocated storage space for a DB instance. To maintain this threshold, logs are automatically rotated every hour and log files older than 24 hours are removed. If the combined log file

size exceeds the threshold after removing old log files, then the largest log files are deleted until the log file size no longer exceeds the threshold.

For 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 MySQL DB instance is 10 times larger than the largest BLOB data size found in your tables, plus the length of other variable length fields (`VARCHAR`, `VARBINARY`, `TEXT`) in the same tables. For information on how to set parameter values, see [Working with DB Parameter Groups and DB Cluster Parameter Groups \(p. 286\)](#). For information on the redo log BLOB size limit, go to [Changes in MySQL 5.6.20](#).

Managing Table-Based 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 RDS does not 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.

Binary Logging Format

MySQL on Amazon RDS supports the *row-based*, *statement-based*, and *mixed* binary logging formats for MySQL version 5.6 and later. The default binary logging format is mixed. For DB instances running MySQL versions 5.1 and 5.5, only mixed binary logging is supported. For details on the different 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 instance and can increase the amount of time to perform a restore operation of a DB instance.

Statement-based replication can cause inconsistencies between the source DB instance 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 instance you want to modify.

You can't modify a default parameter group. If the DB instance is using a default parameter group, create a new parameter group and associate it with the DB instance.

For more information on DB parameter groups, see [Working with DB Parameter Groups and DB Cluster Parameter Groups \(p. 286\)](#).

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**).
6. Choose **Save changes** to save the updates to the DB parameter group.

Important

Changing the `default.mysql5.6`, `default.mysql5.7`, or `default.mysql8.0` DB parameter group affects all MySQL version DB instances that use that parameter group. If you want to specify different binary logging formats for different MySQL 5.6, 5.7, or 8.0 DB instances in an AWS Region, you need to create your own DB parameter group. This parameter group identifies the different logging format and assigns that DB parameter group to the intended DB instances.

Accessing MySQL Binary Logs

You can use the `mysqlbinlog` utility to download or stream binary logs from Amazon RDS instances running MySQL 5.6 or later. The binary log is downloaded to your local computer, where you can perform actions such as replaying the log using the `mysql` utility. For more information about using the `mysqlbinlog` utility, go to [Using mysqlbinlog to Back Up Binary Log Files](#).

To run the `mysqlbinlog` utility against an Amazon RDS instance, use the following options:

- Specify the `--read-from-remote-server` option.
- `--host`: Specify the DNS name from the endpoint of the instance.
- `--port`: Specify the port used by the instance.
- `--user`: Specify a MySQL user that has been granted the replication slave permission.
- `--password`: Specify the password for the user, or omit a password value so that the utility prompts you for a password.
- To have the file downloaded in binary format, specify the `--raw` option.
- `--result-file`: Specify the local file to receive the raw output.
- Specify the names of one or more binary log files. To get a list of the available logs, use the SQL command `SHOW BINARY LOGS`.
- To stream the binary log files, specify the `--stop-never` option.

For more information about `mysqlbinlog` options, go to [mysqlbinlog - Utility for Processing Binary Log Files](#).

For example, see the following.

For Linux, OS X, or Unix:

```
mysqlbinlog \
  --read-from-remote-server \
  --host=MySQL56Instance1.cg034hpkmmt.region.rds.amazonaws.com \
  --port=3306 \
  --user ReplUser \
  --password \
```

```
--raw \
--result-file=/tmp/ \
binlog.00098
```

For Windows:

```
mysqlbinlog ^
--read-from-remote-server ^
--host=MySQL56Instance1.cg034hpkmmt.region.rds.amazonaws.com ^
--port=3306 ^
--user ReplUser ^
--password ^
--raw ^
--result-file=/tmp/ ^
binlog.00098
```

Amazon RDS normally purges a binary log as soon as possible, but the binary log must still be available on the instance to be accessed by mysqlbinlog. To specify the number of hours for RDS to retain binary logs, use the `mysql.rds_set_configuration` stored procedure and specify a period with enough time for you to download the logs. After you set the retention period, monitor storage usage for the DB instance to ensure that the retained binary logs don't take up too much storage.

Note

The `mysql.rds_set_configuration` stored procedure is only available for MySQL version 5.6 or later.

The following example sets the retention period to 1 day.

```
call mysql.rds_set_configuration('binlog retention hours', 24);
```

To display the current setting, use the `mysql.rds_show_configuration` stored procedure.

```
call mysql.rds_show_configuration;
```

PostgreSQL Database Log Files

RDS PostgreSQL generates query and error logs. We write auto-vacuum information and rds_admin actions to the error log. PostgreSQL also logs connections, disconnections, and checkpoints to the error log. For more information, see the [Error Reporting and Logging](#) in the PostgreSQL documentation.

You can set the retention period for system logs using the `rds.log_retention_period` parameter in the DB parameter group associated with your DB instance. The unit for this parameter is minutes. For example, a setting of 1440 would retain logs for one day. The default value is 4320 (three days). The maximum value is 10080 (seven days). Note that your instance must have enough allocated storage to contain the retained log files.

You can enable query logging for your PostgreSQL DB instance by setting 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. We recommend setting this parameter to `all` to log all statements when debugging issues in your DB instance. The default value is `none`. Alternatively, you can set this value to `ddl` to log all data definition language (DDL) statements (CREATE, ALTER, DROP, and so on) or to `mod` to log all DDL and data modification language (DML) statements (INSERT, UPDATE, DELETE, and so on).

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 minus 1 (-1) by default. Enabling this parameter can help you find unoptimized queries.

If you are new to setting parameters in a DB parameter group and associating that parameter group with a DB instance, see [Working with DB Parameter Groups and DB Cluster Parameter Groups \(p. 286\)](#)

The following steps show how to set up query logging:

1. Set the `log_statement` parameter to `all`. The following example shows the information that is written to the `postgres.log` file:

```
2013-11-05 16:48:56 UTC::@[2952]:LOG: received SIGHUP, reloading configuration files
2013-11-05 16:48:56 UTC::@[2952]:LOG: parameter "log_statement" changed to "all"
```

Additional information is written to the `postgres.log` file when you execute 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 `postgres.log` file when the parameter is set to 1:

```
2013-11-05 16:48:56 UTC::@[2952]:LOG: received SIGHUP, reloading configuration files
```

```
2013-11-05 16:48:56 UTC::@[2952]:LOG: parameter "log_min_duration_statement" changed to  
"1"
```

Additional information is written to the `postgres.log` file when you execute 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.reltablespace  
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
```

Logging Amazon RDS API Calls with AWS CloudTrail

Amazon RDS is integrated with AWS CloudTrail, a service that provides a record of actions taken by a user, role, or an AWS service in Amazon RDS. CloudTrail captures all API calls for Amazon RDS as events, including calls from the Amazon RDS console and from code calls to the Amazon RDS APIs. If you create a trail, you can enable continuous delivery of CloudTrail events to an Amazon S3 bucket, including events for Amazon RDS. If you don't configure a trail, you can still view the most recent events in the CloudTrail console in **Event history**. Using the information collected by CloudTrail, you can determine the request that was made to Amazon RDS, the IP address from which the request was made, who made the request, when it was made, and additional details.

To learn more about CloudTrail, see the [AWS CloudTrail User Guide](#).

Amazon RDS Information in CloudTrail

CloudTrail is enabled on your AWS account when you create the account. When activity occurs in Amazon RDS, 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](#).

For an ongoing record of events in your AWS account, including events for Amazon RDS, 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 regions. The trail logs events from all 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:

- [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 Amazon RDS actions are logged by CloudTrail and are documented in the [Amazon RDS API Reference](#). For example, calls to the `CreateDBInstance`, `ModifyDBInstance`, and `CreateDBParameterGroup` actions 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 Amazon RDS 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 are not an ordered stack trace of the public API calls, so they do not appear in any specific order.

The following example shows a CloudTrail log entry that demonstrates the `CreateDBInstance` action.

```
{
    "eventVersion": "1.04",
    "userIdentity": {
        "type": "IAMUser",
        "principalId": "AKIAIOSFODNN7EXAMPLE",
        "arn": "arn:aws:iam::123456789012:user/johndoe",
        "accountId": "123456789012",
        "accessKeyId": "AKIAI44QH8DHBEXAMPLE",
        "userName": "johndoe"
    },
    "eventTime": "2018-07-30T22:14:06Z",
    "eventSource": "rds.amazonaws.com",
    "eventName": "CreateDBInstance",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "192.0.2.0",
    "userAgent": "aws-cli/1.15.42 Python/3.6.1 Darwin/17.7.0 botocore/1.10.42",
    "requestParameters": {
        "enableCloudwatchLogsExports": [
            "audit",
            "error",
            "general",
            "slowquery"
        ],
        "dBInstanceIdentifier": "test-instance",
        "engine": "mysql",
        "masterUsername": "myawsuser",
        "allocatedStorage": 20,
        "dBInstanceClass": "db.m1.small",
        "masterUserPassword": "*****"
    },
    "responseElements": {
        "dBInstanceArn": "arn:aws:rds:us-east-1:123456789012:db:test-instance",
        "storageEncrypted": false,
        "preferredBackupWindow": "10:27-10:57",
        "preferredMaintenanceWindow": "sat:05:47-sat:06:17",
        "backupRetentionPeriod": 1,
        "allocatedStorage": 20,
        "storageType": "standard",
        "engineVersion": "5.6.39",
        "dBInstancePort": 0,
        "optionGroupMemberships": [
            {
                "status": "in-sync",
                "optionGroupName": "default:mysql-5-6"
            }
        ],
        "dBParameterGroups": [
            {
                "dBParameterGroupName": "default.mysql5.6",
                "parameterApplyStatus": "in-sync"
            }
        ],
        "monitoringInterval": 0,
        "dBInstanceClass": "db.m1.small",
        "readReplicaDBInstanceIdentifiers": [],
        "dBSubnetGroup": {
            "dBSubnetGroupName": "default",
            "dBSubnetGroupDescription": "default",
            "subnets": [

```

```

        },
        "subnetAvailabilityZone": {"name": "us-east-1b"},
        "subnetIdentifier": "subnet-cbfff283",
        "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"
}
```

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 \(p. 578\)](#)
- [Security with Amazon Aurora MySQL \(p. 581\)](#)
- [Updating Applications to Connect to Aurora MySQL DB Clusters Using New SSL/TLS Certificates \(p. 583\)](#)
- [Migrating Data to an Amazon Aurora MySQL DB Cluster \(p. 587\)](#)
- [Managing Amazon Aurora MySQL \(p. 623\)](#)
- [Working with Parallel Query for Amazon Aurora MySQL \(p. 644\)](#)
- [Using Advanced Auditing with an Amazon Aurora MySQL DB Cluster \(p. 669\)](#)
- [Single-Master Replication with Amazon Aurora MySQL \(p. 672\)](#)
- [Working with Aurora Multi-Master Clusters \(p. 702\)](#)
- [Integrating Amazon Aurora MySQL with Other AWS Services \(p. 727\)](#)
- [Amazon Aurora MySQL Lab Mode \(p. 762\)](#)
- [Best Practices with Amazon Aurora MySQL \(p. 763\)](#)
- [Amazon Aurora MySQL Reference \(p. 773\)](#)
- [Database Engine Updates for Amazon Aurora MySQL \(p. 794\)](#)

Overview of Amazon Aurora MySQL

The following sections provide an overview of Amazon Aurora MySQL.

Topics

- [Amazon Aurora MySQL Performance Enhancements \(p. 578\)](#)
- [Amazon Aurora MySQL and Spatial Data \(p. 579\)](#)
- [Comparison of Aurora MySQL 5.6 and Aurora MySQL 5.7 \(p. 580\)](#)
- [Comparison of Aurora MySQL 5.7 and MySQL 5.7 \(p. 580\)](#)

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.
- Aurora MySQL 2.x supports the same [spatial data types](#) and [spatial relation functions](#) as MySQL 5.7.
- Aurora MySQL 1.x and 2.x both support 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. Both Aurora MySQL 1.x and 2.x use 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);
```

Comparison of Aurora MySQL 5.6 and Aurora MySQL 5.7

The following Amazon Aurora MySQL features are supported in Aurora MySQL 5.6, but these features are currently not supported in Aurora MySQL 5.7.

- Asynchronous key prefetch (AKP). For more information, see [Working with Asynchronous Key Prefetch in Amazon Aurora \(p. 765\)](#).
- Hash joins. For more information, see [Working with Hash Joins in Aurora MySQL \(p. 771\)](#).
- Native functions for synchronously invoking AWS Lambda functions. You can asynchronously invoke AWS Lambda functions from Aurora MySQL 5.7. For more information, see [Invoking a Lambda Function with an Aurora MySQL Native Function \(p. 753\)](#).
- Scan batching. For more information, see [Aurora MySQL Database Engine Updates 2017-12-11 \(p. 846\)](#).
- Migrating data from MySQL using an Amazon S3 bucket. For more information, see [Migrating Data from MySQL by Using an Amazon S3 Bucket \(p. 590\)](#).

Currently, Aurora MySQL 5.7 does not support features added in Aurora MySQL version 1.16 and later. For information about Aurora MySQL version 1.16, see [Aurora MySQL Database Engine Updates 2017-12-11 \(p. 846\)](#).

The performance schema isn't available for early release of Aurora MySQL 5.7. Upgrade to Aurora 2.03 or higher for performance schema support.

Comparison of Aurora MySQL 5.7 and MySQL 5.7

The following features are supported in MySQL 5.7.12 but are currently not supported in Aurora MySQL 5.7:

- Global transaction identifiers (GTIDs)
- Group replication plugin
- Increased page size
- InnoDB buffer pool loading at startup
- InnoDB full-text parser plugin
- Multisource replication
- Online buffer pool resizing
- Password validation plugin
- Query rewrite plugins
- Replication filtering
- The `CREATE TABLESPACE` SQL statement
- X Protocol

For more information about these features, see the [MySQL 5.7 documentation](#).

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 IAM account must have IAM policies that grant the permissions required to perform Amazon RDS management operations. For more information, see [Identity and Access Management in Amazon Aurora \(p. 191\)](#).

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

- Aurora MySQL DB clusters must be created in an Amazon Virtual Private Cloud (VPC). To control which devices and Amazon EC2 instances can open connections to the endpoint and port of the DB instance for Aurora MySQL DB clusters in a VPC, you use a VPC security group. These endpoint and port connections can be made using Secure Sockets Layer (SSL). In addition, firewall rules at your company can control whether devices running at your company can open connections to a DB instance. For more information on VPCs, see [Amazon Virtual Private Cloud VPCs and Amazon Aurora \(p. 239\)](#).

The supported VPC tenancy depends on the 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. Aurora MySQL supports the following VPC tenancy based on the instance class:

- The db.r3 instance classes support both default and dedicated VPC tenancy.
- The db.r4 instance classes support default and dedicated VPC tenancy only.
- The db.t2 instance classes support default VPC tenancy only.

For more information about instance classes, see [Choosing the DB Instance Class \(p. 76\)](#). 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 \(p. 207\)](#).

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 with Aurora MySQL DB Clusters

Amazon Aurora MySQL DB clusters support Secure Sockets Layer (SSL) connections from applications using the same process and public key as Amazon RDS MySQL DB instances.

Amazon RDS creates an SSL 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 certificate includes the DB instance endpoint as the Common Name (CN) for the SSL 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 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 \(p. 177\)](#).

Aurora MySQL 5.6 supports Transport Layer Security (TLS) version 1.0. Aurora MySQL 5.7 supports TLS version 1.0, 1.1, and 1.2.

We recommend the MariaDB Connector/J client as a client that supports SAN with SSL. For more information, see the [MariaDB Connector/J download](#) page.

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 later:

```
mysql -h myinstance.c9akciq32.rds-us-east-1.amazonaws.com  
--ssl-ca=[full path]rds-combined-ca-bundle.pem --ssl-mode=VERIFY_IDENTITY
```

For MySQL 5.6 and earlier:

```
mysql -h myinstance.c9akciq32.rds-us-east-1.amazonaws.com  
--ssl-ca=[full path]rds-combined-ca-bundle.pem --ssl-verify-server-cert
```

You can require SSL connections for specific users accounts. For example, you can use one of the following statements, depending on your MySQL version, to require SSL connections on the user account encrypted_user.

For MySQL 5.7 and later:

```
ALTER USER 'encrypted_user'@'%' REQUIRE SSL;
```

For MySQL 5.6 and earlier:

```
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 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 [Managing Connections with Amazon RDS Proxy \(Preview\) \(p. 308\)](#).

Note

For more information on SSL connections with MySQL, see the [MySQL documentation](#).

Updating Applications to Connect to Aurora MySQL DB Clusters Using New SSL/TLS Certificates

As of September 19, 2019, Amazon RDS published new Certificate Authority (CA) certificates for connecting to your Aurora DB cluster using Secure Socket Layer or Transport Layer Security (SSL/TLS). The previous CA certificates expire on March 5, 2020. Following, you can find information about updating your applications to use the new certificates. If your application connects to an Aurora DB cluster using SSL/TLS, you must take the following steps before **March 5, 2020**. Doing this means that you can avoid interruption of connectivity between your applications and your Aurora DB clusters.

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 \(p. 179\)](#). For information about using SSL/TLS with Aurora MySQL DB clusters, see [Using SSL with Aurora MySQL DB Clusters \(p. 582\)](#).

Topics

- [Determining Whether any Applications are Connecting to Your Aurora MySQL DB Cluster Using SSL \(p. 584\)](#)
- [Determining Whether a Client Requires Certificate Verification to Connect \(p. 585\)](#)
- [Updating Your Application Trust Store \(p. 585\)](#)
- [Example Java Code for Establishing SSL Connections \(p. 587\)](#)

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);
```

MySQL

The following examples with the MySQL Client show two ways to check a script's MySQL connection to determine whether successful connections require a valid certificate. For more information on all of the connection options with the MySQL Client, see [Client-Side Configuration for Encrypted Connections](#) in the MySQL documentation.

When using the MySQL 5.7 or MySQL 8.0 Client, an SSL connection requires verification against the server CA certificate if for the `--ssl-mode` option you specify `VERIFY_CA` or `VERIFY_IDENTITY`, as in the following example.

```
mysql -h mysql-database.rds.amazonaws.com -uadmin -ppassword --ssl-ca=/tmp/ssl-cert.pem --
ssl-mode=VERIFY_CA
```

When using the MySQL 5.6 Client, an SSL connection requires verification against the server CA certificate if you specify the `--ssl-verify-server-cert` option, as in the following example.

```
mysql -h mysql-database.rds.amazonaws.com -uadmin -ppassword --ssl-ca=/tmp/ssl-cert.pem --
ssl-verify-server-cert
```

Updating Your Application Trust Store

For information about updating the trust store for MySQL applications, see [Installing SSL Certificates](#) in the MySQL documentation.

Note

When you update the trust store, you can retain older certificates in addition to adding the new certificates.

Updating Your Application Trust Store for JDBC

You can update the trust store for applications that use JDBC for SSL/TLS connections.

To update the trust store for JDBC applications

1. Download the 2019 root certificate that works for all AWS Regions and put the file in the trust store directory.

For information about downloading the root certificate, see [Using SSL/TLS to Encrypt a Connection to a DB Cluster \(p. 177\)](#).

2. Convert the certificate to .der format using the following command.

```
openssl x509 -outform der -in rds-ca-2019-root.pem -out rds-ca-2019-root.der
```

Replace the file name with the one that you downloaded.

3. Import the certificate into the key store using the following command.

```
keytool -import -alias rds-root -keystore clientkeystore -file rds-ca-2019-root.der
```

4. Confirm that the key store was updated successfully.

```
keytool -list -v -keystore clientkeystore.jks
```

Enter the metastore password when you are prompted for it.

Your output should contain the following.

```
rds-root, date, trustedCertEntry,  
Certificate fingerprint (SHA1):  
D4:0D:DB:29:E3:75:D0:FF:A6:71:C3:14:0B:BF:5F:47:8D:1C:80:96  
# This fingerprint should match the output from the below command  
openssl x509 -fingerprint -in rds-ca-2019-root.pem -noout
```

If you are using the mysql JDBC driver in an application, set the following properties in the application.

```
System.setProperty("javax.net.ssl.trustStore", certs);  
System.setProperty("javax.net.ssl.trustStorePassword", "password");
```

When you start the application, set the following properties.

```
java -Djavax.net.ssl.trustStore=/path_to_truststore/MyTruststore.jks -  
Djavax.net.ssl.trustStorePassword=my_truststore_password com.companyName.MyApplication
```

Example Java Code for Establishing SSL Connections

The following code example shows how to set up the SSL connection that validates the server certificate using JDBC.

```
public class MySQLSSLTest {  
  
    private static final String DB_USER = "user name";  
    private static final String DB_PASSWORD = "password";  
    // This key store has only the prod root ca.  
    private static final String KEY_STORE_FILE_PATH = "file-path-to-keystore";  
    private static final String KEY_STORE_PASS = "keystore-password";  
  
    public static void test(String[] args) throws Exception {  
        Class.forName("com.mysql.jdbc.Driver");  
  
        System.setProperty("javax.net.ssl.trustStore", KEY_STORE_FILE_PATH);  
        System.setProperty("javax.net.ssl.trustStorePassword", KEY_STORE_PASS);  
  
        Properties properties = new Properties();  
        properties.setProperty("sslMode", "VERIFY_IDENTITY");  
        properties.put("user", DB_USER);  
        properties.put("password", DB_PASSWORD);  
  
        Connection connection = DriverManager.getConnection("jdbc:mysql://jagdeeps-ssl-  
test.cni62e2e7kwh.us-east-1.rds.amazonaws.com:3306", properties);  
        Statement stmt=connection.createStatement();  
  
        ResultSet rs=stmt.executeQuery("SELECT 1 from dual");  
  
        return;  
    }  
}
```

Important

After you have determined that your database connections use SSL/TLS and have updated your application trust store, you can update your database to use the rds-ca-2019 certificates. For instructions, see step 3 in [Updating Your CA Certificate by Modifying Your DB Instance \(p. 180\)](#).

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` must be configured with the default data file name "ibdata1". 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 Amazon RDS MySQL DB instance	Physical	You can migrate from an RDS MySQL DB instance by first creating an Aurora MySQL Read Replica of a MySQL DB instance. When the replica lag between the MySQL DB instance and the Aurora MySQL Read Replica is 0, you can direct your client applications to read from the Aurora Read Replica and then stop replication to make the Aurora MySQL Read Replica a standalone Aurora MySQL DB cluster for reading and writing. For details, see Migrating Data from a MySQL DB Instance to an Amazon Aurora MySQL DB Cluster by Using an Aurora Read Replica (p. 614) .
An RDS MySQL DB snapshot	Physical	You can migrate data directly from an Amazon RDS MySQL DB snapshot to an Amazon Aurora MySQL DB cluster.

Migrating From	Migration Type	Solution
		For details, see Migrating Data from a MySQL DB Instance to an Amazon Aurora MySQL DB Cluster by Using a DB Snapshot (p. 607) .
A MySQL database external to Amazon RDS	Logical	You can create a dump of your data using the <code>mysqldump</code> utility, and then import that data into an existing Amazon Aurora MySQL DB cluster. For details, see Migrating from MySQL to Amazon Aurora by Using mysqldump (p. 607) .
A MySQL database external to Amazon RDS	Physical	You can copy the backup files from your database to an Amazon Simple Storage Service (Amazon S3) bucket, and then restore an Amazon Aurora MySQL DB cluster from those files. This option can be considerably faster than migrating data using <code>mysqldump</code> . For details, see Migrating Data from MySQL by Using an Amazon S3 Bucket (p. 590) .
A MySQL database external to Amazon RDS	Logical	You can save data from your database as text files and copy those files to an Amazon S3 bucket. You can then load that data into an existing Aurora MySQL DB cluster using the <code>LOAD DATA FROM S3</code> MySQL command. For more information, see Loading Data into an Amazon Aurora MySQL DB Cluster from Text Files in an Amazon S3 Bucket (p. 739) .
A database that is not MySQL-compatible	Logical	You can use AWS Database Migration Service (AWS DMS) to migrate data from a database that is not MySQL-compatible. For more information on AWS DMS, see What Is AWS Database Migration Service?

Note

- If you are migrating a MySQL database external to Amazon RDS, the migration options described in the table are supported only if your database supports the InnoDB or MyISAM tablespaces.
- If the MySQL database you are migrating to Aurora MySQL uses memcached, remove memcached before migrating it.

Migrating Data from an External MySQL Database to an Amazon Aurora MySQL DB Cluster

If your database supports the InnoDB or MyISAM tablespaces, you have these options for migrating your data to an Amazon Aurora MySQL DB cluster:

- You can create a dump of your data using the `mysqldump` utility, and then import that data into an existing Amazon Aurora MySQL DB cluster. For more information, see [Migrating from MySQL to Amazon Aurora by Using mysqldump \(p. 607\)](#).
- 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 \(p. 590\)](#).

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 \(p. 83\)](#).

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.

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 \(p. 591\)](#)
- [Backing Up Files to be Restored as an Amazon Aurora MySQL DB Cluster \(p. 592\)](#)
- [Restoring an Amazon Aurora MySQL DB Cluster from an Amazon S3 Bucket \(p. 594\)](#)
- [Synchronizing the Amazon Aurora MySQL DB Cluster with the MySQL Database Using Replication \(p. 602\)](#)

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 Amazon RDS 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.
- Amazon RDS 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 Amazon RDS 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 Amazon RDS 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 KMS keys used for encrypting the backups. For instructions, see [Creating an IAM Policy to Access AWS KMS Resources \(p. 733\)](#).

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

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 Amazon RDS creates the IAM service role required for Amazon RDS 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 Amazon RDS to access Amazon S3

1. Complete the steps in [Creating an IAM Policy to Access Amazon S3 Resources \(p. 729\)](#).
2. Complete the steps in [Creating an IAM Role to Allow Amazon Aurora to Access AWS Services \(p. 734\)](#).
3. Complete the steps in [Associating an IAM Role with an Amazon Aurora MySQL DB Cluster \(p. 734\)](#).

Backing Up Files to be Restored as an Amazon Aurora MySQL DB Cluster

You can create a full backup of your MySQL database files using Percona XtraBackup and upload the backup files to an Amazon S3 bucket. Alternatively, if you already use Percona XtraBackup to back up your MySQL database files, you can upload your existing full and incremental backup directories and files to an Amazon S3 bucket.

Creating a Full Backup With Percona XtraBackup

To create a full backup of your MySQL database files that can be restored from Amazon S3 to create an Amazon Aurora MySQL DB cluster, use the Percona XtraBackup utility (`xtrabackup`) to back up your database.

For example, the following command creates a backup of a MySQL database and stores the files in the `/on-premises/s3-restore/backup` folder.

```
xtrabackup --backup --user=<myuser> --password=<password> --target-dir=</on-premises/s3-restore/backup>
```

If you want to compress your backup into a single file (which can be split, if needed), you can use the `--stream` option to save your backup in one of the following formats:

- Gzip (.gz)
- tar (.tar)
- Percona xbstream (.xbstream)

The following command creates a backup of your MySQL database split into multiple Gzip files.

```
xtrabackup --backup --user=<myuser> --password=<password> --stream=tar \  
--target-dir=</on-premises/s3-restore/backup> | gzip - | split -d --bytes=500MB \  
- </on-premises/s3-restore/backup/backup>.tar.gz
```

The following command creates a backup of your MySQL database split into multiple tar files.

```
xtrabackup --backup --user=<myuser> --password=<password> --stream=tar \  
--target-dir=</on-premises/s3-restore/backup> | split -d --bytes=500MB \  
- </on-premises/s3-restore/backup/backup>.tar
```

The following command creates a backup of your MySQL database split into multiple xbstream files.

```
xtrabackup --backup --user=<myuser> --password=<password> --stream=xbstream \  
--target-dir=</on-premises/s3-restore/backup> | split -d --bytes=500MB \  
- </on-premises/s3-restore/backup/backup>.xbstream
```

Once you have backed up your MySQL database using the Percona XtraBackup utility, you can copy your backup directories and files to an Amazon S3 bucket.

For information on creating and uploading a file to an Amazon S3 bucket, see [Getting Started with Amazon Simple Storage Service](#) in the [Amazon S3 Getting Started Guide](#).

Using Incremental Backups With Percona XtraBackup

Amazon Aurora MySQL supports both full and incremental backups created using Percona XtraBackup. If you already use Percona XtraBackup to perform full and incremental backups of your MySQL database files, you don't need to create a full backup and upload the backup files to Amazon S3. Instead, you can save a significant amount of time by copying your existing backup directories and files for your full and incremental backups to an Amazon S3 bucket. For more information about creating incremental backups using Percona XtraBackup, see [Incremental Backup](#).

When copying your existing full and incremental backup files to an Amazon S3 bucket, you must recursively copy the contents of the base directory. Those contents include the full backup and also all incremental backup directories and files. This copy must preserve the directory structure in the Amazon S3 bucket. Aurora iterates through all files and directories. Aurora uses the `xtrabackup-checkpoints` file included with each incremental backup to identify the base directory and to order incremental backups by log sequence number (LSN) range.

For information on creating and uploading a file to an Amazon S3 bucket, see [Getting Started with Amazon Simple Storage Service](#) in the [Amazon S3 Getting Started Guide](#).

Backup Considerations

When you upload a file to an Amazon S3 bucket, you can use server-side encryption to encrypt the data. You can then restore an Amazon Aurora MySQL DB cluster from those encrypted files. Amazon Aurora MySQL can restore a DB cluster with files encrypted using the following types of server-side encryption:

- Server-side encryption with Amazon S3-managed keys (SSE-S3) – Each object is encrypted with a unique key employing strong multifactor encryption.

- Server-side encryption with AWS KMS–managed keys (SSE-KMS) – Similar to SSE-S3, but you have the option to create and manage encryption keys yourself, and also other differences.

For information about using server-side encryption when uploading files to an Amazon S3 bucket, see [Protecting Data Using Server-Side Encryption](#) in the *Amazon S3 Developer Guide*.

Amazon S3 limits the size of a file uploaded to an Amazon S3 bucket to 5 terabytes (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.

Amazon RDS limits the number of source files uploaded to an Amazon S3 bucket to 1 million files. In some cases, backup data for your database, including all full and incremental backups, can come to a large number of files. In these cases, use a tarball (.tar.gz) file to store full and incremental backup files in the Amazon S3 bucket.

Aurora consumes your backup files based on the file name. Be sure to name your backup files with the appropriate file extension based on the file format—for example, `.xbstream` for files stored using the Percona xbstream format.

Aurora consumes your backup files in alphabetical order and also in natural number order. Always use the `split` option when you issue the `xtrabackup` command to ensure that your backup files are written and named in the proper order.

Aurora doesn't support partial backups created using Percona XtraBackup. You can't use the following options to create a partial backup when you back up the source files for your database: `--tables`, `--tables-exclude`, `--tables-file`, `--databases`, `--databases-exclude`, or `--databases-file`.

For more information about backing up your database with Percona XtraBackup, see [Percona XtraBackup - Documentation](#) and [The xtrabackup Binary](#) on the Percona website.

Aurora supports incremental backups created using Percona XtraBackup. For more information about creating incremental backups using Percona XtraBackup, see [Incremental Backup](#).

Restoring an Amazon Aurora MySQL DB Cluster from an Amazon S3 Bucket

You can restore your backup files from your Amazon S3 bucket to create a new Amazon Aurora MySQL DB cluster by using the Amazon RDS console.

To restore an Amazon Aurora MySQL DB cluster from files on an Amazon S3 bucket

1. Sign in to the AWS Management Console and open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. In the top right corner of the Amazon RDS console, choose the AWS Region in which to create your DB cluster. Choose the same AWS Region as the Amazon S3 bucket that contains your database backup.
3. In the navigation pane, choose **Databases**, and then choose **Restore from S3**.
4. On the **Select engine** page, choose **Amazon Aurora**, choose the MySQL-compatible edition, and then choose **Next**.

The **Specify source backup details** page appears.

Specify source backup details

Source database specifications

Source engine: mysql

Source engine version: 5.6

S3 bucket Refresh

S3 bucket: - Select one -

S3 folder path prefix (optional) Info

IAM role Refresh

Create a new role Yes

5. In **Specify source backup details**, specify the following.

For This Option	Do This
Source engine	Aurora MySQL currently supports restoring from backup files only for the mysql database engine.
Source engine version	Choose the MySQL version of your source database.
S3 bucket	Choose the Amazon S3 bucket that contains your backup files.
S3 folder path prefix (optional)	Specify a file path prefix for the files stored in your Amazon S3 bucket. The S3 Bucket Prefix is optional. If you don't specify a prefix, then Aurora MySQL creates the DB cluster using all of the files and folders in the root folder of the Amazon S3 bucket. If you specify a prefix, then Aurora MySQL creates the DB cluster using the files and folders in the Amazon S3 bucket where the full path for the file begins with the specified prefix.

For This Option	Do This
	<p>Aurora doesn't traverse other subfolders in your Amazon S3 bucket looking for backup files. Only the files from the folder identified by the S3 Bucket Prefix are used. If you store your backup files in a subfolder in your Amazon S3 bucket, specify a prefix that identifies the full path to the folder where the files are stored.</p> <p>For example, suppose that you store your backup files in a subfolder of your Amazon S3 bucket named <code>backups</code>. Suppose also that you have multiple sets of backup files, each in its own directory (<code>gzip_backup1</code>, <code>gzip_backup2</code>, and so on). In this case, you specify a prefix of <code>backups/gzip_backup1</code> to restore from the files in the <code>gzip_backup1</code> folder.</p>
Create a new role	<p>Choose Yes to create a new IAM role, or No to select an existing IAM role, to authorize Aurora to access Amazon S3 on your behalf. For more information, see Required Permissions (p. 591).</p>
IAM role name	<p>This option is available only if Create a new role is set to Yes.</p> <p>Type the name of the new IAM role to be created. The new role is used to authorize Amazon Aurora to access Amazon S3 on your behalf. For more information, see Required Permissions (p. 591).</p>
IAM role	<p>This option is available only if Create a new role is set to No.</p> <p>Select the IAM role that you created to authorize Aurora to access Amazon S3 on your behalf. If you have not created an IAM role, you can instead set Create a new role to Yes to create one. For more information, see Required Permissions (p. 591).</p>
Allow access to KMS key	<p>This option is available only if Create a new role is set to Yes.</p> <p>If you didn't encrypt the backup files, choose No.</p> <p>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.</p> <p>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 master key for Master key. The AWS Management Console creates an IAM policy that enables Amazon RDS to decrypt the data.</p> <p>For more information, see Protecting Data Using Server-Side Encryption in the <i>Amazon S3 Developer Guide</i>.</p>

6. Choose **Next**.

7. On the **Specify DB details** page, specify your DB cluster information.

A typical **Specify DB details** page looks like the following.

The screenshot shows the 'Specify DB details' configuration page. It includes sections for Configuration, DB engine, Capacity type, DB engine version, DB instance class, and Multi-AZ deployment. The 'Provisioned' capacity type is selected, and the DB engine version is set to 'Aurora (MySQL)-5.6.10a'. The 'Create Replica in Different Zone' option for Multi-AZ deployment is selected.

Configuration
Estimate your monthly costs for the DB Instance using the [AWS Simple Monthly Calculator](#)

DB engine
Aurora - compatible with MySQL 5.6

Capacity type [Info](#)
 Provisioned
You provision and manage the server instance sizes.
 Provisioned with Aurora parallel query enabled [Info](#)
You provision and manage the server instance sizes, and Aurora improves the performance of analytic queries by pushing processing down to the Aurora storage layer (currently available for Aurora MySQL 5.6)
 Serverless [Info](#)
You specify the minimum and maximum of resources for a DB cluster. Aurora scales the capacity based on database load.

DB engine version [Info](#)
Aurora (MySQL)-5.6.10a

DB instance class [Info](#)
- Select one -

Multi-AZ deployment [Info](#)
 Create Replica in Different Zone
 No

The following table shows settings for a DB instance.

For This Option	Do This
Capacity type	<p>Choose Provisioned to manage the capacity for your DB instance manually. You might need to change the DB instance class for your instance if your workload changes.</p> <p>Choose Provisioned with Aurora parallel query enabled to manage the capacity for your DB instance manually. With this option, Aurora improves the performance of analytic queries by pushing processing down to the Aurora storage layer (currently available for Aurora MySQL 5.6). For more information, see Working with Parallel Query for Amazon Aurora MySQL (p. 644).</p> <p>Choose Serverless for Aurora to manage the capacity available to your DB instance automatically. For more information, see Using Amazon Aurora Serverless (p. 116).</p>
DB engine version	Applies only to the provisioned capacity type. Choose the version number of your DB engine.
DB instance class	Select 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 Choosing the DB Instance Class (p. 76) .
Multi-AZ deployment	Determine if you want to create Aurora Replicas in other Availability Zones for failover support. If you select Create Replica in Different Zone , then Amazon RDS creates an Aurora Replica for you in your DB cluster in a different Availability Zone than the primary instance for your DB cluster. For more information about multiple Availability Zones, see Choosing the Regions and Availability Zones (p. 80) .
DB instance identifier	<p>Type a name for the primary instance in your DB cluster. This identifier is used in the endpoint address for the primary instance of your DB cluster.</p> <p>The DB instance identifier has the following constraints:</p> <ul style="list-style-type: none"> • It must contain from 1 to 63 alphanumeric characters or hyphens. • Its first character must be a letter. • It cannot end with a hyphen or contain two consecutive hyphens. • It must be unique for all DB instances per AWS account, per AWS Region.
Master username	Type a name using alphanumeric characters to use as the master user name to log on to your DB cluster.
Master password	Type a password that contains from 8 to 41 printable ASCII characters (excluding /, ", and @) for your master user password.

8. Choose **Next**.

9. On the **Configure advanced settings** page, you can customize additional settings for your Aurora MySQL DB cluster. The following table shows the advanced settings for a DB cluster.

For This Option	Do This
Virtual Private Cloud (VPC)	Select the VPC to host the DB cluster. Select Create a New VPC to have Amazon RDS create a VPC for you. For more information, see DB Cluster Prerequisites (p. 100) earlier in this topic.
Subnet group	Select the DB subnet group to use for the DB cluster. For more information, see DB Cluster Prerequisites (p. 100) earlier in this topic.
Public accessibility	Select Yes to give the DB cluster a public IP address; otherwise, select No . The instances in your DB cluster can be a mix of both public and private DB instances. For more information about hiding instances from public access, see Hiding a DB Instance in a VPC from the Internet (p. 251) .
Availability zone	Determine if you want to specify a particular Availability Zone. For more information about Availability Zones, see Choosing the Regions and Availability Zones (p. 80) .
VPC security groups	Select Create new VPC security group to have Amazon RDS create a VPC security group for you. Or, select Select existing VPC security groups and specify one or more VPC security groups to secure network access to the DB cluster. For more information, see DB Cluster Prerequisites (p. 100) earlier in this topic.
DB Cluster Identifier	<p>Type a name for your DB cluster that is unique for your account in the AWS Region you selected. This identifier is used in the cluster endpoint address for your DB cluster. For information on the cluster endpoint, see Amazon Aurora Connection Management (p. 4).</p> <p>The DB cluster identifier has the following constraints:</p> <ul style="list-style-type: none"> • It must contain from 1 to 63 alphanumeric characters or hyphens. • Its first character must be a letter. • It cannot end with a hyphen or contain two consecutive hyphens. • It must be unique for all DB clusters per AWS account, per AWS Region.
Database name	<p>Type a name for your default database of up to 64 alphanumeric characters. If you don't provide a name, Amazon RDS doesn't create a database on the DB cluster you are creating.</p> <p>To create additional databases, connect to the DB cluster and use the SQL command <code>CREATE DATABASE</code>. For more information about connecting to the DB cluster, see Connecting to an Amazon Aurora DB Cluster (p. 166).</p>

For This Option	Do This
Database port	Specify the port for applications and utilities to use to access the database. Aurora MySQL DB clusters default to the default MySQL port, 3306, 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.
DB parameter group	Select a parameter group. Aurora has a default parameter group you can use, or you can create your own parameter group. For more information about parameter groups, see Working with DB Parameter Groups and DB Cluster Parameter Groups (p. 286) .
DB cluster parameter group	Select a DB cluster parameter group. Aurora has a default DB cluster parameter group you can use, or you can create your own DB cluster parameter group. For more information about DB cluster parameter groups, see Working with DB Parameter Groups and DB Cluster Parameter Groups (p. 286) .
Option group	Aurora has a default option group.
Encryption	Select Enable Encryption to enable encryption at rest for this DB cluster. For more information, see Encrypting Amazon Aurora Resources (p. 175) .
Master key	Only available if Encryption is set to Enable Encryption . Select the master key to use for encrypting this DB cluster. For more information, see Encrypting Amazon Aurora Resources (p. 175) .
Priority	Choose a failover priority for the instance. If you don't select a value, the default is tier-1 . This priority determines the order in which Aurora Replicas are promoted when recovering from a primary instance failure. For more information, see Fault Tolerance for an Aurora DB Cluster (p. 380) .
Backup retention period	Select the length of time, from 1 to 35 days, that Aurora retains backup copies of the database. Backup copies can be used for point-in-time restores (PITR) of your database down to the second.
Backtrack	Select Enable Backtrack to enable backtracking or Disable Backtrack 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 Backtracking an Aurora DB Cluster (p. 625) .

For This Option	Do This
Enhanced monitoring	Choose Enable enhanced monitoring to enable gathering metrics in real time for the operating system that your DB cluster runs on. For more information, see Enhanced Monitoring (p. 469) .
Monitoring Role	Only available if Enhanced Monitoring is set to Enable enhanced monitoring . Choose the IAM role that you created to permit Amazon RDS to communicate with Amazon CloudWatch Logs for you, or choose Default to have RDS create a role for you named <code>rds-monitoring-role</code> . For more information, see Enhanced Monitoring (p. 469) .
Granularity	Only available if Enhanced Monitoring is set to Enable enhanced monitoring . Set the interval, in seconds, between when metrics are collected for your DB cluster.
Log exports	Choose the logs that you want to start publishing to Amazon CloudWatch Logs. For more about publishing to CloudWatch Logs, see Publishing Amazon Aurora MySQL Logs to Amazon CloudWatch Logs (p. 759) .
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. 794) .
Maintenance window	Select Select window and specify the weekly time range during which system maintenance can occur. Or, select No preference for Amazon RDS to assign a period randomly.
Enable deletion protection	Select Enable deletion protection to prevent your DB cluster from being deleted. If you create a production DB cluster with the console, deletion protection is enabled by default.

10. Choose **Create database** to launch your Aurora DB instance, and then choose **Close** to close the wizard.

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 \(p. 446\)](#).

database-1

Related

Filter databases

DB identifier	Role	Engine	Region
database-1	Regional	Aurora MySQL	us-east-1
database-1-instance-1	Writer	Aurora MySQL	us-east-1

Connectivity & security Monitoring Logs & events Configuration Maintenance & backup

Endpoints (2)

Filter endpoint

Endpoint name	Status	Type
database-1.cluster-ro-xxxxxx.us-east-2.rds.amazonaws.com	Available	Read
database-1.cluster-writer-xxxxxx.us-east-2.rds.amazonaws.com	Available	Write

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. 603)

- [Synchronizing the Amazon Aurora MySQL DB Cluster with the External MySQL Database \(p. 604\)](#)

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 \(p. 604\)](#).

The following are prerequisites for using encrypted replication:

- Secure Sockets Layer (SSL) must be enabled on the external MySQL master 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.

Note

Currently, encrypted replication with an external MySQL database is only supported for Aurora MySQL version 5.6.

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 master 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 master, 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 master 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 master user using SSL.

For information about connecting to an Aurora MySQL DB cluster with SSL, see [Using SSL with Aurora MySQL DB Clusters \(p. 582\)](#).

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-----\nAAAAB3NzaC1yc2EAAAQABAAQClKsfkNkuSevGj3eyhCe53pcjqP3maAhDFcvBS706V\n-----END CERTIFICATE-----"}')
```

```
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-----
AAAAB3NzaC1yc2EAAAADAQABAAQClKsfkNkuSevGj3eYhCe53pcjqP3maAhDFcvBS706V
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-----
AAAAB3NzaC1yc2EAAAADAQABAAQClKsfkNkuSevGj3eYhCe53pcjqP3maAhDFcvBS706V
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 with Aurora MySQL DB Clusters \(p. 582\)](#).

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:

- 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)			
<input type="text"/> Filter events			
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'	

You specify the position to start replication in a later step.

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 Management 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 RDS_SQL_DB_<host_name>
```

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` 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
);
```

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);
```

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 master database. To do so, connect to the Aurora MySQL DB cluster and run the following command.

```
SHOW SLAVE STATUS;
```

In the command output, the `Seconds_Behind_Master` field shows how far the Aurora MySQL DB cluster is behind the MySQL master. When this value is 0 (zero), the Aurora MySQL DB cluster has caught up to the master, and you can move on to the next step to stop replication.

8. Connect to the MySQL replication master 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 an Amazon RDS 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 an Amazon RDS 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 Amazon RDS MySQL DB snapshot, as described following.

Topics

- [Migrating an RDS MySQL Snapshot to Aurora \(p. 608\)](#)
- [Migrating Data from a MySQL DB Instance to an Amazon Aurora MySQL DB Cluster by Using an Aurora Read Replica \(p. 614\)](#)

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 \(p. 47\)](#).

Migrating an RDS MySQL Snapshot to Aurora

You can migrate a DB snapshot of an Amazon RDS MySQL DB instance to create an Aurora MySQL DB cluster. The new Aurora MySQL DB cluster is populated with the data from the original Amazon RDS 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 \(p. 614\)](#).

Incompatibilities between MySQL and Aurora MySQL include the following:

- You can't migrate a MySQL version 5.7 snapshot to Aurora MySQL version 5.6.
- You can't migrate a DB snapshot created with MySQL 5.6.40 or 5.7.22 to Aurora MySQL.

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? \(p. 608\)](#)
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. 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: <code>alter table <schema>. <table_name> engine=innodb, algorithm=copy;</code>
Compressed tables	Aurora MySQL doesn't support compressed tables (that is, tables created with <code>ROW_FORMAT=COMPRESSED</code>). To reduce your chances of running out of space or to speed up the migration process, expand your compressed tables by setting <code>ROW_FORMAT</code> to <code>DEFAULT</code> , <code>COMPACT</code> , <code>DYNAMIC</code> , or <code>REDUNDANT</code> . For more information, see https://dev.mysql.com/doc/refman/5.6/en/innodb-row-format.html .

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',
                'help_category', 'help_keyword', 'help_relation',
                'help_topic', 'host', 'ndb_binlog_index', 'plugin',
                'proc', 'procs_priv', 'proxies_priv', 'servers', 'slow_log',
                'tables_priv', 'time_zone', 'time_zone_leap_second',
                'time_zone_name', 'time_zone_transition',
                'time_zone_transition_type', 'user'
            )
        )
        or
        (
            -- Compressed tables
            ROW_FORMAT = 'Compressed'
        );
    );


```

The script produces output similar to the output in the following example. The example shows two tables that must be converted from MyISAM to InnoDB. The output also includes the approximate size of each table in megabytes (MB).

==> MyISAM or Compressed Tables Approx size (MB)		
test.name_table		2102.25
test.my_table		65.25

```
| 2 rows in set (0.01 sec)
```

Console

You can migrate a DB snapshot of an Amazon RDS MySQL DB instance to create an Aurora MySQL DB cluster. The new Aurora MySQL DB cluster is populated with the data from the original Amazon RDS MySQL DB instance. The DB snapshot must have been made from an Amazon RDS DB instance running MySQL version 5.6 or 5.7. For information about creating a DB snapshot, see [Creating a DB Snapshot](#).

If the DB snapshot is not in the AWS Region where you want to locate your data, use the Amazon RDS console to copy the DB snapshot to that AWS Region. For information about copying a DB snapshot, see [Copying a DB Snapshot](#).

When you migrate the DB snapshot by using the AWS Management Console, the console takes the actions necessary to create both the DB cluster and the primary instance.

You can also choose for your new Aurora MySQL DB cluster to be encrypted at rest using an AWS Key Management Service (AWS KMS) encryption 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 **Instances**, 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, up to a maximum size of 64 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 \(p. 24\)](#).
 - **DB Instance Identifier:** Type a name for the DB cluster that is unique for your account in the AWS Region you selected. This identifier is used in the endpoint addresses for the instances in your DB cluster. You might choose to add some intelligence to the name, such as including the AWS Region and DB engine you selected, for example `aurora-cluster1`.

The DB instance identifier has the following constraints:

- It must contain from 1 to 63 alphanumeric characters or hyphens.

- Its first character must be a letter.
- It cannot end with a hyphen or contain two consecutive hyphens.
- It must be unique for all DB instances per AWS account, per AWS Region.
- **Virtual Private Cloud (VPC):** If you have an existing VPC, then you can use that VPC with your Aurora MySQL DB cluster by selecting your VPC identifier, for example vpc-a464d1c1. For information on using an existing VPC, see [How to Create a VPC for Use with Amazon Aurora \(p. 239\)](#).

Otherwise, you can choose to have Amazon RDS create a VPC for you by selecting **Create a new VPC**.

- **Subnet group:** If you have an existing subnet group, then you can use that subnet group with your Aurora MySQL DB cluster by selecting your subnet group identifier, for example gs-subnet-group1.

Otherwise, you can choose to have Amazon RDS 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 Amazon RDS 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 an AWS KMS encryption key as the **Master 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 \(p. 794\)](#).

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 \(p. 166\)](#).

AWS CLI

You can migrate a DB snapshot of an Amazon RDS MySQL DB instance to create an Aurora DB cluster. The new DB cluster is then populated with the data from the DB snapshot. The DB snapshot must come from an Amazon RDS DB instance running MySQL version 5.6 or 5.7. For more information, see [Creating a DB Snapshot](#).

If the DB snapshot is not in the AWS Region where you want to locate your data, copy the DB snapshot to that AWS Region. For more information, see [Copying a DB Snapshot](#).

You can create an Aurora DB cluster from a DB snapshot of an Amazon RDS 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 DB cluster, or `--engine aurora` for a MySQL 5.6-compatible DB cluster
- `--kms-key-id`

The AWS Key Management Service (AWS KMS) encryption 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, OS X, 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, OS X, 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

Amazon RDS 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 \(p. 607\)](#).

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 MySQL DB instance that is the replication master. If you encounter an error, you can find help in the [Amazon RDS Community Forum](#) or by contacting AWS Support.

For more information on MySQL Read Replicas, see [Working with Read Replicas of MariaDB, MySQL, and PostgreSQL DB Instances](#).

Creating an Aurora Read Replica

You can create an Aurora Read Replica for a MySQL DB instance by using the console or the AWS CLI.

Console

To create an Aurora Read Replica from a source MySQL DB instance

1. Sign in to the AWS Management Console and open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. In the navigation pane, choose **Databases**.
3. Choose the MySQL DB instance that you want to use as the source for your Aurora Read Replica.
4. For **Actions**, choose **Create Aurora read replica**.
5. Choose the DB cluster specifications you want to use for the Aurora Read Replica, as described in the following table.

Option	Description
DB instance class	Choose a DB instance class that defines the processing and memory requirements for the primary instance in the DB cluster. For more information about DB instance class options, see Choosing the DB Instance Class (p. 76) .
Multi-AZ deployment	Choose Create Replica in Different Zone to create a standby replica of the new DB cluster in another Availability Zone in the target AWS Region for failover support. For more information about multiple Availability Zones, see Choosing the Regions and Availability Zones (p. 80) .
DB instance identifier	Type a name for the primary instance in your Aurora Read Replica DB cluster. This identifier is used in the endpoint address for the primary instance of the new DB cluster. The DB instance identifier has the following constraints: <ul style="list-style-type: none">• It must contain from 1 to 63 alphanumeric characters or hyphens.• Its first character must be a letter.• It cannot end with a hyphen or contain two consecutive hyphens.• It must be unique for all DB instances for each AWS account, for each AWS Region. Because the Aurora Read Replica DB cluster is created from a snapshot of the source DB instance, the master user name and master password for the Aurora Read Replica are the same as the master user name and master password for the source DB instance.
Virtual Private Cloud (VPC)	Select the VPC to host the DB cluster. Select Create new VPC to have Amazon RDS create a VPC for you. For more information, see DB Cluster Prerequisites (p. 100) .
Subnet group	Select the DB subnet group to use for the DB cluster. Select Create new DB subnet group to have Amazon RDS create a DB subnet group for you. For more information, see DB Cluster Prerequisites (p. 100) .

Option	Description
Public accessibility	Select Yes to give the DB cluster a public IP address; otherwise, select No . The instances in your DB cluster can be a mix of both public and private DB instances. For more information about hiding instances from public access, see Hiding a DB Instance in a VPC from the Internet (p. 251) .
Availability zone	Determine if you want to specify a particular Availability Zone. For more information about Availability Zones, see Choosing the Regions and Availability Zones (p. 80) .
VPC security groups	Select Create new VPC security group to have Amazon RDS create a VPC security group for you. Select Select existing VPC security groups to specify one or more VPC security groups to secure network access to the DB cluster. For more information, see DB Cluster Prerequisites (p. 100) .
Database port	Specify the port for applications and utilities to use to access the database. Aurora MySQL DB clusters default to the default MySQL port, 3306. Firewalls at some companies block connections to this port. If your company firewall blocks the default port, choose another port for the new DB cluster.
DB parameter group	Select a DB parameter group for the Aurora MySQL DB cluster. Aurora has a default DB parameter group you can use, or you can create your own DB parameter group. For more information about DB parameter groups, see Working with DB Parameter Groups and DB Cluster Parameter Groups (p. 286) .
DB cluster parameter group	Select a DB cluster parameter group for the Aurora MySQL DB cluster. Aurora has a default DB cluster parameter group you can use, or you can create your own DB cluster parameter group. For more information about DB cluster parameter groups, see Working with DB Parameter Groups and DB Cluster Parameter Groups (p. 286) .
Encryption	<p>Choose Disable encryption if you don't want your new Aurora DB cluster to be encrypted. Choose Enable encryption for your new Aurora DB cluster to be encrypted at rest. If you choose Enable encryption, you must choose an AWS KMS encryption key as the Master key value.</p> <p>If your MySQL DB instance isn't encrypted, specify an encryption key to have your DB cluster encrypted at rest.</p> <p>If your MySQL DB instance is encrypted, specify an encryption key to have your DB cluster encrypted at rest using the specified encryption key. You can specify the encryption key used by the MySQL DB instance or a different key. You can't create an unencrypted DB cluster from an encrypted MySQL DB instance.</p>

Option	Description
Priority	Choose a failover priority for the DB cluster. If you don't select a value, the default is tier-1 . This priority determines the order in which Aurora Replicas are promoted when recovering from a primary instance failure. For more information, see Fault Tolerance for an Aurora DB Cluster (p. 380) .
Backup retention period	Select the length of time, from 1 to 35 days, that Aurora retains backup copies of the database. Backup copies can be used for point-in-time restores (PITR) of your database down to the second.
Enhanced Monitoring	Choose Enable enhanced monitoring to enable gathering metrics in real time for the operating system that your DB cluster runs on. For more information, see Enhanced Monitoring (p. 469) .
Monitoring Role	Only available if Enhanced Monitoring is set to Enable enhanced monitoring . Choose the IAM role that you created to permit Amazon RDS to communicate with Amazon CloudWatch Logs for you, or choose Default to have RDS create a role for you named <code>rds-monitoring-role</code> . For more information, see Enhanced Monitoring (p. 469) .
Granularity	Only available if Enhanced Monitoring is set to Enable enhanced monitoring . Set the interval, in seconds, between when metrics are collected for your DB cluster.
Auto minor version upgrade	This setting doesn't apply to Aurora MySQL DB clusters. For more information about engine updates for Aurora MySQL, see Database Engine Updates for Amazon Aurora MySQL (p. 794) .
Maintenance window	Select Select window and specify the weekly time range during which system maintenance can occur. Or, select No preference for Amazon RDS 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, OS X, 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:master-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:master-mysql-  
instance
```

If you use the console to create an Aurora Read Replica, then Amazon RDS automatically creates the primary instance for your DB cluster Aurora Read Replica. If you use the 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, OS X, 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 Key Management Service (AWS KMS) encryption 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 **mysqlmasterARN**, 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=mysqlmasterARN
&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 Amazon RDS 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 master 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 master MySQL DB instance information is in the **Replication source** field.

aurora-mysql-db-cluster

Details

ARN
arn:aws:rds:█████████████████████:aurora-mysql-db-cluster

DB cluster
aurora-mysql-db-cluster (available)

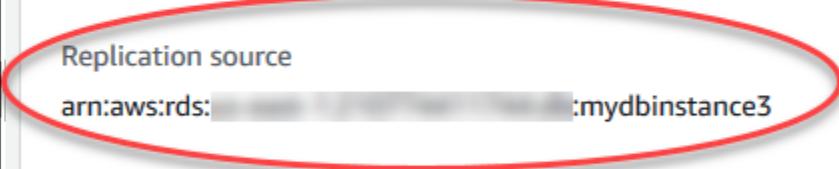
DB cluster role
Replica

Replication source
arn:aws:rds:█████████████████████:mydbinstance3

Cluster endpoint
aurora-mysql-db-cluster.█████████████████████ rds.amazonaws.com

Reader endpoint
aurora-mysql-db-cluster.█████████████████████ rds.amazonaws.com

Port
3306



AWS CLI

To view the MySQL to Aurora MySQL replication relationships for your Aurora MySQL DB clusters by using the AWS CLI, use the `describe-db-clusters` and `describe-db-instances` commands.

To determine which MySQL DB instance is the master, 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 master.

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, OS X, or Unix:

```
aws rds describe-db-clusters \
--db-cluster-identifier myreadreplicacluster
```

```
aws rds describe-db-instances \
--db-instance-identifier mysqlmaster
```

For Windows:

```
aws rds describe-db-clusters ^
--db-cluster-identifier myreadreplicacluster
```

```
aws rds describe-db-instances ^
--db-instance-identifier mysqlmaster
```

Promoting an Aurora Read Replica

After migration completes, you can promote the Aurora Read Replica to a stand-alone DB cluster and direct your client applications to the endpoint for the Aurora Read Replica. For more information on the Aurora endpoints, see [Amazon Aurora Connection Management \(p. 4\)](#). 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 master 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` command on your Aurora Read Replica and reading the **Seconds behind master** value.

You can start writing to the Aurora Read Replica after write transactions to the master 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 master, you risk breaking replication to Aurora. If this happens, you must delete and recreate your Aurora Read Replica.

After you promote, confirm that the promotion has completed by choosing **Instances** in the navigation pane and confirming that there is a **Promoted Read Replica cluster to stand-alone database cluster** event for the Aurora Read Replica. After promotion is complete, the master MySQL DB Instance and the Aurora Read Replica are unlinked, and you can safely delete the DB instance if you want to.

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**.

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, OS X, 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 \(p. 623\)](#)
- [Backtracking an Aurora DB Cluster \(p. 625\)](#)
- [Testing Amazon Aurora Using Fault Injection Queries \(p. 639\)](#)
- [Altering Tables in Amazon Aurora Using Fast DDL \(p. 642\)](#)
- [Displaying Volume Status for an Aurora DB Cluster \(p. 643\)](#)

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 \(p. 284\)](#).

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. The following table describes the specifications of the DB instance classes supported by Aurora MySQL.

Instance Class	vCPU	ECU	Memory (GiB)
db.t2.small	1	1	2
db.t2.medium	2	2	4
db.r3.large	2	6.5	15.25
db.r3.xlarge	4	13	30.5
db.r3.2xlarge	8	26	61
db.r3.4xlarge	16	52	122
db.r3.8xlarge	32	104	244

Instance Class	vCPU	ECU	Memory (GiB)
db.r4.large	2	7	15.25
db.r4.xlarge	4	13.5	30.5
db.r4.2xlarge	8	27	61
db.r4.4xlarge	16	53	122
db.r4.8xlarge	32	99	244
db.r4.16xlarge	64	195	488
db.r5.large	2	10	16
db.r5.xlarge	4	19	32
db.r5.2xlarge	8	38	64
db.r5.4xlarge	16	71	128
db.r5.8xlarge	32	132	256
db.r5.12xlarge	48	173	384
db.r5.16xlarge	64	264	512
db.r5.24xlarge	96	347	768

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, up to 16,000.

Instance Class	max_connections Default Value
db.t2.small	45
db.t2.medium	90
db.r3.large	1000
db.r3.xlarge	2000
db.r3.2xlarge	3000
db.r3.4xlarge	4000
db.r3.8xlarge	5000
db.r4.large	1000

Instance Class	max_connections Default Value		
db.r4.xlarge	2000		
db.r4.2xlarge	3000		
db.r4.4xlarge	4000		
db.r4.8xlarge	5000		
db.r4.16xlarge	6000		
db.r5.large	1000		
db.r5.xlarge	2000		
db.r5.2xlarge	3000		
db.r5.4xlarge	4000		
db.r5.8xlarge	5000		
db.r5.12xlarge	6000		
db.r5.16xlarge	6000		
db.r5.24xlarge	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 instances. The much lower connectivity limits for T2 instances are because T2 instances 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.

Backtracking an Aurora DB Cluster

With Amazon Aurora with MySQL compatibility, 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 \(p. 380\)](#).

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.

Backtrack 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 \(p. 638\)](#).

Backtrack Limitations

The following limitations apply to backtracking:

- Backtracking is only available on 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, backtracking is not possible 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 is not 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 Databases in an Aurora DB Cluster \(p. 335\)](#).
- 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 is not supported for the China (Ningxia) region.
- You can't use Backtrack with Aurora multi-master 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 enable backtracking.

Topics

- [Configuring Backtracking with the Console When Creating a DB Cluster \(p. 627\)](#)
- [Configuring Backtrack with the Console When Modifying a DB Cluster \(p. 628\)](#)

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 \(p. 100\)](#). The following image shows the **Backtrack** section.

Backtrack

Backtrack lets you quickly move an Aurora database to a prior point in time without needing to restore data from a backup. [Info](#)

Enable Backtrack

Target Backtrack window [Info](#)
The Backtrack window determines how far back in time you could go. Aurora will try to retain enough log information to support that window of time.

12 hours (up to 72)

Typical user cost [Info](#)
The cost of Backtrack depends on how often you are updating your database. This is an estimate based on typical workloads for your selected instance size (db.r4.large).

\$ 5.26 USD / month

Disable Backtrack

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.

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. If backtracking is disabled in the **Backtrack** section, choose **Enable Backtrack**.

Note

Currently, you can enable backtracking only for a DB cluster that was created with the Backtrack feature enabled. The **Backtrack** section doesn't appear for a DB cluster that was created with the Backtrack feature disabled.

5. For **Target Backtrack window**, modify the amount of time that you want to be able to backtrack. The limit is 72 hours.

The screenshot shows the 'Backtrack' configuration page in the Amazon Aurora console. It includes sections for 'Enable Backtrack' (selected), 'Target Backtrack window' (set to 24 hours), 'Estimated Cost' (\$28.76 USD / month), and 'Disable Backtrack'.

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.
6. Choose **Continue**.
 7. 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.
 8. 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 \(p. 100\)](#).

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, OS X, 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 \(p. 100\)](#).

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

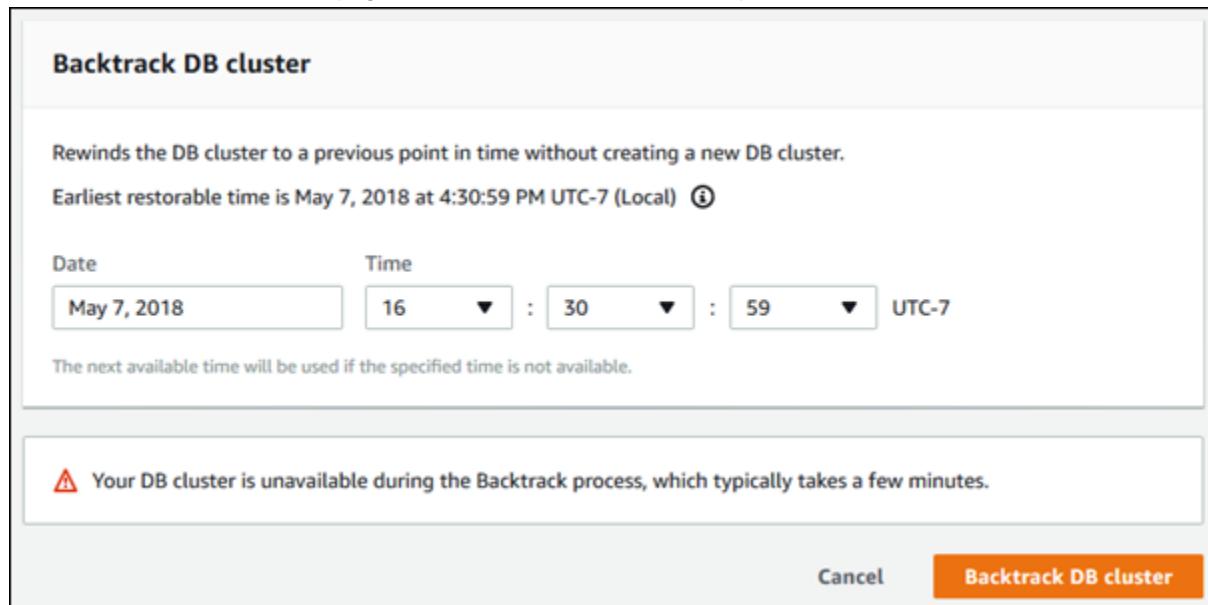
For database clones, you can't backtrack the DB cluster earlier than the date and time when the clone was created. For more information about database cloning, see [Cloning Databases in an Aurora DB Cluster \(p. 335\)](#).

[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, OS X, 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` action. This action 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.

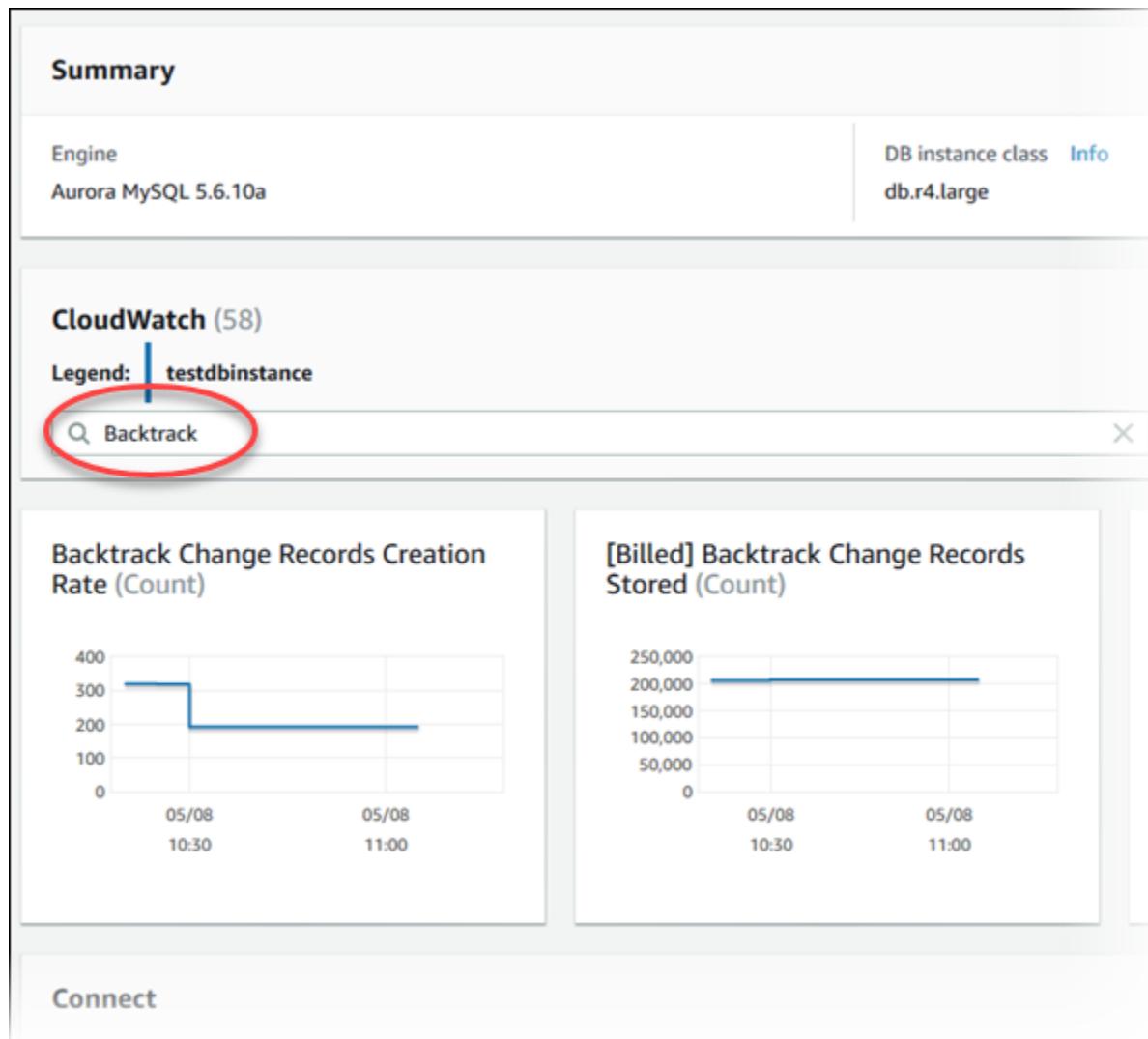
The backtrack information is in the **Backtrack** section.

Backtrack
Backtrack window
Enabled
Target window (12 hours)
Actual window (12 hours)
Earliest backtrack time
May 7, 2018 at 8:19:20 PM UTC-7 (Local) 

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, OS X, 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 action 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

637

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, OS X, 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 action 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.

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).

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` – 0.

The following example disables the Backtrack feature for the `sample-cluster` by setting `--backtrack-window` to 0.

For Linux, OS X, 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.

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

Fault injection queries that specify a crash force a crash of the Aurora 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 \(p. 4\)](#).

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 master instance for the Aurora DB cluster is simulated. The *dispatcher* writes updates to the cluster volume for an Amazon Aurora DB cluster.
- **NODE**—A crash of both the MySQL-compatible database and the dispatcher for the Amazon Aurora instance is simulated. For this fault injection simulation, the cache is also deleted.

The default crash type is `INSTANCE`.

Testing an Aurora Replica Failure

You can simulate the failure of an Aurora Replica using the `ALTER SYSTEM SIMULATE READ REPLICA FAILURE` fault injection query.

An Aurora Replica failure will block all requests to an Aurora Replica or all Aurora Replicas in the DB cluster for a specified time interval. When the time interval completes, the affected Aurora Replicas will be automatically synced up with master instance.

Syntax

```
ALTER SYSTEM SIMULATE percentage_of_failure PERCENT READ REPLICA FAILURE
[ TO ALL | TO "replica name" ]
```

```
FOR INTERVAL quantity { YEAR | QUARTER | MONTH | WEEK | DAY | HOUR | MINUTE | SECOND };
```

Options

This fault injection query takes the following parameters:

- **percentage_of_failure**—The percentage of requests to block during the failure event. This value can be a double between 0 and 100. If you specify 0, then no requests are blocked. If you specify 100, then all requests are blocked.
- **Failure type**—The type of failure to simulate. Specify TO ALL to simulate failures for all Aurora Replicas in the DB cluster. Specify TO and the name of the Aurora Replica to simulate a failure of a single Aurora Replica. The default failure type is TO ALL.
- **quantity**—The amount of time for which to simulate the Aurora Replica failure. The interval is an amount followed by a time unit. The simulation will occur for that amount of the specified unit. For example, 20 MINUTE will result in the simulation running for 20 minutes.

Note

Take care when specifying the time interval for your Aurora Replica failure event. If you specify too long of a time interval, and your master 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 DB Cluster \(p. 643\)](#).
- **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 DB Cluster \(p. 643\)](#).
- **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

In MySQL, many data definition language (DDL) operations have a significant performance impact. Performance impacts occur even with recent online DDL improvements.

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

In Amazon Aurora, you can use fast DDL to execute 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.

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 \(p. 762\)](#).

Note

Fast DDL is available for Aurora version 1.12 and later. For more information about Aurora versions, see [Database Engine Updates for Amazon Aurora MySQL \(p. 794\)](#)

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 does not support partitioned tables.
- Fast DDL does not support InnoDB tables that use the REDUNDANT row format.
- 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.

Syntax

```
ALTER TABLE tbl_name ADD COLUMN col_name column_definition
```

Options

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.

Displaying Volume Status for an Aurora 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.

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 \(p. 639\)](#).

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

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   |
+-----+-----+
```

Working with Parallel Query for Amazon Aurora MySQL

Following, you can find a description of the parallel query performance optimization for Amazon Aurora with MySQL compatibility. This feature uses a special execution path for certain data-intensive queries, taking advantage of the Aurora shared storage architecture. Currently, Aurora MySQL versions that are compatible with MySQL 5.6 support parallel query. 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.

Topics

- [Overview of Parallel Query for Aurora MySQL \(p. 645\)](#)
- [Administering a Parallel Query Cluster \(p. 647\)](#)
- [Upgrade Considerations for Parallel Query \(p. 647\)](#)
- [Creating a DB Cluster that Works with Parallel Query \(p. 647\)](#)
- [Enabling and Disabling Parallel Query \(p. 651\)](#)
- [Performance Tuning for Parallel Query \(p. 653\)](#)
- [Creating Schema Objects to Take Advantage of Parallel Query \(p. 653\)](#)
- [Verifying Which Statements Use Parallel Query \(p. 653\)](#)
- [Monitoring Parallel Query \(p. 656\)](#)
- [How Parallel Query Works with SQL Constructs \(p. 658\)](#)

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.

When the parallel query feature is enabled, the Aurora MySQL engine automatically determines when queries can benefit, without requiring SQL changes such as hints or table attributes. 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 or if another optimization technique makes more sense, such as query caching, buffer pool caching, or index lookups. See [Verifying Which Statements Use Parallel Query \(p. 653\)](#) if you find parallel query isn't being used when you expect it.

Topics

- [Benefits \(p. 645\)](#)
- [Architecture \(p. 645\)](#)
- [Limitations \(p. 646\)](#)

Benefits

With parallel query, you can run data-intensive analytic queries on Aurora MySQL tables, in many cases with 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 execution, 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, which 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 execution 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*) and performing all further processing 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.

Limitations

The following limitations apply to the parallel query feature:

- Currently, Aurora MySQL versions that are compatible with MySQL 5.6 support parallel query. (Note that the PostgreSQL database engine has an unrelated feature that is also called "parallel query".)
- Currently, you can only use parallel query with the following instance classes:
 - All instance classes in the db.r3 series.
 - All instance classes in the db.r4 series.
 - All instance classes that Aurora MySQL supports in the db.r5 series.

Note

You can't create db.t2 or db.t3 instances for parallel query.

- The parallel query option is available in the following regions:
 - US East (N. Virginia)
 - US East (Ohio)
 - US West (Oregon)
 - Europe (Ireland)
 - Asia Pacific (Tokyo)
- Using parallel query requires creating a new cluster, or restoring from an existing Aurora MySQL cluster snapshot, as described following.
- The Performance Insights feature is currently not available for clusters that are enabled for parallel query.
- The backtrack feature is currently not available for clusters that are enabled for parallel query.
- You can't stop and start DB clusters that are enabled for parallel query.
- Currently, partitioned tables aren't supported for parallel query. You can use partitioned tables in parallel query clusters. Queries against those tables use the non-parallel query execution path.

Note

A join, union, or other multipart query can partially use parallel query, even if some query blocks refer to partitioned tables. The query blocks that refer only to nonpartitioned tables can use the parallel query optimization.

- To work with parallel query, currently a table must use the COMPACT row format, which requires the Antelope file format of the InnoDB storage engine.
- TEXT, BLOB, 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.
- 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.

The `JSON` data type is a `BLOB`-like type that is only available in Aurora and is compatible with MySQL 5.7. Parallel query is only available in the Aurora compatible with MySQL 5.6. Thus, this type currently can't be present in a table in a cluster that includes the parallel query feature.

- 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.
- 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.
- Currently, correlated subqueries can't use the parallel query optimization.
- Parallel query works with `SELECT` statements that do no writes or locking, and only under the `REPEATABLE READ` isolation level. For example, parallel query doesn't work with `SELECT FOR UPDATE` or with the `WHERE` clauses of `UPDATE` or `DELETE` statements.
- Parallel query is only available for tables for which no fast online data definition language (DDL) operations are pending.
- The parallel query feature works with most, but not all, operators and functions in the `WHERE` clause. For examples that illustrate compatible operations, see [How Parallel Query Works with SQL Constructs \(p. 658\)](#).
- 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 \(p. 656\)](#). You can check the limit on concurrent sessions for a given DB instance by querying the status variable `Aurora_pq_max_concurrent_requests`.
- Currently, parallel query doesn't support AWS Identity and Access Management (IAM) database authentication.

Administering a Parallel Query Cluster

Administering a cluster that is enabled for parallel query requires setup steps (either creating or restoring a full Aurora MySQL cluster) and deciding how broadly to enable parallel query across the cluster.

You might need to create new versions of some large tables where parallel query is useful, for example to make the table nonpartitioned or to remove full-text search indexes. For details, see [Creating Schema Objects to Take Advantage of Parallel Query \(p. 653\)](#).

After setup is complete, ongoing administration involves monitoring performance and removing obstacles to parallel query usage. For those instructions, see [Performance Tuning for Parallel Query \(p. 653\)](#).

Upgrade Considerations for Parallel Query

Currently, you can't do an in-place upgrade of an Aurora MySQL cluster to enable the parallel query feature. Using parallel query requires creating a new cluster, or restoring from an existing Aurora MySQL 5.6 cluster snapshot.

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 5.6-compatible database. 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 \(p. 100\)](#).

However, certain options are different:

- When you choose an Aurora MySQL engine version, make sure to choose the latest engine that is compatible with MySQL 5.6. Currently, Aurora MySQL versions that are compatible with MySQL 5.6 support parallel query.
- When you create or restore the DB cluster, make sure to choose the **parallelquery** engine mode.

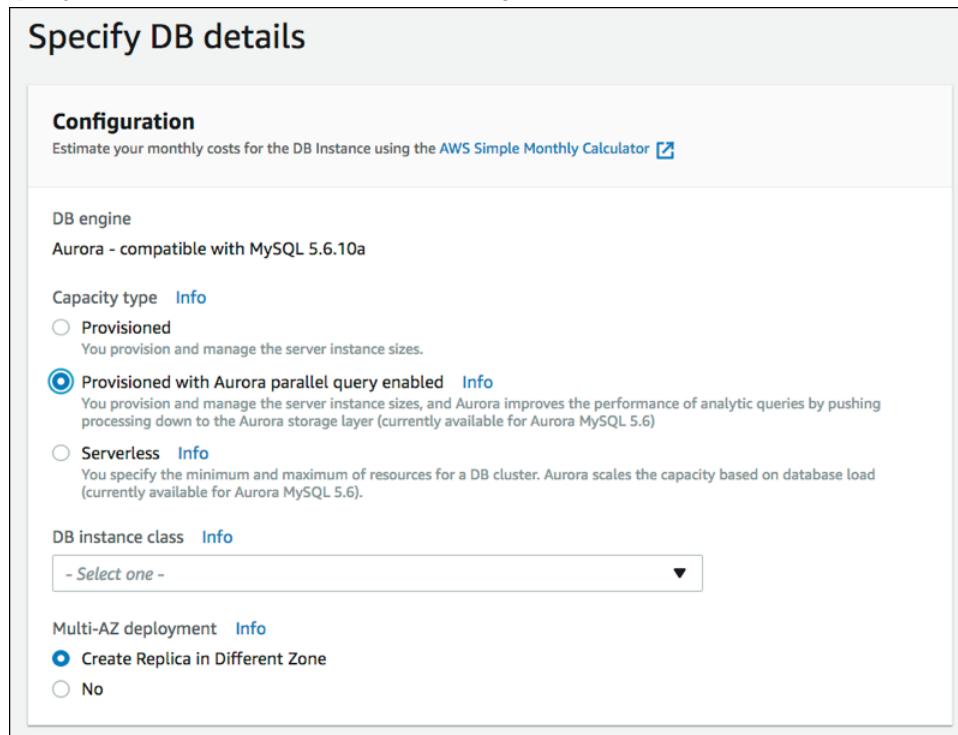
Whether you create a new cluster or restore from a snapshot, you use the same techniques to add new DB instances that you do with other Aurora MySQL clusters.

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

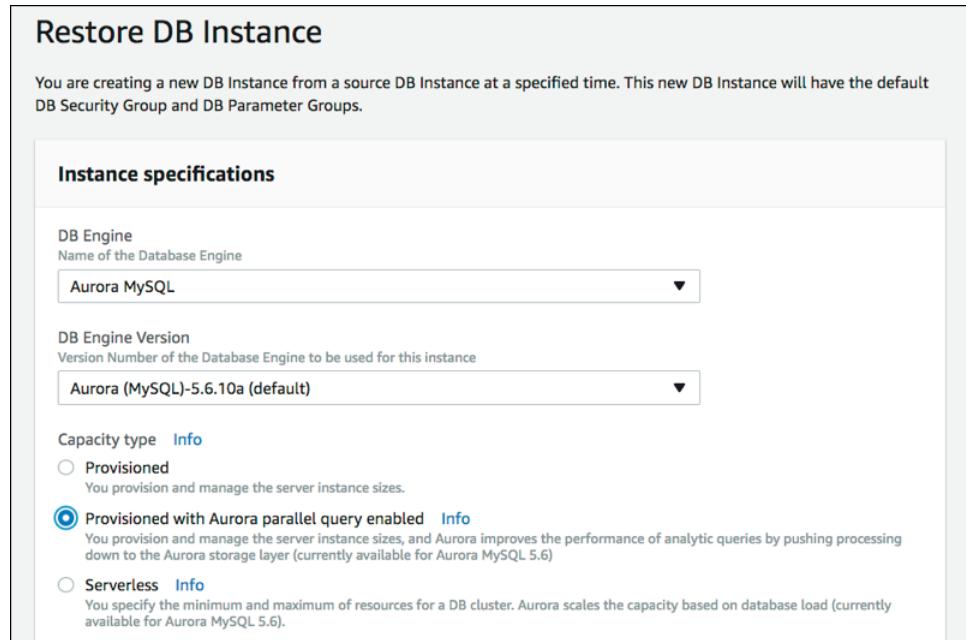
1. Follow the general AWS Management Console procedure in [Creating an Amazon Aurora DB Cluster \(p. 100\)](#).
2. On the **Select engine** screen, choose the MySQL 5.6-compatible edition of Aurora.
3. On the **Specify DB details** screen, for **Capacity type**, choose **Provisioned with Aurora parallel query enabled** as in the screenshot following.



To restore a snapshot to a parallel query cluster with the AWS Management Console

1. Locate a snapshot of a MySQL 5.6-compatible database instance.

2. Follow the general AWS Management Console procedure in [Restoring from a DB Cluster Snapshot \(p. 385\)](#).
3. For **DB Engine Mode**, choose **parallelquery**, as shown in the screenshot following.



To verify that a new cluster can use parallel query

1. Create or restore a cluster using the preceding techniques.
2. Check that the `aurora_pq_supported` configuration setting is true.

```
mysql> select @@aurora_pq_supported;
+-----+
| @@aurora_pq_supported |
+-----+
|          1           |
+-----+
```

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. Check which combinations of Aurora MySQL version, AWS instance class, AWS Region, Availability Zone, and so on, are available for parallel query clusters. To do so, use the `aws rds describe-orderable-db-instance-options` command, which produces output in JSON format. The following code example shows how to check which combinations are available for parallel query clusters in a specified AWS Region.

```
aws rds describe-orderable-db-instance-options --  
engine aurora --query 'OrderableDBInstanceOptions[?  
contains(SupportedEngineModes, `parallelquery`)==`true`].
```

```
{Engine:Engine,EngineVersion:EngineVersion,SupportedEngineModes:SupportedEngineModes,--region us-east-1}
```

The preceding command produces output similar to the following:

```
[  
  {  
    "DBInstanceClass": "db.r3.2xlarge",  
    "EngineVersion": "5.6.10a",  
    "Engine": "aurora",  
    "SupportedEngineModes": [  
      "parallelquery"  
    ]  
  },  
  {  
    "DBInstanceClass": "db.r3.4xlarge",  
    "EngineVersion": "5.6.10a",  
    ...  
}
```

2. Follow the general AWS CLI procedure in [Creating an Amazon Aurora DB Cluster \(p. 100\)](#).
3. Specify the following set of options:
 - For the --engine option, use aurora.
 - 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 --engine-version option, use 5.6.10a.

The following code example shows how.

```
aws rds create-db-cluster --db-cluster-identifier $CLUSTER_ID  
  --engine aurora --engine-mode parallelquery --engine-version 5.6.10a \  
  --master-username $MASTER_USER_ID --master-user-password $MASTER_USER_PW \  
  --db-subnet-group-name $SUBNET_GROUP --vpc-security-group-ids $SECURITY_GROUP  
  
aws rds create-db-instance --db-instance-identifier ${INSTANCE_ID}-1 \  
  --engine aurora \  
  --db-cluster-identifier $CLUSTER_ID --db-instance-class ${INSTANCE_CLASS}
```

4. Verify that a cluster you created or restored has the parallel query feature available. Check that the `aurora_pq_supported` configuration setting is true.

```
mysql> select @@aurora_pq_supported;  
+-----+  
| @@aurora_pq_supported |  
+-----+  
| 1 |  
+-----+
```

To restore a snapshot to a parallel query cluster with the AWS CLI

1. Check which combinations of Aurora MySQL version, AWS instance class, AWS Region, Availability Zone, and so on, are available for parallel query clusters. To do so, use the `aws rds describe-orderable-db-instance-options` command, which produces output in JSON format. The following code example shows how.

```
# See choices for a specified AWS Region. aws rds describe-orderable-db-
instance-options --engine aurora --query 'OrderableDBInstanceOptions[?
contains(SupportedEngineModes,`parallelquery`)==`true`].
{Engine:Engine,EngineVersion:EngineVersion,SupportedEngineModes:SupportedEngineModes,Db-
--region us-east-1 [ { "DBInstanceClass": "db.r3.2xlarge",
"EngineVersion": "5.6.10a", "Engine": "aurora", "SupportedEngineModes": [
"parallelquery" ] }, { "DBInstanceClass": "db.r3.4xlarge",
"EngineVersion": "5.6.10a", ...
```

2. Locate a snapshot from a MySQL 5.6-compatible database.
3. Follow the general AWS CLI procedure in [Restoring from a DB Cluster Snapshot \(p. 385\)](#).
4. For the --engine-mode option, specify parallelquery. The following code example shows how.

```
aws rds restore-db-instance-from-db-snapshot \
--db-instance-identifier mynewdbinstance \
--db-snapshot-identifier mydbsnapshot \
--engine-mode parallelquery
```

5. Verify that a cluster you created or restored has the parallel query feature available. Check that the aurora_pq_supported configuration setting is true.

```
mysql> select @@aurora_pq_supported;
+-----+
| @@aurora_pq_supported |
+-----+
| 1 |
+-----+
```

Enabling and Disabling Parallel Query

You can enable and disable 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 enabled 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 enable or disable parallel query dynamically.

Currently, when you change the global setting for the `aurora_pq` parameter, you must do so for the whole cluster, not for individual DB instances. To toggle the `aurora_pq` parameter at the cluster level, use the techniques for working with parameter groups, as described in [Working with DB Parameter Groups and DB Cluster Parameter Groups \(p. 286\)](#). Follow these steps:

1. Create a custom cluster parameter group.
2. Update `aurora_pq` to the desired value.
3. Attach the custom cluster parameter group to the Aurora cluster where you intend to use the parallel query feature.
4. Restart all the DB instances of the cluster.

You can modify the parallel query parameter by using the [ModifyDBClusterParameterGroup API operation](#) or the AWS Management Console.

Note

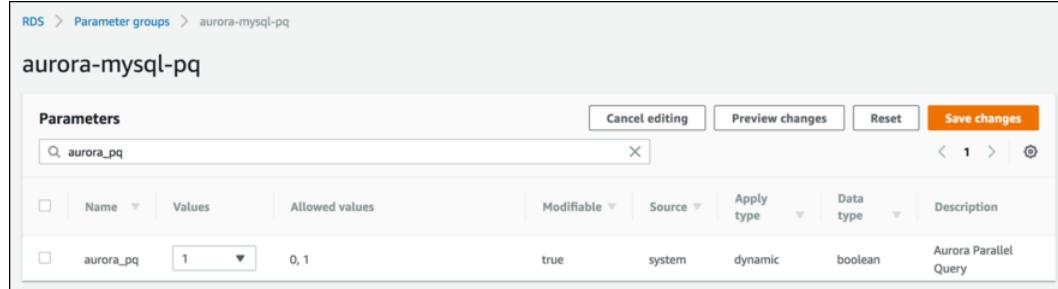
When parallel query is enabled, Aurora MySQL determines whether to use it at runtime for each query. In the case of joins, unions, subqueries, and so on, Aurora MySQL determines whether to use parallel query at runtime for each query block. For details, see [Verifying Which Statements Use Parallel Query \(p. 653\)](#) and [How Parallel Query Works with SQL Constructs \(p. 658\)](#).

Enabling and Disabling Parallel Query for a DB Instance Using the Console

You can enable or disable parallel query at the DB instance level by working with parameter groups.

To enable or disable parallel query for an Aurora MySQL cluster with the AWS Management Console

1. Create a custom parameter group, as described in [Working with DB Parameter Groups and DB Cluster Parameter Groups \(p. 286\)](#).
2. Update `aurora_pq` to **0** (disabled) or **1** (enabled), as shown in the following screenshot. On clusters where the parallel query feature is available, `aurora_pq` is enabled by default.



3. Attach the custom cluster parameter group to the Aurora DB cluster where you intend to use the parallel query feature.

Enabling and Disabling Parallel Query for a DB Instance Using the CLI

You can modify the parallel query parameter by using the `modify-db-cluster-parameter-group` command.

To enable or disable parallel query for a DB instance with the CLI

- Modify the parallel query parameter by using the `modify-db-cluster-parameter-group` command.

You can also enable or disable 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_pq = { 'ON' / 'OFF' }.`

You can also add the session-level parameter to the JDBC configuration or within your application code to enable or disable parallel query dynamically.

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:

- Monitor which queries use parallel query.
- Verify that it is being used for the most data-intensive and long-running queries.
- Fine-tune conditions on your cluster to enable parallel query to apply to the queries you expect, and to run with the right level of concurrency for your workload.

Creating Schema Objects to Take Advantage of Parallel Query

Because parallel query requires tables to use the `ROW_FORMAT=Compact` setting, check your Aurora configuration settings for any changes to the `INNODB_FILE_FORMAT` configuration option. (Alternative `ROW_FORMAT` settings on the `CREATE TABLE` statement require the `INNODB_FILE_FORMAT` configuration option to already be set to `'Barracuda'`.) Issue the `SHOW TABLE STATUS` statement to confirm the row format for all the tables in a database.

Parallel query currently requires tables to be nonpartitioned. Thus, check your `CREATE TABLE` statements and `SHOW CREATE TABLE` output and remove any `PARTITION BY` clauses. For existing partitioned tables, first copy the data into nonpartitioned tables with the same column definitions and indexes. Then rename old and new tables so that the nonpartitioned table is used by existing queries and ETL workflows.

Before changing your schema to enable parallel query to work with more tables, conduct tests to 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`, test the performance of workloads against 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 is not 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 you created recently to perform experiments.

As you monitor or tune cluster performance, you need 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 execution 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 \(p. 658\)](#).

The following example demonstrates the difference between a traditional execution plan and a parallel query plan. This query is Query 3 from the TPC-H benchmark. Many of the sample queries throughout this section use the tables from the TPC-H dataset.

```
SELECT l_orderkey,
       sum(l_extendedprice * (1 - l_discount)) AS revenue,
       o_orderdate,
       o_shipppriority
  FROM customer,
       orders,
       lineitem
 WHERE c_mktsegment = 'AUTOMOBILE'
   AND c_custkey = o_custkey
   AND l_orderkey = o_orderkey
   AND o_orderdate < date '1995-03-13'
   AND l_shipdate > date '1995-03-13'
 GROUP BY l_orderkey,
          o_orderdate,
          o_shipppriority
 ORDER BY revenue DESC,
          o_orderdate LIMIT 10;
```

With parallel query disabled, the query might have an execution plan like the following, which uses hash join but not parallel query.

id	select_type	table	rows	Extra
1	SIMPLE	customer	5798330	Using where; Using index; Using temporary; Using filesort
1	SIMPLE	orders	154545408	Using where; Using join buffer (Hash Join Outer table orders)
1	SIMPLE	lineitem	606119300	Using where; Using join buffer (Hash Join Outer table lineitem)

After parallel query is enabled, two steps in this execution plan can use the parallel query optimization, as shown under the `Extra` column in the EXPLAIN output. The I/O-intensive and CPU-intensive processing for those steps is pushed down to the storage layer.

id	Extra
1	Using where; Using index; Using temporary; Using filesort

```
| 1 | ... | Using where; Using join buffer (Hash Join Outer table orders); Using parallel query (4 columns, 1 filters, 1 exprs; 0 extra) |
| 1 | ... | Using where; Using join buffer (Hash Join Outer table lineitem); Using parallel query (4 columns, 1 filters, 1 exprs; 0 extra) |
+---+...
+-
+
```

For information about how to interpret EXPLAIN output for a parallel query and the parts of SQL statements that parallel query can apply to, see [How Parallel Query Works with SQL Constructs \(p. 658\)](#).

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, and this example query ran using an early version of parallel query, your results might be different. Always conduct your own performance tests to confirm the findings with your own environment, workload, and so on.

```
-- Without parallel query
+-----+-----+-----+
| l_orderkey | revenue      | o_orderdate | o_shippriority |
+-----+-----+-----+
| 92511430  | 514726.4896  | 1995-03-06   |                 0 |
.
.
| 28840519  | 454748.2485  | 1995-03-08   |                 0 |
+-----+-----+-----+
10 rows in set (24 min 49.99 sec)
```

```
-- With parallel query
+-----+-----+-----+
| l_orderkey | revenue      | o_orderdate | o_shippriority |
+-----+-----+-----+
| 92511430  | 514726.4896  | 1995-03-06   |                 0 |
.
.
| 28840519  | 454748.2485  | 1995-03-08   |                 0 |
+-----+-----+-----+
10 rows in set (1 min 49.91 sec)
```

Many of the sample queries throughout this section use the tables from this TPC-H dataset, particularly the PART table, which has 20 million rows and the following definition.

```
+-----+-----+-----+-----+-----+
| Field       | Type        | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+
| p_partkey   | int(11)     | NO   | PRI | NULL    |       |
| p_name       | varchar(55) | NO   |     | NULL    |       |
| p_mfgr       | char(25)    | NO   |     | NULL    |       |
| p_brand      | char(10)    | NO   |     | NULL    |       |
| p_type       | varchar(25) | NO   |     | NULL    |       |
| p_size       | int(11)     | NO   |     | NULL    |       |
| p_container  | char(10)    | NO   |     | NULL    |       |
| p_retailprice| decimal(15,2) | NO   |     | NULL    |       |
| p_comment    | varchar(23) | NO   |     | NULL    |       |
+-----+-----+-----+-----+-----+
```

Experiment with your workload to get a sense of whether individual SQL statements can take advantage of parallel query. Then, use the following monitoring techniques to help verify how often parallel query is used in real workloads over time. For real workloads, extra factors such as concurrency limits apply.

Monitoring Parallel Query

In addition to the Amazon CloudWatch metrics described in [Monitoring Amazon Aurora DB Cluster Metrics \(p. 457\)](#), Aurora provides other global status variables. You can use these global status variables to help monitor parallel query execution and 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 an executed query. For example, suppose your execution 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. You can then adjust your cluster settings, query mix, DB instances where parallel query is enabled, and so on, so that parallel query runs for the queries that you expect.

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. This counter can

	be incremented by an EXPLAIN statement even though the query isn't actually performed.
Aurora_pq_request_not_chosen_below_min_rows	The number of times parallel query wasn't chosen due to the number of rows in the table. This counter can be incremented by an EXPLAIN statement even though the query isn't actually performed.
Aurora_pq_request_not_chosen_small_table	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. This counter can be incremented by an EXPLAIN statement even though the query isn't actually performed.
Aurora_pq_request_not_chosen_high_buffer	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. This counter can be incremented by an EXPLAIN statement even though the query isn't actually performed.
Aurora_pq_request_not_chosen_few_pages_out_of_buffer	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. This counter can be incremented by an EXPLAIN statement even though the query isn't actually performed.
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 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 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 execution path, due to the query being started inside a long-running transaction. This counter can be incremented by an EXPLAIN statement even though the query isn't actually performed.

Aurora_pq_request_not_chosen_unsupported

The number of parallel query requests that use the nonparallel query execution 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.

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 and how Aurora MySQL features interact with parallel query. These details 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.

Topics

- [EXPLAIN statement \(p. 658\)](#)
- [WHERE Clause \(p. 660\)](#)
- [Function Calls in WHERE Clause \(p. 661\)](#)
- [Aggregate Functions, GROUP BY Clauses, and HAVING Clauses \(p. 662\)](#)
- [Comparison Operators \(p. 663\)](#)
- [Joins \(p. 663\)](#)
- [Subqueries \(p. 664\)](#)
- [UNION \(p. 665\)](#)
- [Views \(p. 665\)](#)
- [Data Manipulation Language \(DML\) Statements \(p. 666\)](#)
- [Transactions and Locking \(p. 666\)](#)
- [Indexes \(p. 668\)](#)
- [Built-In Caching Mechanisms \(p. 668\)](#)
- [MyISAM Temporary Tables \(p. 669\)](#)

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 execution 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 (that is, a value not in the millions) suggests that the query isn't accessing enough data to make parallel query worthwhile, and 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, based on the characteristics of the primary key, that the query will scan only a small number of rows. In this case, parallel query isn't required.

```
mysql> explain select count(*) from part where p_partkey between 1 and 100;
+-----+-----+-----+-----+-----+-----+-----+
+-----+-----+
| id | select_type | table | type   | possible_keys | key        | key_len | ref   | rows |
|    |            |       | range  | PRIMARY      | PRIMARY    | 4        | NULL  | 99   |
|    |            |       | Using where; Using index |
+-----+-----+-----+-----+-----+-----+-----+
```

The output showing whether parallel query will be used takes into account all available factors at the moment that the `EXPLAIN` statement is run. The optimizer might make a different choice when the

query is actually run, if the situation changed in the meantime. For example, `EXPLAIN` might report that a statement will use parallel query. But when the query is actually run later, it might not use parallel query based on the conditions then. Such conditions can include several other parallel queries running concurrently, 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 |
| Extra |           |       |       |             |     |         |     |      |
+-----+-----+-----+-----+-----+-----+-----+
+-----+-----+
| 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 |
+-----+...+-----+
```

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) |
+----+...
+-----+
|
```

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. 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) |
+----+...
+-----+
+
```

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 executed 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

Hash joins are always available in parallel query-enabled clusters. Parallel query is typically used for the kinds of resource-intensive queries that benefit from the hash join optimization.

```
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, 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) |
+---+-----+...+-----+
+-----+

```

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 execution 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> 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

Parallel query only applies to statements executed under the `REPEATABLE READ` isolation level. This isolation level is the only one that you can set on Aurora DB instances that are Read Replicas. You can use all the isolation levels on the Aurora primary instance.

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_txn where p_retailprice > 10.0;
+-----+...+-----+
| id | ... | rows      | Extra
|   1 | ... | 2976129 | Using where; Using parallel query (1 columns, 1 filters, 0 exprs; 0 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_txn 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_txn 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_not_chosen_long_trx`.

```
mysql> show global status like '%pq%trx%';
+-----+-----+
| Variable_name           | Value |
+-----+-----+
| Aurora_pq_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 |
+-----+
```

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.

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 execution 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 and whether it is worthwhile to keep the data for that table in the buffer pool.

The query cache avoids re-executing 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 that are enabled with parallel query and not.

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 them.

Note

You can publish Aurora MySQL general, slow, audit, and error log data to a log group in CloudWatch Logs. For more information, see [Publishing Amazon Aurora MySQL Logs to Amazon CloudWatch Logs \(p. 759\)](#).

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, and the `server_audit_events` parameter to specify what events to log.

Use the `server_audit_excl_users` and `server_audit_incl_users` parameters to specify who gets audited. If `server_audit_excl_users` and `server_audit_incl_users` are 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 only those 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 because `server_audit_incl_users` is given higher priority.

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 \(p. 291\)](#) 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 command to modify DB cluster parameters programmatically.

Modifying these parameters doesn't require a DB cluster restart.

server_audit_logging

Enables or disables Advanced Auditing. This parameter defaults to OFF; set it to ON to enable Advanced Auditing.

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.
- 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_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. 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](#).

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`.

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. 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](#).

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.

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.

Logs (28)		
<input type="text"/> Filter name		
Name	Last written	Size
error/mysql-error-running.log	Fri Jan 12 15:00:00 GMT-800 2018	18.8 kB
error/mysql-error-running.log.2018-01-11.22	Thu Jan 11 14:00:00 GMT-800 2018	96.7 kB
error/mysql-error-running.log.2018-01-11.23	Thu Jan 11 14:30:00 GMT-800 2018	19.4 kB
error/mysql-error-running.log.2018-01-12.00	Thu Jan 11 15:30:00 GMT-800 2018	38 kB
error/mysql-error-running.log.2018-01-12.01	Thu Jan 11 16:30:00 GMT-800 2018	38.2 kB

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 \(p. 563\)](#) and [Downloading a Database Log File \(p. 563\)](#).

Audit Log Details

Log files are in UTF-8 format. Logs are written in multiple files, the number of which varies based on instance size. To see the latest events, you might have to review all of the audit log files.

Log entries are not in sequential order. You can use the timestamp value for ordering.

Log files are rotated when they reach 100 MB in aggregate. This limit is not configurable.

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 executed query. For TABLE events, it indicates the table name.

Field	Description
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 \(p. 702\)](#).

Topics

- [Using Aurora Replicas \(p. 672\)](#)
- [Replication Options for Amazon Aurora MySQL \(p. 673\)](#)
- [Performance Considerations for Amazon Aurora MySQL Replication \(p. 673\)](#)
- [High Availability Considerations for Amazon Aurora MySQL Replication \(p. 674\)](#)
- [Monitoring Amazon Aurora MySQL Replication \(p. 674\)](#)
- [Replicating Amazon Aurora MySQL DB Clusters Across AWS Regions \(p. 674\)](#)
- [Replication Between Aurora and MySQL or Between Aurora and Another Aurora DB Cluster \(p. 684\)](#)
- [Using GTID-Based Replication for Aurora MySQL \(p. 698\)](#)

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 \(p. 47\)](#).

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 master 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 executing 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 executed 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 an Aurora Read Replica of an Aurora MySQL DB cluster in a different AWS Region.

For more information, see [Replicating Amazon Aurora MySQL DB Clusters Across AWS Regions \(p. 674\)](#).

- 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 \(p. 684\)](#).

- An Amazon RDS MySQL DB instance as the master and an Aurora MySQL DB cluster, by creating an Aurora Read Replica of an Amazon RDS MySQL DB instance.

Typically, this approach is used for migration to Aurora MySQL, rather than for ongoing replication.

For more information, see [Migrating Data from a MySQL DB Instance to an Amazon Aurora MySQL DB Cluster by Using a DB Snapshot \(p. 607\)](#).

Note

Rebooting the primary instance of an Amazon Aurora DB cluster also automatically reboots the Aurora Replicas for that DB cluster, in order to re-establish an entry point that guarantees read/write consistency across the DB cluster.

Performance Considerations for Amazon Aurora MySQL Replication

Starting in Aurora MySQL 1.17.4, the following features help you to fine-tune the performance of Aurora MySQL replication.

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.

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.

High Availability Considerations for Amazon Aurora MySQL Replication

Having more Aurora Replicas in your cluster helps to ensure high availability. A database instance with a full copy of your data is always available for you to query, even if some database instances become unavailable.

The tradeoff with having multiple Aurora Replicas is that replicas become unavailable for brief periods when the underlying database instances are restarted. These restarts can happen during maintenance operations, or when a replica begins to lag too far behind the master. Restarting a replica interrupts existing connections to the corresponding database instance. Restarting an Aurora cluster causes all the replicas to become unavailable at the same time.

Starting in Aurora MySQL 1.17.4, the following feature helps to ensure high availability even during these intervals when replicas are restarted.

The *zero-downtime restart* (ZDR) feature preserves existing connections when an Aurora MySQL Replica is restarted, for example if the replica falls too far behind the master. Any open transaction is rolled back, and your application must retry it. To enable this feature, turn on the `aurora_enable_zdr` parameter in the cluster parameter group. This parameter is off by default.

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 `ReplicaLag` metric. Because Aurora Replicas read from the same cluster volume as the primary instance, the `ReplicaLag` metric has a different meaning for an Aurora MySQL 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 primary instance.

If you need the most current value for Aurora Replica lag, you can query the `mysql.ro_replica_status` table 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 `ReplicaLag` metric. The values in the `mysql.ro_replica_status` are also provided 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 an Amazon Aurora DB Cluster \(p. 433\)](#).

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. 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 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 Read Replica region.
 - For each data modification made in the source databases, Amazon RDS transfers data from the source region to the Read Replica region.

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 \(p. 675\)](#)
- [Creating an Amazon Aurora MySQL DB Cluster That Is a Cross-Region Read Replica \(p. 676\)](#)
- [Viewing Amazon Aurora MySQL Cross-Region Replicas \(p. 682\)](#)
- [Promoting a Read Replica to Be a DB Cluster \(p. 682\)](#)
- [Troubleshooting Amazon Aurora MySQL Cross Region Replicas \(p. 684\)](#)

Before You Begin

Before you can create an Aurora MySQL DB cluster that is a cross-region Read Replica, you must enable 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 enable 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, see [Amazon Aurora DB Cluster and DB Instance Parameters \(p. 288\)](#) and [Working with DB Parameter Groups and DB Cluster Parameter Groups \(p. 286\)](#).

Creating an Amazon Aurora MySQL DB Cluster That Is a Cross-Region Read Replica

You can create an Aurora DB cluster that is a cross-region Read Replica by using the AWS Management Console, the AWS Command Line Interface (AWS CLI), or the Amazon RDS API. You can create cross-region Read Replicas from both encrypted and unencrypted DB clusters.

When you create a cross-region Read Replica for Aurora MySQL by using the AWS Management Console, Amazon RDS creates a DB cluster in the target AWS Region, and then automatically creates a DB instance that is the primary instance for that DB cluster.

When you create a cross-region Read Replica using the AWS CLI or RDS API, you first create the DB cluster in the target AWS Region and wait for it to become active. Once it is active, you then create a DB instance that is the primary instance for that DB cluster.

Replication begins when the primary instance of the Read Replica DB cluster becomes available.

Use the following procedures to create a cross-region Read Replica from an Aurora MySQL DB cluster. These procedures work for creating Read Replicas from either encrypted or unencrypted DB clusters.

Console

To create an Aurora MySQL DB cluster that is a cross-region read replica with the AWS Management Console

1. Sign in to the AWS Management Console and open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. In the top-right corner of the AWS Management Console, select the AWS Region that hosts your source DB cluster.
3. In the navigation pane, choose **Instances**.
4. Choose the check box for the DB instance that you want to create a cross-region Read Replica for. For **Actions**, choose **Create cross region read replica**.
5. On the **Create cross region read replica** page, choose the option settings for your cross-region Read Replica DB cluster, as described in the following table.

Option	Description
Destination region	Choose the AWS Region to host the new cross-region Read Replica DB cluster.
Destination DB subnet group	Choose the DB subnet group to use for the cross-region Read Replica DB cluster.
Publicly accessible	Choose Yes to give the cross-region Read Replica DB cluster a public IP address; otherwise, select No .
Encryption	Select Enable Encryption to enable encryption at rest for this DB cluster. For more information, see Encrypting Amazon Aurora Resources (p. 175) .
Master key	Only available if Encryption is set to Enable Encryption . Select the master key to use for encrypting this DB cluster. For more information, see Encrypting Amazon Aurora Resources (p. 175) .
DB instance class	Choose a DB instance class that defines the processing and memory requirements for the primary instance in the

Option	Description
	DB cluster. For more information about DB instance class options, see Choosing the DB Instance Class (p. 76) .
Multi-AZ deployment	Choose Yes to create a Read Replica of the new DB cluster in another Availability Zone in the target AWS Region for failover support. For more information about multiple Availability Zones, see Choosing the Regions and Availability Zones (p. 80) .
Read replica source	Choose the source DB cluster to create a cross-region Read Replica for.
DB instance identifier	<p>Type a name for the primary instance in your cross-region Read Replica DB cluster. This identifier is used in the endpoint address for the primary instance of the new DB cluster.</p> <p>The DB instance identifier has the following constraints:</p> <ul style="list-style-type: none"> • It must contain from 1 to 63 alphanumeric characters or hyphens. • Its first character must be a letter. • It cannot end with a hyphen or contain two consecutive hyphens. • It must be unique for all DB instances for each AWS account, for each AWS Region. <p>Because the cross-region Read Replica DB cluster is created from a snapshot of the source DB cluster, the master user name and master password for the Read Replica are the same as the master user name and master password for the source DB cluster.</p>
DB cluster identifier	<p>Type a name for your cross-region Read Replica DB cluster that is unique for your account in the target AWS Region for your replica. This identifier is used in the cluster endpoint address for your DB cluster. For information on the cluster endpoint, see Amazon Aurora Connection Management (p. 4).</p> <p>The DB cluster identifier has the following constraints:</p> <ul style="list-style-type: none"> • It must contain from 1 to 63 alphanumeric characters or hyphens. • Its first character must be a letter. • It cannot end with a hyphen or contain two consecutive hyphens. • It must be unique for all DB clusters for each AWS account, for each AWS Region.

Option	Description
Priority	Choose a failover priority for the primary instance of the new DB cluster. This priority determines the order in which Aurora Replicas are promoted when recovering from a primary instance failure. If you don't select a value, the default is tier-1 . For more information, see Fault Tolerance for an Aurora DB Cluster (p. 380) .
Database port	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.
Enhanced monitoring	Choose Enable enhanced monitoring to enable gathering metrics in real time for the operating system that your DB cluster runs on. For more information, see Enhanced Monitoring (p. 469) .
Monitoring Role	Only available if Enhanced Monitoring is set to Enable enhanced monitoring . Choose the IAM role that you created to permit Amazon RDS to communicate with Amazon CloudWatch Logs for you, or choose Default to have RDS create a role for you named <code>rds-monitoring-role</code> . For more information, see Enhanced Monitoring (p. 469) .
Granularity	Only available if Enhanced Monitoring is set to Enable enhanced monitoring . Set the interval, in seconds, between when metrics are collected for your DB cluster.
Auto minor version upgrade	This setting doesn't apply to Aurora MySQL DB clusters. For more information about engine updates for Aurora MySQL, see Database Engine Updates for Amazon Aurora MySQL (p. 794) .

6. Choose **Create** to create your cross-region Read Replica for Aurora.

AWS CLI

To create an Aurora MySQL DB cluster that is a cross-region read replica with the CLI

1. Call the AWS CLI `create-db-cluster` command in the AWS Region where you want to create the Read Replica DB cluster. Include the `--replication-source-identifier` option and specify the Amazon Resource Name (ARN) of the source DB cluster to create a Read Replica for.

For cross-region replication where the DB cluster identified by `--replication-source-identifier` is encrypted, you must specify the `--kms-key-id` option and the `--storage-encrypted` option. You must also specify either the `--source-region` or `--pre-signed-url` option. Using `--source-region` autogenerates a pre-signed URL that is a valid request for the `CreateDBCluster` API operation that can be executed in the source AWS Region that contains the encrypted DB cluster to be replicated. Using `--pre-signed-url` requires you to construct a pre-signed URL manually instead. The KMS key ID is used to encrypt the Read Replica, and must be a KMS encryption key valid for the destination AWS Region. To learn more about these options, see [create-db-cluster](#).

Note

You can set up cross-region replication from an unencrypted DB cluster to an encrypted Read Replica by specifying `--storage-encrypted` and providing a value for `--kms-key-id`. In this case, you don't need to specify `--source-region` or `--pre-signed-url`.

You don't need to include the `--master-username` and `--master-user-password` 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 command is called in the us-east-1 region.

For Linux, OS X, 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, OS X, or Unix:

```
aws rds create-db-cluster \
--db-cluster-identifier sample-replica-cluster \
--engine aurora \
--replication-source-identifier arn:aws:rds:us-west-2:123456789012:cluster:sample-
master-cluster \
--kms-key-id my-us-east-1-key \
--source-region us-west-2 \
--storage-encrypted
```

For Windows:

```
aws rds create-db-cluster ^
--db-cluster-identifier sample-replica-cluster ^
--engine aurora ^
--replication-source-identifier arn:aws:rds:us-west-2:123456789012:cluster:sample-
master-cluster ^
--kms-key-id my-us-east-1-key ^
--source-region us-west-2 ^
--storage-encrypted
```

2. Check that the DB cluster has become available to use by using the AWS CLI `describe-db-clusters` command, as shown in the following example.

```
aws rds describe-db-clusters --db-cluster-identifier sample-replica-cluster
```

When the **describe-db-clusters** results show a status of `available`, create the primary instance for the DB cluster so that replication can begin. To do so, use the AWS CLI [create-db-instance](#) command as shown in the following example.

For Linux, OS X, or Unix:

```
aws rds create-db-instance \
--db-cluster-identifier sample-replica-cluster \
--db-instance-class db.r3.large \
--db-instance-identifier sample-replica-instance \
--engine aurora
```

For Windows:

```
aws rds create-db-instance ^
--db-cluster-identifier sample-replica-cluster ^
--db-instance-class db.r3.large ^
--db-instance-identifier sample-replica-instance ^
--engine aurora
```

When the DB instance is created and available, replication begins. You can determine if the DB instance is available by calling the AWS CLI [describe-db-instances](#) command.

RDS API

To create an Aurora MySQL DB cluster that is a cross-region read replica with the API

1. Call the RDS API [CreateDBCluster](#) action in the AWS Region where you want to create the Read Replica DB cluster. Include the `ReplicationSourceIdentifier` parameter and specify the Amazon Resource Name (ARN) of the source DB cluster to create a Read Replica for.

For cross-region replication where the DB cluster identified by `ReplicationSourceIdentifier` is encrypted, you must specify the `KmsKeyId` parameter and set the `StorageEncrypted` parameter to `true`. You must also specify the `PreSignedUrl` parameter. The pre-signed URL must be a valid request for the `CreateDBCluster` API operation that can be executed in the source AWS Region that contains the encrypted DB cluster to be replicated. The KMS key ID is used to encrypt the Read Replica, and must be a KMS encryption key valid for the destination AWS Region. To automatically rather than manually generate a presigned URL, use the AWS CLI [create-db-cluster](#) command with the `--source-region` option instead.

Note

You can set up cross-region replication from an unencrypted DB cluster to an encrypted Read Replica by specifying `StorageEncrypted` as `true` and providing a value for `KmsKeyId`. In this case, you don't need to specify `PreSignedUrl`.

You don't need to include the `MasterUsername` and `MasterUserPassword` parameters, because those values are taken from the source DB cluster.

The following code example creates a Read Replica in the `us-east-1` region from an unencrypted DB cluster snapshot in the `us-west-2` region. The action is called in the `us-east-1` region.

```
https://rds.us-east-1.amazonaws.com/
?Action/CreateDBCluster
&ReplicationSourceIdentifier=arn:aws:rds:us-west-2:123456789012:cluster:sample-
master-cluster
&DBClusterIdentifier=sample-replica-cluster
&Engine=aurora
&SignatureMethod=HmacSHA256
&SignatureVersion=4
&Version=2014-10-31
&X-Amz-Algorithm=AWS4-HMAC-SHA256
&X-Amz-Credential=AKIADQKE4SARGYLE/20161117/us-east-1/rds/aws4_request
&X-Amz-Date=20160201T001547Z
&X-Amz-SignedHeaders=content-type;host;user-agent;x-amz-content-sha256;x-amz-date
&X-Amz-Signature=a04c831a0b54b5e4cd236a90dc9f5fab7185eb3b72b5ebe9a70a4e95790c8b7
```

The following code example creates a Read Replica in the us-east-1 region from an encrypted DB cluster snapshot in the us-west-2 region. The action is called in the us-east-1 region.

```
https://rds.us-east-1.amazonaws.com/
?Action/CreateDBCluster
&KmsKeyId=my-us-east-1-key
&StorageEncrypted=true
&PreSignedUrl=https%253A%252F%252Frds.us-west-2.amazonaws.com%252F
    %253FACTION%253DCreateDBCluster
    %2526DestinationRegion%253Dus-east-1
    %2526KmsKeyId%253Dmy-us-east-1-key
    %2526ReplicationSourceIdentifier%253Darn%25253Aaws%25253Ards%25253Aus-
west-2%25253A123456789012%25253Acluster%25253Asample-master-cluster
    %2526SignatureMethod%253DHmacSHA256
    %2526SignatureVersion%253D4
    %2526Version%253D2014-10-31
    %2526X-Amz-Algorithm%253DAWS4-HMAC-SHA256
    %2526X-Amz-Credential%253DAKIADQKE4SARGYLE%252F20161117%252Fus-west-2%252Frds
%252Faws4_request
    %2526X-Amz-Date%253D20161117T215409Z
    %2526X-Amz-Expires%253D3600
    %2526X-Amz-SignedHeaders%253Dcontent-type%253Bhost%253Buser-agent%253Bx-amz-
content-sha256%253Bx-amz-date
    %2526X-Amz-Signature
%253D255a0f17b4e717d3b67fad163c3ec26573b882c03a65523522cf890a67fca613
    &ReplicationSourceIdentifier=arn:aws:rds:us-west-2:123456789012:cluster:sample-
master-cluster
    &DBClusterIdentifier=sample-replica-cluster
    &Engine=aurora
    &SignatureMethod=HmacSHA256
    &SignatureVersion=4
    &Version=2014-10-31
    &X-Amz-Algorithm=AWS4-HMAC-SHA256
    &X-Amz-Credential=AKIADQKE4SARGYLE/20161117/us-east-1/rds/aws4_request
    &X-Amz-Date=20160201T001547Z
    &X-Amz-SignedHeaders=content-type;host;user-agent;x-amz-content-sha256;x-amz-date
    &X-Amz-Signature=a04c831a0b54b5e4cd236a90dc9f5fab7185eb3b72b5ebe9a70a4e95790c8b7
```

2. Check that the DB cluster has become available to use by using the RDS API [DescribeDBClusters](#) action, as shown in the following example.

```
https://rds.us-east-1.amazonaws.com/
?Action=DescribeDBClusters
&DBClusterIdentifier=sample-replica-cluster
&SignatureMethod=HmacSHA256
```

```
&SignatureVersion=4
&Version=2014-10-31
&X-Amz-Algorithm=AWS4-HMAC-SHA256
&X-Amz-Credential=AKIADQKE4SARGYLE/20161117/us-east-1/rds/aws4_request
&X-Amz-Date=20160201T002223Z
&X-Amz-SignedHeaders=content-type;host;user-agent;x-amz-content-sha256;x-amz-date
&X-Amz-Signature=84c2e4f8fba7c577ac5d820711e34c6e45ffcd35be8a6b7c50f329a74f35f426
```

When the **DescribeDBClusters** results show a status of `available`, create the primary instance for the DB cluster so that replication can begin. To do so, use the RDS API [CreateDBInstance](#) action as shown in the following example.

```
https://rds.us-east-1.amazonaws.com/
?Action/CreateDBInstance
&DBClusterIdentifier=sample-replica-cluster
&DBInstanceClass=db.r3.large
&DBInstanceIdentifier=sample-replica-instance
&Engine=aurora
&SignatureMethod=HmacSHA256
&SignatureVersion=4
&Version=2014-10-31
&X-Amz-Algorithm=AWS4-HMAC-SHA256
&X-Amz-Credential=AKIADQKE4SARGYLE/20161117/us-east-1/rds/aws4_request
&X-Amz-Date=20160201T003808Z
&X-Amz-SignedHeaders=content-type;host;user-agent;x-amz-content-sha256;x-amz-date
&X-Amz-Signature=125fe575959f5bbcebd53f2365f907179757a08b5d7a16a378dfa59387f58cdb
```

When the DB instance is created and available, replication begins. You can determine if the DB instance is available by calling the AWS CLI [DescribeDBInstances](#) command.

Viewing Amazon Aurora MySQL Cross-Region Replicas

You can view the cross-region replication relationships for your Amazon Aurora MySQL DB clusters by calling the [describe-db-clusters](#) AWS CLI command or the [DescribeDBClusters](#) RDS API operation. In the response, refer to the `ReadReplicaIdentifiers` field for the DB cluster identifiers of any cross-region Read Replica DB clusters, and refer to the `ReplicationSourceIdentifier` element for the ARN of the source DB cluster that is the replication master.

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 [Replica Lag](#) metric to determine when all updates have been made to the Read Replica.
2. Promote the Read Replica by using the **Promote read replica** 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. In the Amazon RDS console, choose **Instances**.
The **Instance** pane appears.
3. In the **Instances** pane, select 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, OS X, 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 the resolutions for the specified errors.

Source cluster [DB cluster ARN] doesn't have binlogs enabled

To resolve this issue, enable binary logging on the source DB cluster. For more information, see [Before You Begin \(p. 675\)](#).

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 only have one cross-region DB cluster that is a Read Replica for each source DB cluster in any AWS Region. To create a new cross-region DB cluster that is a Read Replica in a particular AWS Region, you must delete the existing one.

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

Because Amazon Aurora MySQL is compatible with MySQL, you can set up replication between a MySQL database and an Amazon Aurora MySQL DB cluster. 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 master or the replica, and you can replicate with an Amazon RDS MySQL DB instance, a MySQL database external to Amazon RDS, or another Aurora MySQL DB cluster.

You can also replicate with an Amazon RDS MySQL DB instance or Aurora MySQL DB cluster in another AWS Region. When you're performing replication across AWS regions, ensure that your DB clusters and DB instances are publicly accessible. Aurora MySQL DB clusters must be part of a public subnet in your VPC.

If you want to configure replication between an Aurora MySQL DB cluster and an Aurora MySQL DB cluster in another region, you can create an Aurora MySQL DB cluster as a Read Replica in a different AWS Region than the source DB cluster. For more information, see [Replicating Amazon Aurora MySQL DB Clusters Across AWS Regions \(p. 674\)](#).

With Aurora MySQL 2.04 and higher, you can replicate between Aurora MySQL and an external source or target that uses global transaction identifiers (GTIDs) for replication. Ensure that the GTID-

related parameters in the Aurora MySQL DB cluster have settings that are compatible with the GTID status of the external database. To learn how to do this, see [Using GTID-Based Replication for Aurora MySQL \(p. 698\)](#).

Warning

When you replicate between Aurora MySQL and MySQL, ensure that you use only InnoDB tables. If you have MyISAM tables that you want to replicate, you can convert them to InnoDB before setting up replication with the following command.

```
alter table <schema>.<table_name> engine=innodb, algorithm=copy;
```

Setting up MySQL replication with Aurora MySQL involves the following steps, which are discussed in detail following in this topic:

1. [Enable Binary Logging on the Replication Master \(p. 685\)](#)
2. [Retain Binary Logs on the Replication Master Until No Longer Needed \(p. 690\)](#)
3. [Create a Snapshot of Your Replication Master \(p. 691\)](#)
4. [Load the Snapshot into Your Replica Target \(p. 693\)](#)
5. [Enable Replication on Your Replica Target \(p. 694\)](#)
6. [Monitor Your Replica \(p. 696\)](#)

Setting Up Replication with MySQL or Another Aurora DB Cluster

To set up Aurora replication with MySQL, take the following steps.

1. Enable Binary Logging on the Replication Master

Find instructions on how to enable binary logging on the replication master for your database engine following.

Database Engine	Instructions
Aurora	<p>To enable binary logging on an Aurora MySQL DB cluster</p> <p>Set the <code>binlog_format</code> parameter to <code>ROW</code>, <code>STATEMENT</code>, or <code>MIXED</code>. <code>MIXED</code> is recommended unless you have a need for a specific binlog format. The <code>binlog_format</code> parameter is a cluster-level parameter that is in the default cluster parameter group. If you are changing the <code>binlog_format</code> parameter from <code>OFF</code> to another value, then you need to reboot your Aurora DB cluster for the change to take effect.</p> <p>For more information, see Amazon Aurora DB Cluster and DB Instance Parameters (p. 288) and Working with DB Parameter Groups and DB Cluster Parameter Groups (p. 286).</p>
RDS MySQL	<p>To enable binary logging on an Amazon RDS DB instance</p> <p>You cannot enable binary logging directly for an Amazon RDS DB instance, but you can enable it by doing one of the following:</p> <ul style="list-style-type: none">• Enable automated backups for the DB instance. You can enable automated backups when you create a DB instance, or you can enable backups by modifying an existing

Database Engine	Instructions
	<p>DB instance. For more information, see Creating a DB Instance Running the MySQL Database Engine in the <i>Amazon Relational Database Service User Guide</i>.</p> <ul style="list-style-type: none">• Create a Read Replica for the DB instance. For more information, see Working with Read Replicas of MariaDB, MySQL, and PostgreSQL DB Instances in the <i>Amazon Relational Database Service User Guide</i>.

Database Engine	Instructions
MySQL (external)	<p>To set up encrypted replication</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>Note If you don't need to use encrypted replication, you can skip these steps.</p> <p>The following are prerequisites for using encrypted replication:</p> <ul style="list-style-type: none"> Secure Sockets Layer (SSL) must be enabled on the external MySQL master database. A client key and client certificate must be prepared for the Aurora MySQL DB cluster. <p>During encrypted replication, the Aurora MySQL DB cluster acts a client to the MySQL database server. The certificates and keys for the Aurora MySQL client are in files in .pem format.</p> <ol style="list-style-type: none"> 1. Ensure that you are prepared for encrypted replication: <ul style="list-style-type: none"> If you don't have SSL enabled on the external MySQL master 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 master, 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 master database. <p>For more information, see Creating SSL Certificates and Keys Using openssl in the MySQL documentation.</p> <p>You need the certificate authority certificate, the client key, and the client certificate.</p> <ol style="list-style-type: none"> 2. Connect to the Aurora MySQL DB cluster as the master user using SSL. <p>For information about connecting to an Aurora MySQL DB cluster with SSL, see Using SSL with Aurora MySQL DB Clusters (p. 582).</p> <ol style="list-style-type: none"> 3. Run the <code>mysql.rds_import_binlog_ssl_material</code> stored procedure to import the SSL information into the Aurora MySQL DB cluster. <p>For the <code>ssl_material_value</code> parameter, insert the information from the .pem format files for the Aurora MySQL DB cluster in the correct JSON payload.</p> <p>The following example imports SSL information into an Aurora MySQL DB cluster. In .pem format files, the body code typically is longer than the body code shown in the example.</p> <pre>call mysql.rds_import_binlog_ssl_material('{"ssl_ca": "-----BEGIN CERTIFICATE-----\nAAAAB3NzaC1yc2EAAAQABAAQClKsfkNkuSevGj3eYhCe53pcjqP3maAhDFcvBS706V\nhz2ItxCih+PnDSUaw+WNQn/mZphTk/a/gU8jEzoOWbkM4xyyb/wB96xbiFveSFJuOp/\nd6RJhJOI0iBXr\n-----END CERTIFICATE-----"}')</pre>

Database Engine	Instructions
	<pre>lsLnBItntckiJ7FbtxJMLvvwJryDUilBMTjYtwB+QhYXUMOzce5Pjz5/i8SeJtjnV3iAoG/ cQk+0Fzz qaeJAAHco+CY/5WrUBkrHmFJr6HcXkvJdWPkyQS3xqC0+FmUzofz221CBt5IMucxXPkX4rWi +z7wB3Rb BQoQzd8v7yeb7OzlPnWOyN0qFU0XA246RA8QFYiCNYwi3f05p6KLxEXAMPLE -----END CERTIFICATE-----\n", "ssl_cert": "-----BEGIN CERTIFICATE----- AAAAB3NzaC1yc2EAAAQABAAQClKsfkNkuSevGj3eYhCe53pcjqP3maAhDFcvBS706V hz2ItxCih+PnDSUaw+WNQn/mZphTk/a/gU8jEzoOWbkM4yxyb/wB96xbiFveSFJuOp/ d6RJhJOI0iBXr lsLnBItntckiJ7FbtxJMLvvwJryDUilBMTjYtwB+QhYXUMOzce5Pjz5/i8SeJtjnV3iAoG/ cQk+0Fzz qaeJAAHco+CY/5WrUBkrHmFJr6HcXkvJdWPkyQS3xqC0+FmUzofz221CBt5IMucxXPkX4rWi +z7wB3Rb BQoQzd8v7yeb7OzlPnWOyN0qFU0XA246RA8QFYiCNYwi3f05p6KLxEXAMPLE -----END CERTIFICATE-----\n", "ssl_key": "-----BEGIN RSA PRIVATE KEY----- AAAAB3NzaC1yc2EAAAQABAAQClKsfkNkuSevGj3eYhCe53pcjqP3maAhDFcvBS706V hz2ItxCih+PnDSUaw+WNQn/mZphTk/a/gU8jEzoOWbkM4yxyb/wB96xbiFveSFJuOp/ d6RJhJOI0iBXr lsLnBItntckiJ7FbtxJMLvvwJryDUilBMTjYtwB+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 with Aurora MySQL DB Clusters \(p. 582\)](#).

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 enable 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 master-replica relationships.

If encrypted replication isn't required, ensure that the external MySQL database is started with 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
```

Database Engine	Instructions
	<pre data-bbox="499 297 670 325">sync_binlog=1</pre> <p>If encrypted replication is required, ensure that the external MySQL database is started with SSL and binlogs enabled.</p> <p>The entries in the /etc/my.cnf file include the .pem file locations for the MySQL database server.</p> <pre data-bbox="499 593 1005 819"> log-bin=mysql-bin server-id=2133421 innodb_flush_log_at_trx_commit=1 sync_binlog=1 # Setup SSL. ssl-ca=/home/sslcerts/ca.pem ssl-cert=/home/sslcerts/server-cert.pem ssl-key=/home/sslcerts/server-key.pem </pre>
	<p>Additionally, the <code>sql_mode</code> option for your MySQL DB instance must be set to 0, or must not be included in your my.cnf file.</p> <p>While connected to the external MySQL database, record the external MySQL database's binary log position.</p> <pre data-bbox="499 1094 833 1121">mysql> SHOW MASTER STATUS;</pre> <p>Your output should be similar to the following:</p> <pre data-bbox="499 1262 1334 1537"> +-----+-----+-----+ File Position Binlog_Do_DB Binlog_Ignore_DB Executed_Gtid_Set +-----+-----+-----+ mysql-bin.000031 107 +-----+-----+-----+ 1 row in set (0.00 sec) </pre>
	<p>For more information, see Setting the Replication Master Configuration in the MySQL documentation.</p> <p>3. Start the mysql service.</p> <pre data-bbox="499 1706 825 1733">sudo service mysqld start</pre>

2. Retain Binary Logs on the Replication Master Until No Longer Needed

When you use MySQL binlog replication, Amazon RDS doesn't manage the replication process. As a result, you need to ensure that the binlog files on your replication master are retained until after the changes have been applied to the replica. This maintenance helps ensure that you can restore your master database in the event of a failure.

Find instructions on how to retain binary logs for your database engine following.

Database Engine	Instructions
Aurora	<p>To retain binary logs on an Aurora MySQL DB cluster</p> <p>You do not have access to the binlog files for an Aurora MySQL DB cluster. As a result, you must choose a time frame to retain the binlog files on your replication master long enough to ensure that the changes have been applied to your replica before the binlog file is deleted by Amazon RDS. You can retain binlog files on an Aurora MySQL DB cluster for up to 90 days.</p> <p>If you are setting up replication with a MySQL database or RDS 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 binlog retention time frame, use the mysql_rds_set_configuration procedure and specify a configuration parameter of 'binlog retention hours' along with the number of hours to retain binlog files on the DB cluster, up to 2160 (90 days). The following example that sets the retention period for binlog files to 6 days:</p> <div style="border: 1px solid black; padding: 5px;"><pre>CALL mysql.rds_set_configuration('binlog retention hours', 144);</pre></div> <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> command on your replica and checking the Seconds behind master field. If the Seconds behind master field is 0, then there is no replica lag. When there is no replica lag, 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 you specify a value for 'binlog retention hours' that is higher than 2160, then 2160 is used.</p>
RDS MySQL	<p>To retain binary logs on an Amazon RDS DB instance</p> <p>You can retain binlog files on an Amazon RDS DB instance by setting the binlog retention hours just as with an Aurora MySQL DB cluster, described in the previous section.</p> <p>You can also retain binlog files on an Amazon RDS DB instance by creating a Read Replica for the DB instance. This Read Replica is temporary and solely for the purpose of retaining binlog files. After the Read Replica has been created, call the mysql_rds_stop_replication procedure on the Read Replica (the <code>mysql.rds_stop_replication</code> procedure is only available for MySQL versions 5.5, 5.6 and later, and 5.7 and later). While replication is stopped, Amazon RDS doesn't delete any of the binlog files on the replication master. After you have set up replication with your permanent replica, you can delete the Read Replica when the replica lag (Seconds behind master field) between your replication master and your permanent replica reaches 0.</p>

Database Engine	Instructions
MySQL (external)	<p>To retain binary logs on an external MySQL database</p> <p>Because binlog files on an external MySQL database are not managed by Amazon RDS, they are retained until you delete them.</p> <p>After replication has been started, you can verify that changes have been applied to your replica by running the <code>SHOW SLAVE STATUS</code> command on your replica and checking the Seconds behind master field. If the Seconds behind master field is 0, then there is no replica lag. When there is no replica lag, you can delete old binlog files.</p>

3. Create a Snapshot of Your Replication Master

You use a snapshot of your replication master to load a baseline copy of your data onto your replica and then start replicating from that point on.

Find instructions on how to create a snapshot of your replication master for your database engine following.

Database Engine	Instructions
Aurora	<p>To create a snapshot of an Aurora MySQL DB cluster</p> <ol style="list-style-type: none"> 1. Create a DB cluster snapshot of your Amazon Aurora DB cluster. For more information, see Creating a DB Cluster Snapshot (p. 383). 2. Create a new Aurora DB cluster by restoring from the DB cluster snapshot that you just created. Be sure to retain the same DB parameter group for your restored DB cluster as your original DB cluster. Doing this ensures that the copy of your DB cluster has binary logging enabled. For more information, see Restoring from a DB Cluster Snapshot (p. 385). 3. In the console, choose Databases and choose the primary instance (writer) for your restored Aurora DB cluster to show its details. Scroll to Recent Events. An event message shows that includes the binlog file name and position. The event message is in the following format. <div style="border: 1px solid black; padding: 5px; margin-top: 10px;"> <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 <code>describe-events</code> command from the AWS CLI. The following shows an example <code>describe-events</code> command with example output.</p> <div style="border: 1px solid black; padding: 5px; margin-top: 10px;"> <pre>PROMPT> aws rds describe-events</pre> </div> <div style="border: 1px solid black; padding: 5px; margin-top: 10px;"> <pre>{ "Events": [{ "EventCategories": [], "SourceType": "db-instance", "SourceArn": "arn:aws:rds:us-west-2:123456789012:db:sample-restored-instance",</pre> </div>

Database Engine	Instructions
	<pre> "Date": "2016-10-28T19:43:46.862Z", "Message": "Binlog position from crash recovery is mysql-bin- changelog.000003 4278", "SourceIdentifier": "sample-restored-instance" }] } </pre>
	<p>You can also get the binlog file name and position by checking the MySQL error log for the last MySQL binlog file position or for the DB_CRASH_RECOVERY_BINLOG_POSITION entry.</p> <p>4. If your replica target is an Aurora DB cluster owned by another AWS account, an external MySQL database, or an RDS MySQL DB instance, then you can't load the data from an Amazon Aurora DB cluster snapshot. Instead, create a dump of your Amazon Aurora DB cluster by connecting to your DB cluster using a MySQL client and issuing the <code>mysqldump</code> command. Be sure to run the <code>mysqldump</code> command against the copy of your Amazon Aurora DB cluster that you created. The following is an example.</p> <pre>PROMPT> mysqldump --databases <database_name> --single-transaction --order-by-primary -r backup.sql -u <local_user> -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 MySQL	<p>To create a snapshot of an Amazon RDS DB instance</p> <ol style="list-style-type: none"> 1. Create a Read Replica of your Amazon RDS DB instance. For more information on creating a Read Replica, see Creating a Read Replica in the <i>Amazon Relational Database Service User Guide</i>. 2. Connect to your Read Replica and stop replication by running the <code>mysql_rds_stop_replication</code> procedure. 3. While the Read Replica is Stopped, Connect to the Read Replica and run the <code>SHOW SLAVE STATUS</code> command. Retrieve the current binary log file name from the <code>Relay_Master_Log_File</code> field and the log file position from the <code>Exec_Master_Log_Pos</code> field. Save these values for when you start replication. 4. While the Read Replica remains Stopped, create a DB snapshot of the Read Replica. For more information on creating a DB snapshot, see Creating a DB Cluster Snapshot (p. 383). 5. Delete the Read Replica.

Database Engine	Instructions
MySQL (external)	<p>To create a snapshot of an external MySQL database</p> <ol style="list-style-type: none"> Before you create a snapshot, you need to ensure that the binlog location for the snapshot is current with the data in your master instance. To do this, you must first stop any write operations to the instance with the following command: <div style="border: 1px solid black; padding: 5px; margin-top: 10px;"> <pre>mysql> FLUSH TABLES WITH READ LOCK;</pre> </div> <ol style="list-style-type: none"> Create a dump of your MySQL database using the <code>mysqldump</code> command as shown following: <div style="border: 1px solid black; padding: 5px; margin-top: 10px;"> <pre>PROMPT> sudo mysqldump --databases <database_name> --master-data=2 --single-transaction \ --order-by-primary -r backup.sql -u <local_user> -p</pre> </div> <ol style="list-style-type: none"> After you have created the snapshot, unlock the tables in your MySQL database with the following command: <div style="border: 1px solid black; padding: 5px; margin-top: 10px;"> <pre>mysql> UNLOCK TABLES;</pre> </div>

4. Load the Snapshot into Your Replica Target

If you plan to load data from a dump of a MySQL database that is external to Amazon RDS, then you might want to create an EC2 instance to copy the dump files to, and then load the data into your DB cluster or DB instance from that EC2 instance. Using this approach, you can compress the dump file(s) before copying them to the EC2 instance in order to reduce the network costs associated with copying data to Amazon RDS. You can also encrypt the dump file or files to secure the data as it is being transferred across the network.

Find instructions on how to load the snapshot of your replication master into your replica target for your database engine following.

Database Engine	Instructions
Aurora	<p>To load a snapshot into an Aurora MySQL DB cluster</p> <ul style="list-style-type: none"> If the snapshot of your replication master is a DB cluster snapshot, then you can restore from the DB cluster snapshot to create a new Aurora MySQL DB cluster as your replica target. For more information, see Restoring from a DB Cluster Snapshot (p. 385). If the snapshot of your replication master is a DB snapshot, then you can migrate the data from your DB snapshot into a new Aurora MySQL DB cluster. For more information, see Migrating Data to an Amazon Aurora DB Cluster (p. 172). If the snapshot of your replication master is the output from the <code>mysqldump</code> command, then follow these steps: <ol style="list-style-type: none"> Copy the output of the <code>mysqldump</code> command from your replication master to a location that can also connect to your Aurora MySQL DB cluster. Connect to your Aurora MySQL DB cluster using the <code>mysql</code> command. The following is an example.

Database Engine	Instructions
	<pre>PROMPT> mysql -h <host_name> -port=3306 -u <db_master_user> -p</pre> <p>3. At the <code>mysql</code> prompt, run the <code>source</code> command and pass it the name of your database dump file to load the data into the Aurora MySQL DB cluster, for example:</p> <pre>mysql> source backup.sql;</pre>
RDS MySQL	<p>To load a snapshot into an Amazon RDS DB instance</p> <ol style="list-style-type: none"> 1. Copy the output of the <code>mysqldump</code> command from your replication master to a location that can also connect to your MySQL DB instance. 2. Connect to your MySQL DB instance using the <code>mysql</code> command. The following is an example. <pre>PROMPT> mysql -h <host_name> -port=3306 -u <db_master_user> -p</pre> <p>3. At the <code>mysql</code> prompt, run the <code>source</code> command and pass it the name of your database dump file to load the data into the MySQL DB instance, for example:</p> <pre>mysql> source backup.sql;</pre>
MySQL (external)	<p>To load a snapshot into an external MySQL database</p> <p>You cannot load a DB snapshot or a DB cluster snapshot into an external MySQL database. Instead, you must use the output from the <code>mysqldump</code> command.</p> <ol style="list-style-type: none"> 1. Copy the output of the <code>mysqldump</code> command from your replication master to a location that can also connect to your MySQL database. 2. Connect to your MySQL database using the <code>mysql</code> command. The following is an example. <pre>PROMPT> mysql -h <host_name> -port=3306 -u <db_master_user> -p</pre> <p>3. At the <code>mysql</code> prompt, run the <code>source</code> command and pass it the name of your database dump file to load the data into your MySQL database. The following is an example.</p> <pre>mysql> source backup.sql;</pre>

5. Enable Replication on Your Replica Target

Before you enable replication, we recommend that you take a manual snapshot of the Aurora MySQL DB cluster or RDS MySQL DB instance replica target. If a problem arises and you need to re-establish replication with the DB cluster or DB instance replica target, you can restore the DB cluster or DB instance from this snapshot instead of having to import the data into your replica target again.

Also, create a user ID that is used solely for replication. The following is an example.

```
mysql> CREATE USER 'repl_user'@'<domain_name>' IDENTIFIED BY '<password>';
```

The user requires the REPLICATION CLIENT and REPLICATION SLAVE privileges. Grant these privileges to the user.

```
GRANT REPLICATION CLIENT, REPLICATION SLAVE ON *.* TO 'repl_user'@'<domain_name>';
```

If you need to use encrypted replication, require SSL connections for the replication user. For example, you can use one of the following statement to require SSL connections on the user account `repl_user`.

```
GRANT USAGE ON *.* TO 'repl_user'@'<domain_name>' REQUIRE SSL;
```

Note

If REQUIRE SSL isn't included, the replication connection might silently fall back to an unencrypted connection.

Find instructions on how to enable replication for your database engine following.

Database Engine	Instructions
Aurora	<p>To enable replication from an Aurora MySQL DB cluster</p> <p>1. If your DB cluster replica target was created from a DB cluster snapshot, then connect to the DB cluster replica target and issue the <code>SHOW MASTER STATUS</code> command. Retrieve the current binary log file name from the <code>File</code> field and the log file position from the <code>Position</code> field.</p> <p>If your DB cluster replica target was created from a DB snapshot, then you need the binlog file and binlog position that are the starting place for replication. You retrieved these values from the <code>SHOW SLAVE STATUS</code> command when you created the snapshot of your replication master.</p> <p>2. Connect to the DB cluster and issue the <code>mysql_rds_set_external_master</code> and <code>mysql_rds_start_replication</code> procedures to start replication with your replication master using the binary log file name and location from the previous step. The following is an example.</p> <pre>CALL mysql.rds_set_external_master ('mydbinstance.123456789012.us-east-1.rds.amazonaws.com', 3306, 'repl_user', '<password>', 'mysql-bin-changelog.000031', 107, 0); CALL mysql.rds_start_replication;</pre>
RDS MySQL	<p>To enable replication from an Amazon RDS DB instance</p> <p>1. If your DB instance replica target was created from a DB snapshot, then you need the binlog file and binlog position that are the starting place for replication. You retrieved these values from the <code>SHOW SLAVE STATUS</code> command when you created the snapshot of your replication master.</p> <p>2. Connect to the DB instance and issue the <code>mysql_rds_set_external_master</code> and <code>mysql_rds_start_replication</code> procedures to start replication with your replication master</p>

Database Engine	Instructions
	<p>using the binary log file name and location from the previous step. The following is an example.</p> <pre>CALL mysql.rds_set_external_master ('mydbcluster.cluster-123456789012.us-east-1.rds.amazonaws.com', 3306, 'repl_user', '<password>', 'mysql-bin-changelog.000031', 107, 0); CALL mysql.rds_start_replication;</pre>
MySQL (external)	<p>To enable replication from an external MySQL database</p> <ol style="list-style-type: none"> 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> command when you created the snapshot of your replication master. If your external MySQL replica target was populated from the output of the <code>mysqldump</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. <pre>-- -- 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> <ol style="list-style-type: none"> 2. Connect to the external MySQL replica target, and issue <code>CHANGE MASTER TO</code> and <code>START SLAVE</code> to start replication with your replication master using the binary log file name and location from the previous step, for example: <pre>CHANGE MASTER TO MASTER_HOST = 'mydbcluster.cluster-123456789012.us-east-1.rds.amazonaws.com', MASTER_PORT = 3306, MASTER_USER = 'repl_user', MASTER_PASSWORD = '<password>', MASTER_LOG_FILE = 'mysql-bin-changelog.000031', MASTER_LOG_POS = 107; START SLAVE;</pre>

6. Monitor Your Replica

When you set up MySQL replication with an Aurora MySQL DB cluster, you must monitor failover events for the Aurora MySQL DB cluster when it is the replica target. If a failover occurs, then the DB cluster that is your replica target might be recreated on a new host with a different network address. For information on how to monitor failover events, see [Using Amazon RDS Event Notification \(p. 543\)](#).

You can also monitor how far the replica target is behind the replication master by connecting to the replica target and running the `SHOW SLAVE STATUS` command. In the command output, the `Seconds Behind Master` field tells you how far the replica target is behind the master.

Stopping Replication Between Aurora and MySQL or Between Aurora and Another Aurora DB Cluster

To stop binlog 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 Binlog Replication on the Replica Target \(p. 697\)](#)

[2. Disable Binary Logging on the Replication Master \(p. 697\)](#)

1. Stop Binlog Replication on the Replica Target

Find instructions on how to stop binlog replication for your database engine following.

Database Engine	Instructions
Aurora	To stop binlog replication on an Aurora MySQL DB cluster replica target Connect to the Aurora DB cluster that is the replica target, and call the mysql_rds_stop_replication procedure. 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.
RDS MySQL	To stop binlog replication on an Amazon RDS DB instance Connect to the RDS DB instance that is the replica target and call the mysql_rds_stop_replication procedure. The <code>mysql.rds_stop_replication</code> procedure is only available for MySQL versions 5.5 and later, 5.6 and later, and 5.7 and later.
MySQL (external)	To stop binlog replication on an external MySQL database Connect to the MySQL database and call the <code>STOP REPLICATION</code> command.

2. Disable Binary Logging on the Replication Master

Find instructions on how to disable binary logging on the replication master for your database engine following.

Database Engine	Instructions
Aurora	To disable binary logging on an Amazon Aurora DB cluster 1. Connect to the Aurora DB cluster that is the replication master, and set the binlog retention time frame to 0. To set the binlog retention time frame, use the mysql_rds_set_configuration procedure and specify a configuration parameter of 'binlog_retention_hours' along with the number of hours to retain binlog files on the DB cluster, in this case 0, as shown in the following example. <pre>CALL mysql.rds_set_configuration('binlog_retention_hours', 0);</pre> 2. Set the <code>binlog_format</code> parameter to OFF on the replication master. The <code>binlog_format</code> parameter is a cluster-level parameter that is in the default cluster parameter group. After you have changed the <code>binlog_format</code> parameter value, reboot your DB cluster for the change to take effect.

Database Engine	Instructions
	For more information, see Amazon Aurora DB Cluster and DB Instance Parameters (p. 288) and Modifying Parameters in a DB Parameter Group (p. 291) .
RDS MySQL	<p>To disable binary logging on an Amazon RDS DB instance</p> <p>You cannot disable binary logging directly for an Amazon RDS DB instance, but you can disable it by doing the following:</p> <ol style="list-style-type: none"> 1. Disable automated backups for the DB instance. You can disable automated backups by modifying an existing DB instance and setting the Backup Retention Period to 0. For more information, see Modifying a DB Instance Running the MySQL Database Engine and Working With Backups in the Amazon Relational Database Service User Guide. 2. Delete all Read Replicas for the DB instance. For more information, see Working with Read Replicas of MariaDB, MySQL, and PostgreSQL DB Instances in the <i>Amazon Relational Database Service User Guide</i>.
MySQL (external)	<p>To disable binary logging on an external MySQL database</p> <p>Connect to the MySQL database and call the <code>STOP REPLICATION</code> command.</p> <ol style="list-style-type: none"> 1. From a command shell, stop the mysql service, <pre style="border: 1px solid black; padding: 5px;">sudo service mysqld stop</pre> <ol style="list-style-type: none"> 2. Edit the my.cnf file (this file is usually under /etc). <pre style="border: 1px solid black; padding: 5px;">sudo vi /etc/my.cnf</pre> <p>Delete the <code>log_bin</code> and <code>server_id</code> options from the [mysqld] section.</p> <p>For more information, see Setting the Replication Master Configuration in the MySQL documentation.</p> <ol style="list-style-type: none"> 3. Start the mysql service. <pre style="border: 1px solid black; padding: 5px;">sudo service mysqld start</pre>

Using GTID-Based Replication for Aurora MySQL

Following, you can learn how to use global transaction identifiers (GTIDs) with binary log (binlog) replication between an Aurora MySQL cluster and an external source.

Note

For Aurora, you can only use this feature with Aurora MySQL clusters that use binlog replication to or from an external MySQL database. The other database might be an Amazon RDS MySQL instance, an on-premises MySQL database, or an Aurora DB cluster in a different AWS Region. To learn how to configure that kind of replication, see [Replication Between Aurora and MySQL or Between Aurora and Another Aurora DB Cluster \(p. 684\)](#).

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.

Note

GTID-based replication is supported for MySQL 5.7-compatible clusters in Aurora MySQL version 2.04 and later. GTID-based replication isn't supported for MySQL 5.6-compatible clusters in Aurora MySQL version 1.

Topics

- [Overview of Global Transaction Identifiers \(GTIDs\) \(p. 699\)](#)
- [Parameters for GTID-Based Replication \(p. 699\)](#)
- [Configuring GTID-Based Replication for an Aurora MySQL Cluster \(p. 700\)](#)
- [Disabling GTID-Based Replication for an Aurora MySQL DB Cluster \(p. 701\)](#)

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 masters and replicas are consistent.

You typically use GTID-based replication with Aurora when replicating from an external MySQL-compatible database into an Aurora cluster. You can set up this replication configuration as part of a migration from an on-premises or Amazon RDS database into Aurora MySQL. If the external database already uses GTIDs, enabling GTID-based replication for the Aurora cluster simplifies the replication process.

You configure GTID-based replication for an Aurora MySQL cluster by first setting the relevant configuration parameters in a DB cluster parameter group. You then associate that parameter group with the cluster.

Parameters for GTID-Based Replication

Use the following parameters to configure GTID-based replication.

Parameter	Valid Values	Description
gtid_mode	OFF, OFF_PERMISSIVE, ON_PERMISSIVE, ON	OFF specifies that new transactions are anonymous transactions (that is, don't have GTIDs), and a transaction must be anonymous to be replicated. OFF_PERMISSIVE specifies that new transactions are anonymous transactions, but all transactions can be replicated.

Parameter	Valid Values	Description
		ON_PERMISSIVE specifies that new transactions are GTID transactions, but all transactions can be replicated. ON specifies that new transactions are GTID transactions, and a transaction must be a GTID transaction to be replicated.
enforce_gtid_mode	OFF, ON, WARN	OFF allows transactions to violate GTID consistency. ON prevents transactions from violating GTID consistency. WARN allows transactions to violate GTID consistency but generates a warning when a violation occurs.

For GTID-based replication, use these settings for the DB cluster parameter group for your Aurora MySQL DB cluster:

- ON and ON_PERMISSIVE apply only to outgoing replication from an RDS DB instance or Aurora MySQL cluster. Both of these values cause your RDS DB instance or Aurora DB cluster to use GTIDs for transactions that are replicated to an external database. ON requires that the external database also use GTID-based replication. ON_PERMISSIVE makes GTID-based replication optional on the external database.
- OFF_PERMISSIVE, if set, means that your RDS DB instances or Aurora DB cluster can accept incoming replication from an external database. It can do this whether the external database uses GTID-based replication or not.
- OFF, if set, means that your RDS DB instances or Aurora DB cluster only accept incoming replication from external databases that don't use GTID-based replication.

Tip

Incoming replication is the most common binlog replication scenario for Aurora MySQL clusters. For incoming replication, we recommend that you set the GTID mode to OFF_PERMISSIVE. That setting allows incoming replication from external databases regardless of the GTID settings at the replication source.

For more information about parameter groups, see [Working with DB Parameter Groups and DB Cluster Parameter Groups \(p. 286\)](#).

Note

In the AWS Management Console, the gtid_mode parameter appears as gtid-mode.

Configuring GTID-Based Replication for an Aurora MySQL Cluster

When GTID-based replication is enabled for an Aurora MySQL DB cluster, the GTID settings apply to both inbound and outbound binlog replication.

To enable GTID-based replication for an Aurora MySQL cluster

1. Create or edit a DB cluster parameter group using the following parameter settings:

- `gtid_mode` – ON or ON_PERMISSIVE
 - `enforce_gtid_consistency` – ON
2. Associate the DB cluster parameter group with the Aurora MySQL cluster. To do so, follow the procedures in [Working with DB Parameter Groups and DB Cluster Parameter Groups \(p. 286\)](#).

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 \(p. 791\)](#).

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);
```

2. Reset the `gtid_mode` to ON_PERMISSIVE.

- a. Make sure that the DB cluster parameter group associated with the Aurora MySQL cluster has `gtid_mode` set to ON_PERMISSIVE.

For more information about setting configuration parameters using parameter groups, see [Working with DB Parameter Groups and DB Cluster Parameter Groups \(p. 286\)](#).

- 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 master in the previous step to run the following query.

```
SELECT MASTER_POS_WAIT(file, position);
```

For example, if the file name is `mysql-bin-changelog.000031` and the position is `107`, run the following statement.

```
SELECT MASTER_POS_WAIT(mysql-bin-changelog.000031, 107);
```

If the Read Replica is past the specified position, the query returns immediately. Otherwise, the function waits. When the query returns for all Read Replicas, go to the next step.

5. Reset the GTID parameters to disable GTID-based replication:

- a. Make sure that the DB cluster parameter group associated with the Aurora MySQL cluster has the following parameter settings:
 - `gtid_mode` – OFF
 - `enforce_gtid_consistency` – OFF
- b. Restart the Aurora MySQL DB cluster.

Working with Aurora Multi-Master Clusters

Following, you can learn about Aurora multi-master clusters. In a multi-master cluster, all DB instances have read-write capability. Multi-master clusters have different availability characteristics, support for database features, and procedures for monitoring and troubleshooting than single-master clusters.

Topics

- [Overview of Aurora Multi-Master Clusters \(p. 702\)](#)
- [Creating an Aurora Multi-Master Cluster \(p. 707\)](#)
- [Managing Aurora Multi-Master Clusters \(p. 712\)](#)
- [Application Considerations for Aurora Multi-Master Clusters \(p. 715\)](#)
- [Performance Considerations for Aurora Multi-Master Clusters \(p. 724\)](#)
- [Approaches to Aurora Multi-Master Clusters \(p. 726\)](#)

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.

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 \(p. 703\)](#)
- [Multi-Master Cluster Architecture \(p. 705\)](#)
- [Recommended Workloads for Multi-Master Clusters \(p. 705\)](#)
- [Advantages of Multi-Master Clusters \(p. 706\)](#)
- [Limitations of Multi-Master Clusters \(p. 706\)](#)

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

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 reshaping 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 \(p. 717\)](#).

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 2.

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 \(p. 724\)](#).

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 \(p. 714\)](#).

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

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 \(p. 726\)](#).

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 \(p. 726\)](#).
- Aurora detects write conflicts immediately, not when the transaction commits. For details about the write conflict mechanism, see [Conflict Resolution for Multi-Master Clusters \(p. 725\)](#).

Limitations of Multi-Master Clusters

Note

Aurora multi-master clusters are highly specialized for continuous availability use cases. Thus, such clusters might not be generally applicable to all workloads. Your requirements for performance, scalability, and availability might be satisfied by using a larger DB instance class with an Aurora single-master cluster. If so, consider using a provisioned or Aurora Serverless cluster.

AWS and Aurora Limitations

The following limitations currently apply to the AWS and Aurora features that you can use with multi-master clusters:

- Currently, you can have a maximum of two 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.
- 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 \(p. 716\)](#).

- Multi-master clusters have some processing and network overhead for coordination between DB instances. This overhead has the following consequences for write-intensive and read-intensive applications:
 - Throughput benefits are most obvious on busy clusters with multiple concurrent write operations. In many cases, a traditional Aurora cluster with a single primary instance can handle the write traffic for a cluster. In these cases, the benefits of multi-master clusters are mostly for high availability rather than performance.
 - Single-query performance is generally lower than for an equivalent single-master cluster.
- You can't take a snapshot created on a single-master cluster and restore it on a multi-master cluster, or the opposite. Instead, to transfer all data from one kind of cluster to the other, use a logical dump produced by a tool such as AWS Database Migration Service (AWS DMS) or the `mysqldump` command.
- You can't use the parallel query, Aurora Serverless, or Global Database features on a multi-master cluster.

The multi-master aspect is a permanent choice for a cluster. You can't switch an existing Aurora cluster between a multi-master cluster and another kind such as Aurora Serverless or parallel query.

- The zero-downtime patching (ZDP) and zero-downtime restart (ZDR) features aren't available for multi-master clusters.
- Integration with other AWS services such as AWS Lambda, Amazon S3, and AWS Identity and Access Management isn't available for multi-master clusters.
- The Performance Insights feature isn't available for multi-master clusters.
- You can't clone a multi-master cluster.
- You can't enable the backtrack feature for multi-master clusters.

Database Engine Limitations

The following limitations apply to the database engine features that you can use with a multi-master cluster:

- You can't perform binary log (binlog) replication to or from a multi-master cluster. This limitation means you also can't use global transaction ID (GTID) replication in a multi-master cluster.
- The event scheduler isn't available for multi-master clusters.
- The hash join optimization isn't enabled on multi-master clusters.
- The query cache isn't available on multi-master clusters.
- You can't use certain SQL language features on multi-master clusters. For the full list of SQL differences, and instructions about adapting your SQL code to address these limitations, see [SQL Considerations for Multi-Master Clusters \(p. 715\)](#).

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 \(p. 100\)](#).

Console

To create an Aurora multi-master cluster from the AWS Management Console, you make the following choices. On the first screen, you select an Aurora cluster:

Create database

Database settings



Quick create [Info](#)

Provides the fastest way to get started with your database. You can modify

Engine options

Amazon Aurora

Amazon
Aurora

MySQL



PostgreSQL



Oracle

ORACLE®

You also choose MySQL 5.6 compatibility and location **Regional**:

Edition

- Amazon Aurora with MySQL compatibility
- Amazon Aurora with PostgreSQL compatibility

DB engine version [Info](#)

Aurora MySQL 1.21.0 (Compatible with MySQL 5.6)

Select engine version Aurora MySQL 1.21.0 (Compatible with MySQL 5.6) to use create parallel query, multiple writers, serverless, or global databases.

Database location

Regional [Info](#)

You provision your Aurora database in a single region.

Global [Info](#)

You can provision your Aurora database in multiple regions. Writes in the primary typical latency of <1 sec to secondary regions.

On the second screen, choose **Multiple writers** under **Database features**.

Database features

One writer and multiple readers

The readers connect to the same storage volume as the writer instance and supports only read operations. [Info](#)

Multiple writers

Supports read and write operations, and performs all of the data modifications to the cluster volume. [Info](#)

One writer

You provi...
Aurora im...
pushing p...
[Info](#)

Serverless

You speci...
resources
on databa...

Fill in the other settings for the cluster. This part of the procedure is the same as the general procedure for creating an Aurora cluster in [Creating a DB Cluster \(p. 101\)](#).

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 \(p. 280\)](#). Use the same AWS instance class for all DB instances within the multi-master cluster.

After you create the multi-master cluster and associated DB instances, you see the cluster in the AWS Management Console **Databases** page as follows. All DB instances show the role **Writer**.

The screenshot shows the 'Databases' section of the Amazon RDS console. At the top, there is a search bar labeled 'Filter databases'. Below it, a table lists database instances. The columns are 'DB Name', 'Role', and 'CPU'. There are two rows under 'db-multi-master': 'instance-name1' (Writer, 6.72 CPU) and 'instance-name2' (Writer, 6.72 CPU). The 'DB Name' column has a dropdown arrow icon.

DB Name	Role	CPU
db-multi-master	Cluster	-
instance-name1	Writer	6.72
instance-name2	Writer	6.72

AWS CLI

To create a multi-master cluster with the AWS CLI, run the [create-db-cluster](#) AWS CLI command and include the option `--engine_mode=multimaster`.

The following command shows the syntax for creating an Aurora cluster with multi-master replication. For the general procedure to create an Aurora cluster, see [Creating a DB Cluster \(p. 101\)](#).

For Linux, OS X, 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 \(p. 280\)](#). Use the same AWS instance class for all DB instances within the multi-master cluster.

RDS API

To create a multi-master cluster with the RDS API, run the [CreateDBCluster](#) operation. Specify the value `multimaster` for the `EngineMode` parameter. For the general procedure to create an Aurora cluster, see [Creating a DB Cluster \(p. 101\)](#).

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 \(p. 280\)](#). Use the same AWS instance class for all DB instances within the multi-master cluster.

Adding a DB Instance to a Multi-Master Cluster

You need more than one DB instance to see the benefits of a multi-master cluster. You can create another DB instance afterward using the procedures from [Adding Aurora Replicas to a DB Cluster \(p. 280\)](#). The difference for multi-master clusters is that the new DB instance has read-write capability instead of a read-only Aurora Replica. Use the same AWS instance class for all DB instances within the multi-master cluster.

Managing Aurora Multi-Master Clusters

You do most management and administration for Aurora multi-master clusters the same way as for other kinds of Aurora clusters. The following sections explain the differences and unique features of multi-master clusters for administration and management.

Topics

- [Monitoring an Aurora Multi-Master Cluster \(p. 712\)](#)
- [Data Ingestion Performance for Multi-Master Clusters \(p. 713\)](#)
- [Exporting Data from a Multi-Master Cluster \(p. 713\)](#)
- [High Availability Considerations for Aurora Multi-Master Clusters \(p. 714\)](#)
- [Replication Between Multi-Master Clusters and Other Clusters \(p. 714\)](#)
- [Upgrading a Multi-Master Cluster \(p. 714\)](#)

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 \(p. 727\)](#).

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 \(p. 715\)](#)
- [Connection Management for Multi-Master Clusters \(p. 716\)](#)
- [Consistency Model for Multi-Master Clusters \(p. 717\)](#)
- [Multi-Master Clusters and Transactions \(p. 718\)](#)
- [Write Conflicts and Deadlocks in Multi-Master Clusters \(p. 718\)](#)
- [Multi-Master Clusters and Locking Reads \(p. 719\)](#)
- [Performing DDL Operations on a Multi-Master Cluster \(p. 720\)](#)
- [Using Autoincrement Columns \(p. 721\)](#)
- [Multi-Master Clusters Feature Reference \(p. 722\)](#)

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 \(p. 720\)](#).
- 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 \(p. 721\)](#).

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 \(p. 720\)](#).
- 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 \(p. 721\)](#).
- 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 \(p. 723\)](#).

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 \(p. 717\)](#).

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 \(p. 724\)](#).

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 \(p. 717\)](#).
- 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 executed later in the transaction are as fast as if executed 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 executed 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 executed. A locking read executed 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 execution 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 executed 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. You can check whether that DB instance is running an additional server process that performs the underlying processing for DDL statements by using the `aurora_mm_has_primary` status variable. The following example shows how.

```
mysql> select @@aurora_server_id, @@aurora_has_primary;
+-----+-----+
| @@aurora_server_id | @@aurora_mm_has_primary |
+-----+-----+
| mmr-demo-test-mm-3-1 | false           |
+-----+-----+
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 \(p. 723\)](#).

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 \(p. 712\)](#).

Topics

- [Query Performance for Multi-Master Clusters \(p. 725\)](#)
- [Conflict Resolution for Multi-Master Clusters \(p. 725\)](#)
- [Optimizing Buffer Pool and Dictionary Cache Usage \(p. 725\)](#)

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 \(p. 724\)](#).

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 \(p. 725\)](#).
- If you need read-write workload isolation, designate one or more DB instances as read-only, as described in [Using Instance Read-Only Mode \(p. 724\)](#). 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 \(p. 726\)](#)
- [Using a Multi-Master Cluster Without Sharding \(p. 726\)](#)
- [Using a Multi-Master Cluster as an Active Standby \(p. 727\)](#)

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 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 use it with 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 with an Aurora MySQL Native Function \(p. 753\)](#).
- 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 \(p. 739\)](#).

- 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 \(p. 747\)](#).
- Automatically add or remove Aurora Replicas with Application Auto Scaling. For more information, see [Using Amazon Aurora Auto Scaling with Aurora Replicas \(p. 351\)](#).
- Perform sentiment analysis with Amazon Comprehend, or a wide variety of machine learning algorithms with Amazon SageMaker. For more information, see [Using Machine Learning \(ML\) Capabilities with Amazon Aurora \(p. 366\)](#).

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 \(p. 728\)](#).

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

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 \(p. 728\)](#).

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 \(p. 738\)](#).

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 \(p. 753\)](#).
- 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 \(p. 739\)](#).
- 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 \(p. 747\)](#).
- Export log data to Amazon CloudWatch Logs MySQL. For more information, see [Publishing Amazon Aurora MySQL Logs to Amazon CloudWatch Logs \(p. 759\)](#).
- Automatically add or remove Aurora Replicas with Application Auto Scaling. For more information, see [Using Amazon Aurora Auto Scaling with Aurora Replicas \(p. 351\)](#).

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 \(p. 729\)](#)
 - [Creating an IAM Policy to Access AWS Lambda Resources \(p. 730\)](#)
 - [Creating an IAM Policy to Access CloudWatch Logs Resources \(p. 731\)](#)
 - [Creating an IAM Policy to Access AWS KMS Resources \(p. 733\)](#)
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 \(p. 734\)](#).
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 \(p. 734\)](#).

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

Note

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 additional permissions** 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 \(p. 734\)](#).

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 \(p. 734\)](#).

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": "EnableCreationAndManagementOfRDSCloudwatchLogGroups",  
            "Effect": "Allow",  
            "Action": "logs:CreateLogGroup",  
            "Resource": "arn:aws:logs:  
                <region>:  
                <account>/log-group/<log_group_name>  
        },  
        {  
            "Sid": "EnableLogStreamCreation",  
            "Effect": "Allow",  
            "Action": "logs:CreateLogStream",  
            "Resource": "arn:aws:logs:  
                <region>:  
                <account>/log-group/<log_group_name>/log-stream/<log_stream_name>  
        },  
        {  
            "Sid": "EnableLogDelivery",  
            "Effect": "Allow",  
            "Action": "logs:PutLogEvents",  
            "Resource": "arn:aws:logs:  
                <region>:  
                <account>/log-group/<log_group_name>/log-stream/<log_stream_name>  
        }  
    ]  
}
```

```
        "Effect": "Allow",
        "Action": [
            "logs:CreateLogGroup",
            "logs:PutRetentionPolicy"
        ],
        "Resource": [
            "arn:aws:logs:*::log-group:/aws/rds/*"
        ]
    },
    {
        "Sid": "EnableCreationAndManagementOfRDSCloudwatchLogStreams",
        "Effect": "Allow",
        "Action": [
            "logs:CreateLogStream",
            "logs:PutLogEvents",
            "logs:DescribeLogStreams",
            "logs:GetLogEvents"
        ],
        "Resource": [
            "arn:aws:logs:*::log-group:/aws/rds/*:log-stream:*
        ]
    }
]
```

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**, 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 `log-group:/aws/rds/*` for **Log Group Name**, and choose **Add**.
8. Choose **Add ARN for log-stream**.
9. In the **Add ARN(s)** dialog box, enter the following values:
 - **Log Group Name** – `log-group:/aws/rds/*`
 - **Log Stream** – `log-stream`
 - **Log Stream Name** – `*`

10. In the **Add ARN(s)** dialog box, choose **Add**
11. Choose **Review policy**.
12. 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.
13. Choose **Create policy**.
14. Complete the steps in [Creating an IAM Role to Allow Amazon Aurora to Access AWS Services \(p. 734\)](#).

Creating an IAM Policy to Access AWS KMS Resources

Aurora can access AWS Key Management Service 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 ID.
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 \(p. 734\)](#).

Creating an IAM Role to Allow Amazon Aurora to Access AWS Services

After creating an IAM policy to allow Aurora to access AWS resources, you must create an IAM role and attach the IAM policy to the new IAM role.

To create an IAM role to permit your Amazon RDS cluster to communicate with other AWS services on your behalf, take the following steps.

To create an IAM role to allow Amazon RDS to access AWS services

1. Open the [IAM console](#).
2. In the navigation pane, choose **Roles**.
3. Choose **Create role**.
4. Under **AWS service**, choose **RDS**.
5. Under **Select your use case**, choose **RDS – Add Role to Database**.
6. Choose **Next: Permissions**.
7. On the **Attach permissions policies** page, enter the name of your policy in the **Search** field.
8. When it appears in the list, select the policy that you defined earlier using the instructions in one of the following sections:
 - [Creating an IAM Policy to Access Amazon S3 Resources \(p. 729\)](#)
 - [Creating an IAM Policy to Access AWS Lambda Resources \(p. 730\)](#)
 - [Creating an IAM Policy to Access CloudWatch Logs Resources \(p. 731\)](#)
 - [Creating an IAM Policy to Access AWS KMS Resources \(p. 733\)](#)
9. Choose **Next: Tags**, and then choose **Next: Review**.
10. In **Role name**, enter a name for your IAM role, for example `RDSLoadFromS3`. You can also add an optional **Role description** value.
11. Choose **Create Role**.
12. Complete the steps in [Associating an IAM Role with an Amazon Aurora MySQL DB Cluster \(p. 734\)](#).

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 \(p. 734\)](#) with that 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.

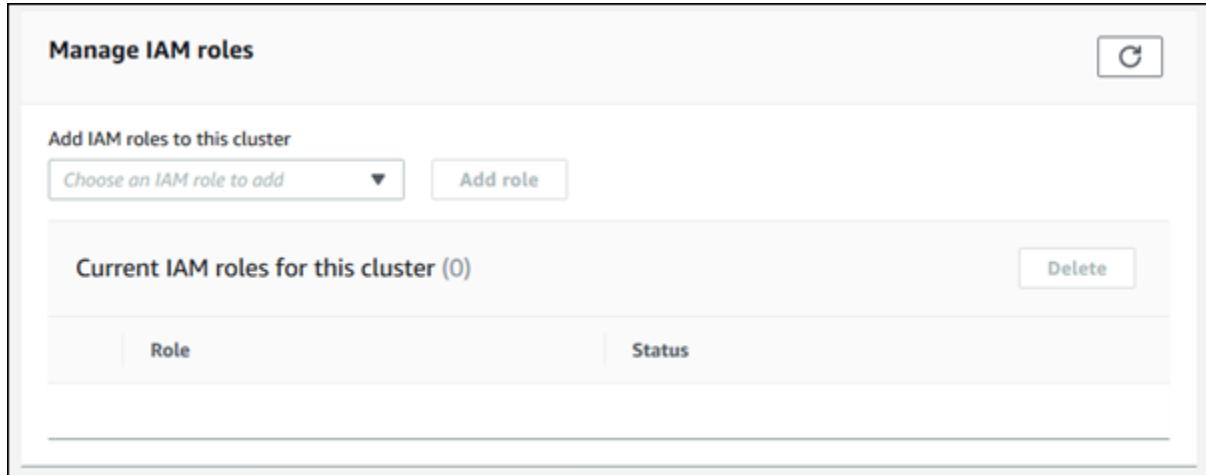
Cluster-Level Parameter	Description
<code>aws_default_lambda_role</code>	Use when invoking a Lambda function from your DB cluster.

Cluster-Level Parameter	Description	
<code>aws_default_logs_role</code>	This parameter is no longer required for exporting log data from your DB cluster to Amazon CloudWatch Logs. Aurora MySQL now uses a service-linked role for the required permissions. For more information about service-linked roles, see Using Service-Linked Roles for Amazon Aurora (p. 235) .	
<code>aws_default_s3_role</code>	<p>Used when invoking the <code>LOAD DATA FROM S3</code>, <code>LOAD XML FROM S3</code>, or <code>SELECT INTO OUTFILE S3</code> statement from your DB cluster.</p> <p>The IAM role specified in this parameter is used only if an IAM role isn't specified for <code>aurora_load_from_s3_role</code> or <code>aurora_select_into_s3_role</code> for the appropriate statement.</p> <p>For earlier versions of Aurora, the IAM role specified for this parameter is always used.</p>	
<code>aurora_load_from_s3_role</code>	<p>Used 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 earlier versions of Aurora, this parameter is not available.</p>	
<code>aurora_select_into_s3_role</code>	<p>Used 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 earlier versions of Aurora, this parameter is not available.</p>	

To associate an IAM role to permit your Amazon RDS cluster to communicate with other AWS services on your behalf, take the following steps.

To associate an IAM role with an Aurora DB cluster using the console

1. Open the RDS console at <https://console.aws.amazon.com/rds/>.
2. Choose **Databases**.
3. Choose the name of the Aurora DB cluster that you want to associate an IAM role with to show its details.
4. On the **Connectivity & security** tab, in the **Manage IAM roles** section, choose the role to add under **Add IAM roles to this cluster**.



5. Choose **Add role**.
6. (Optional) To stop associating an IAM role with a DB cluster and remove the related permission, choose the role and choose **Delete**.
7. In the RDS console, choose **Parameter groups** in the navigation pane.
8. If you are already using a custom DB parameter group, you can select that group to use instead of creating a new DB cluster parameter group. If you are using the default DB cluster parameter group, create a new DB cluster parameter group, as described in the following steps:
 - a. Choose **Create parameter group**.
 - b. For **Parameter group family**, choose `aurora5.6` for an Aurora MySQL 5.6-compatible DB cluster, or choose `aurora-mysql5.7` for an Aurora MySQL 5.7-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.

Create parameter group

Parameter group details
To create a parameter group, select a parameter group family, then name and describe your parameter group

Parameter group family
DB family that this DB parameter group will apply to

Type

Group name
Identifier for the DB parameter group

Description
Description for the DB parameter group

Cancel **Create**

- f. Choose **Create**.
9. On the **Parameter groups** page, select your DB cluster parameter group and choose **Edit** for **Parameter group actions**.
10. Set the appropriate cluster-level parameters to the related IAM role ARN values. For example, you can set just the `aws_default_s3_role` parameter to `arn:aws:iam::123456789012:role/AllowAuroraS3Role`.
11. Choose **Save changes**.
12. To change the DB cluster parameter group for your DB cluster, complete the following steps:
 - a. Choose **Databases**, and then choose your Aurora DB cluster.
 - b. Choose **Modify**.
 - c. Scroll to **Database options** and set **DB cluster parameter group** to the DB cluster parameter group.
 - d. Choose **Continue**.
 - e. Verify your changes and then choose **Apply immediately**.
 - f. Choose **Modify cluster**.
 - g. Choose **Databases**, and then choose the primary instance for your DB cluster.
 - h. For **Actions**, choose **Reboot**.

When the instance has rebooted, your IAM role is associated with your DB cluster.

For more information about cluster parameter groups, see [Aurora MySQL Parameters \(p. 773\)](#).

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`.

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 Parameters \(p. 773\)](#).

Enabling Network Communication from Amazon Aurora MySQL to Other AWS Services

To invoke AWS Lambda functions or access files from Amazon S3, the network configuration of your Aurora DB cluster must allow outbound connections to endpoints for those services. 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
```

If you encounter these messages while invoking AWS Lambda functions or accessing files from Amazon S3, check if your Aurora DB cluster is public or private. If your Aurora DB cluster is private, you must configure it to enable connections.

For an Aurora DB cluster to be public, it must be marked as publicly accessible. If you look at the details for the DB cluster in the AWS Management Console, **Publicly Accessible** is **Yes** if this is the case. The DB cluster must also be in an Amazon VPC public subnet. For more information about publicly accessible DB instances, see [Working with a DB Instance in a VPC \(p. 249\)](#). 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 invoke AWS Lambda functions or access Amazon S3 files. If so, configure the cluster so it can connect to Internet addresses through Network Address Translation (NAT). As an alternative for Amazon S3, you can instead configure the VPC to have a VPC endpoint for Amazon S3 associated with the DB cluster's route table. For more information about configuring NAT in your VPC, see [NAT Gateways](#). For more information about configuring VPC endpoints, see [VPC Endpoints](#).

Related Topics

- [Integrating Aurora with Other AWS Services \(p. 350\)](#)
- [Managing an Amazon Aurora DB Cluster \(p. 260\)](#)

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.

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

This feature currently isn't available for Aurora Serverless clusters.

Giving Aurora Access to Amazon S3

Before you can load data from an Amazon S3 bucket, you must first give your Aurora MySQL DB cluster permission to access Amazon S3.

To give Aurora MySQL access to Amazon S3

1. Create an AWS Identity and Access Management (IAM) policy that provides the bucket and object permissions that allow your Aurora MySQL DB cluster to access Amazon S3. For instructions, see [Creating an IAM Policy to Access Amazon S3 Resources \(p. 729\)](#).
2. Create an IAM role, and attach the IAM policy you created in [Creating an IAM Policy to Access Amazon S3 Resources \(p. 729\)](#) to the new IAM role. For instructions, see [Creating an IAM Role to Allow Amazon Aurora to Access AWS Services \(p. 734\)](#).
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 \(p. 290\)](#).

4. Set either the `aurora_load_from_s3_role` or `aws_default_s3_role` DB cluster parameter to the Amazon Resource Name (ARN) of the new IAM role. If an IAM role isn't specified for `aurora_load_from_s3_role`, Aurora uses the IAM role specified in `aws_default_s3_role`.

If the cluster is part of an Aurora global database, set this parameter for each Aurora cluster in the global database. Although only the primary cluster in an Aurora global database can load data, another cluster might be promoted by the failover mechanism and become the primary cluster.

For more information about DB cluster parameters, see [Amazon Aurora DB Cluster and DB Instance Parameters \(p. 288\)](#).

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 \(p. 734\)](#) 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 \(p. 734\)](#).
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 \(p. 738\)](#).

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 be granted the `LOAD FROM S3` privilege to issue either statement. The master user name for a DB cluster is granted the `LOAD FROM S3` privilege by default. You can grant the privilege to another user by using the following statement.

```
GRANT LOAD FROM S3 ON *.* TO 'user'@'domain-or-ip-address'
```

The `LOAD FROM S3` privilege is specific to Amazon Aurora and is not available for MySQL databases or RDS 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 LOAD FROM S3` statement causes replication to stop with an error. You can safely skip the error to resume replication. To skip the error on an RDS 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.

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 \(p. 742\)](#).

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 \(p. 740\)](#).
- **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.
- **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 carriage return 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.

- **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 execution of the statement, use the `WHERE` clause to filter the records on the Amazon S3 URI for the manifest file used in the statement. If you have used the same manifest file before, filter the results using the `timestamp` field.

```
select * from mysql.aurora_s3_load_history where load_prefix = 'S3_URI';
```

The following table describes the fields in the `aurora_s3_load_history` table.

Field	Description
<code>load_prefix</code>	The URI that was specified in the load statement. This URI can map to any of the following: <ul style="list-style-type: none"> A single data file for a <code>LOAD DATA FROM S3 FILE</code> statement An Amazon S3 prefix that maps to multiple data files for a <code>LOAD DATA FROM S3 PREFIX</code> statement A single manifest file that contains the names of files to be loaded for a <code>LOAD DATA FROM S3 MANIFEST</code> statement
<code>file_name</code>	The name of a file that was loaded into Aurora from Amazon S3 using the URI identified in the <code>load_prefix</code> field.
<code>version_number</code>	The version number of the file identified by the <code>file_name</code> field that was loaded, if the Amazon S3 bucket has a version number.
<code>bytes_loaded</code>	The size of the file loaded, in bytes.
<code>load_timestamp</code>	The timestamp when the <code>LOAD DATA FROM S3</code> statement completed.

Examples

The following statement loads data from an Amazon S3 bucket that is in the same region as the Aurora DB cluster. The statement reads the comma-delimited data in the file `customerdata.txt` that is in the `dbbucket` Amazon S3 bucket, and then loads the data into the table `store-schema.customer-table`.

```
LOAD DATA FROM S3 's3://dbbucket/customerdata.csv'
  INTO TABLE store-schema.customer-table
  FIELDS TERMINATED BY ','
  LINES TERMINATED BY '\n'
  (ID, FIRSTNAME, LASTNAME, ADDRESS, EMAIL, PHONE);
```

The following statement loads data from an Amazon S3 bucket that is in a different region from the Aurora DB cluster. The statement reads the comma-delimited data from all files that match the `employee-data` object prefix in the `my-data` Amazon S3 bucket in the `us-west-2` region, and then loads the data into the `employees` table.

```
LOAD DATA FROM S3 PREFIX 's3-us-west-2://my-data/employee_data'
```

```
INTO TABLE employees
FIELDS TERMINATED BY ','
LINES TERMINATED BY '\n'
(ID, FIRSTNAME, LASTNAME, EMAIL, SALARY);
```

The following statement loads data from the files specified in a JSON manifest file named q1_sales.json into the sales table.

```
LOAD DATA FROM S3 MANIFEST 's3-us-west-2://aurora-bucket/q1_sales.json'
INTO TABLE sales
FIELDS TERMINATED BY ','
LINES TERMINATED BY '\n'
(MONTH, STORE, GROSS, NET);
```

LOAD XML FROM S3

You can use the `LOAD XML FROM S3` statement to load data from XML files stored on an Amazon S3 bucket in one of three different XML formats:

- Column names as attributes of a `<row>` element. The attribute value identifies the contents of the table field.

```
<row column1="value1" column2="value2" .../>
```

- Column names as child elements of a `<row>` element. The value of the child element identifies the contents of the table field.

```
<row>
  <column1>value1</column1>
  <column2>value2</column2>
</row>
```

- Column names in the `name` attribute of `<field>` elements in a `<row>` element. The value of the `<field>` element identifies the contents of the table field.

```
<row>
  <field name='column1'>value1</field>
  <field name='column2'>value2</field>
</row>
```

Syntax

```
LOAD XML FROM S3 'S3-URI'
[REPLACE | IGNORE]
INTO TABLE tbl_name
[PARTITION (partition_name,...)]
[CHARACTER SET charset_name]
[ROWS IDENTIFIED BY '<element-name>']
[IGNORE number {LINES | ROWS}]
[(field_name_or_user_var,...)]
[SET col_name = expr,...]
```

Parameters

Following, you can find a list of the required and optional parameters used by the `LOAD DATA FROM S3` statement. You can find more details about some of these parameters in [LOAD XML Syntax](#) in the MySQL documentation.

- **FILE | PREFIX** – Identifies whether to load the data from a single file, or from all files that match a given prefix. **FILE** is the default.
- **REPLACE | IGNORE** – Determines what action to take if an input row has the same unique key values as an existing row in the database table.
 - Specify **REPLACE** if you want the input row to replace the existing row in the table.
 - Specify **IGNORE** if you want to discard the input row. **IGNORE** is the default.
- **INTO TABLE** – Identifies the name of the database table to load the input rows into.
- **PARTITION** – Requires that all input rows be inserted into the partitions identified by the specified list of comma-separated partition names. If an input row cannot be inserted into one of the specified partitions, then the statement fails and an error is returned.
- **CHARACTER SET** – Identifies the character set of the data in the input file.
- **ROWS IDENTIFIED BY** – Identifies the element name that identifies a row in the input file. The default is `<row>`.
- **IGNORE *number* LINES | ROWS** – Specifies to ignore a certain number of lines or rows at the start of the input file. For example, you can use **IGNORE 1 LINES** to skip over the first line in the text file, or **IGNORE 2 ROWS** to skip over the first two rows of data in the input XML.
- **field_name_or_user_var, ...** – Specifies a comma-separated list of one or more XML element names or user variables that identify which elements to load by name. The name of a user variable used for this purpose must match the name of an element from the XML file, prefixed with `@`. You can employ user variables to store the corresponding field values for subsequent reuse.

For example, the following statement loads the first column from the input file into the first column of `table1`, and sets the value of the `table_column2` column in `table1` to the input value of the second column divided by 100.

```
LOAD XML FROM S3 's3://mybucket/data.xml'
  INTO TABLE table1
    (column1, @var1)
    SET table_column2 = @var1/100;
```

- **SET** – Specifies a comma-separated list of assignment operations that set the values of columns in the table to values not included in the input file.

For example, the following statement sets the first two columns of `table1` to the values in the first two columns from the input file, and then sets the value of the `column3` in `table1` to the current time stamp.

```
LOAD XML FROM S3 's3://mybucket/data.xml'
  INTO TABLE table1
    (column1, column2)
    SET column3 = CURRENT_TIMESTAMP;
```

You can use subqueries in the right side of **SET** assignments. For a subquery that returns a value to be assigned to a column, you can use only a scalar subquery. Also, you cannot use a subquery to select from the table that is being loaded.

Related Topics

- [Integrating Amazon Aurora MySQL with Other AWS Services \(p. 727\)](#)
- [Saving Data from an Amazon Aurora MySQL DB Cluster into Text Files in an Amazon S3 Bucket \(p. 747\)](#)
- [Managing an Amazon Aurora DB Cluster \(p. 260\)](#)
- [Migrating Data to an Amazon Aurora DB Cluster \(p. 172\)](#)

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 \(p. 739\)](#).

Note

This feature currently isn't available for Aurora Serverless clusters.

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 \(p. 729\)](#).
2. Create an IAM role, and attach the IAM policy you created in [Creating an IAM Policy to Access Amazon S3 Resources \(p. 729\)](#) to the new IAM role. For instructions, see [Creating an IAM Role to Allow Amazon Aurora to Access AWS Services \(p. 734\)](#).
3. Set either the `aurora_select_into_s3_role` or `aws_default_s3_role` DB cluster parameter to the Amazon Resource Name (ARN) of the new IAM role. If an IAM role isn't specified for `aurora_select_into_s3_role`, Aurora uses the IAM role specified in `aws_default_s3_role`.

If the cluster is part of an Aurora global database, set this parameter for each Aurora cluster in the global database.

For more information about DB cluster parameters, see [Amazon Aurora DB Cluster and DB Instance Parameters \(p. 288\)](#).

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 \(p. 734\)](#) 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 \(p. 734\)](#).

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 \(p. 738\)](#).

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 be granted the `SELECT INTO S3` privilege to issue the statement. The master user name for a DB cluster is granted the `SELECT`

INTO S3 privilege by default. You can grant the privilege to another user by using the following statement.

```
GRANT SELECT INTO S3 ON *.* TO 'user'@'domain-or-ip-address'
```

The `SELECT INTO S3` privilege is specific to Amazon Aurora MySQL and is not available for MySQL databases or RDS 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 SELECT INTO S3` statement causes replication to stop with an error. You can safely skip the error to resume replication. To skip the error on an RDS 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.

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 \(p. 742\)](#).

The data files included in the manifest created by the `SELECT INTO OUTFILE S3` statement are listed in the order that they're created by the statement. For example, suppose that a `SELECT INTO OUTFILE`

S3 statement specified `s3-us-west-2://bucket/prefix` as the path in which to store data files and creates three data files and a manifest file. The specified Amazon S3 bucket contains a manifest file named `s3-us-west-2://bucket/prefix.manifest`, that contains the following information.

```
{
  "entries": [
    {
      "url": "s3-us-west-2://bucket/prefix.part_00000"
    },
    {
      "url": "s3-us-west-2://bucket/prefix.part_00001"
    },
    {
      "url": "s3-us-west-2://bucket/prefix.part_00002"
    }
  ]
}
```

SELECT INTO OUTFILE S3

You can use the `SELECT INTO OUTFILE S3` statement to query data from a DB cluster and save it directly into delimited text files stored in an Amazon S3 bucket. Compressed or encrypted files are not supported.

Syntax

```

SELECT
  [ALL | DISTINCT | DISTINCTROW ]
  [HIGH_PRIORITY]
  [STRAIGHT_JOIN]
  [SQL_SMALL_RESULT] [SQL_BIG_RESULT] [SQL_BUFFER_RESULT]
  [SQL_CACHE | SQL_NO_CACHE] [SQL_CALC_FOUND_ROWS]
  select_expr [, select_expr ...]
  [FROM table_references
    [PARTITION partition_list]
  [WHERE where_condition]
  [GROUP BY {col_name | expr | position}
    [ASC | DESC], ... [WITH ROLLUP]]
  [HAVING where_condition]
  [ORDER BY {col_name | expr | position}
    [ASC | DESC], ...]
  [LIMIT {[offset,] row_count | row_count OFFSET offset}]
  [PROCEDURE procedure_name(argument_list)]
  INTO OUTFILE S3 's3_uri'
  [CHARACTER SET charset_name]
  [export_options]
  [MANIFEST {ON | OFF}]
  [OVERWRITE {ON | OFF}]

  export_options:
  [{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 \(p. 748\)](#).
- **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 \(p. 739\)](#).

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 \(p. 748\)](#).

- **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 execute 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 execution error, because only a portion of data needs to be re-selected and uploaded to Amazon S3 if an error occurs during the execution of a particular statement. 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 execution.

Currently, there's no way to directly monitor the progress of the `SELECT INTO OUTFILE S3` statement during execution. 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 during execution.

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 \(p. 350\)](#)
- [Loading Data into an Amazon Aurora MySQL DB Cluster from Text Files in an Amazon S3 Bucket \(p. 739\)](#)
- [Managing an Amazon Aurora DB Cluster \(p. 260\)](#)
- [Migrating Data to an Amazon Aurora DB Cluster \(p. 172\)](#)

Invoking a Lambda Function from an Amazon Aurora MySQL DB Cluster

You can invoke an AWS Lambda function from an Amazon Aurora with MySQL compatibility DB cluster with a native function or a stored procedure. Before invoking a Lambda function from an Aurora MySQL, the Aurora DB cluster must have access to Lambda.

In recent Aurora MySQL versions, 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.

Topics

- [Giving Aurora Access to Lambda \(p. 752\)](#)
- [Invoking a Lambda Function with an Aurora MySQL Native Function \(p. 753\)](#)
- [Invoking a Lambda Function with an Aurora MySQL Stored Procedure \(p. 755\)](#)

Giving Aurora Access to Lambda

Before you can invoke Lambda functions from an Aurora MySQL, you must first give your Aurora MySQL DB 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 \(p. 730\)](#).

2. Create an IAM role, and attach the IAM policy you created in [Creating an IAM Policy to Access AWS Lambda Resources \(p. 730\)](#) to the new IAM role. For instructions, see [Creating an IAM Role to Allow Amazon Aurora to Access AWS Services \(p. 734\)](#).
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 \(p. 288\)](#).

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 \(p. 734\)](#) 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 \(p. 734\)](#).

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 \(p. 738\)](#).

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. For more information about Aurora MySQL versions, see [Database Engine Updates for Amazon Aurora MySQL \(p. 794\)](#).

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 invoked function's execution before moving on to another action, use the synchronous function `lambda_sync`. When you don't need to know the result of the execution before moving on to another action, use the asynchronous function `lambda_async`.

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 master 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 INVOKE LAMBDA 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 executed once per SQL statement, but once per row modified, one row at a time. Trigger execution is synchronous, and the data-modifying statement will not return until trigger execution 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 doesn't support JSON parsing. 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 doesn't support JSON parsing. 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"}');
```

Related Topics

- [Integrating Aurora with Other AWS Services \(p. 350\)](#)
- [Managing an Amazon Aurora DB Cluster \(p. 260\)](#)
- [AWS Lambda Developer Guide](#)

Invoking a Lambda Function with an Aurora MySQL Stored Procedure

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

In Aurora MySQL version 1.8 and later, you can use the native function method instead of these stored procedures to invoke a Lambda function. Starting with Amazon Aurora version 1.16, the stored procedure `mysql.lambda_async` is deprecated. If you are using Aurora version 1.16 or later, we strongly recommend that you work with native Lambda functions instead. For more information about the native functions, see [Working with Native Functions to Invoke a Lambda Function \(p. 753\)](#).

Currently, in Aurora MySQL 2.* you cannot use the native function technique to invoke a lambda function. For an Aurora MySQL 5.7-compatible cluster, use the stored procedure technique described in the following section.

Working with the mysql.lambda_async Procedure to Invoke a Lambda Function

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!');
```

Related Topics

- [Integrating Aurora with Other AWS Services \(p. 350\)](#)
- [Managing an Amazon Aurora DB Cluster \(p. 260\)](#)

- [AWS Lambda Developer Guide](#)

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 \(p. 669\)](#).

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 \(p. 728\)](#). 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 \(p. 235\)](#).

- If you don't want to export audit logs to CloudWatch Logs, make sure that all methods of exporting audit logs are disabled. These methods are the AWS Management Console, the AWS CLI, the RDS API, and the `server_audit_logs_upload` parameter.
- The procedure is slightly different for Aurora Serverless clusters than for provisioned clusters. Serverless clusters automatically upload all the kinds of logs that you enable through the configuration parameters. Therefore, you enable or disable log upload for Serverless clusters by turning different log types on and off in the DB cluster parameter group. You don't modify the settings of the cluster itself through the AWS Management Console, AWS CLI, or RDS API. For information about enabling MySQL logs for Serverless clusters, see [Aurora Serverless and Parameter Groups \(p. 121\)](#).

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, OS X, 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, OS X, 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.

All of the events from all of the DB instances in a DB cluster are pushed to a log group using different log streams.

If a log group with the specified name exists, Aurora uses that log group to export log data for the Aurora DB cluster. You can use automated configuration, such as AWS CloudFormation, to create log groups with predefined log retention periods, metric filters, and customer access. Otherwise, a new log group is automatically created using the default log retention period, **Never Expire**, in CloudWatch Logs. You can use the CloudWatch Logs console, the AWS CLI, or the CloudWatch Logs API to change the log retention period. For more information about changing log retention periods in CloudWatch Logs, see [Change Log Data Retention in CloudWatch Logs](#).

You can use the CloudWatch Logs console, the AWS CLI, or the CloudWatch Logs API to search for information within the log events for a DB cluster. For more information about searching and filtering log data, see [Searching and Filtering Log Data](#).

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 \(p. 762\)](#).

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 \(p. 291\)](#). For information on parameter groups and Amazon Aurora, see [Aurora MySQL Parameters \(p. 773\)](#).

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. For more information about Aurora lab mode, see [Amazon Aurora MySQL Lab Mode \(p. 762\)](#).

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.
Fast DDL	This feature allows you to execute an <code>ALTER TABLE tbl_name ADD COLUMN col_name column_definition</code> operation nearly instantaneously. The operation completes without requiring the table to be copied and without materially impacting other DML statements. Since

Feature	Description
	it does not consume temporary storage for a table copy, it makes DDL statements practical even for large tables on small instance classes. Fast DDL is currently only supported for adding a nullable column, without a default value, at the end of a table. For more information about using this feature, see Altering Tables in Amazon Aurora Using Fast DDL (p. 642) .

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.

Topics

- [Determining Which DB Instance You Are Connected To \(p. 763\)](#)
- [Using T2 Instances \(p. 763\)](#)
- [Invoking an AWS Lambda Function \(p. 764\)](#)
- [Working with Asynchronous Key Prefetch in Amazon Aurora \(p. 765\)](#)
- [Working with Multi-Threaded Replication Slaves in Amazon Aurora MySQL \(p. 767\)](#)
- [Using Amazon Aurora to Scale Reads for Your MySQL Database \(p. 767\)](#)
- [Using Amazon Aurora for Disaster Recovery with Your MySQL Databases \(p. 770\)](#)
- [Migrating from MySQL to Amazon Aurora MySQL with Reduced Downtime \(p. 770\)](#)
- [Using XA Transactions with Amazon Aurora MySQL \(p. 770\)](#)
- [Working with Hash Joins in Aurora MySQL \(p. 771\)](#)
- [Working with Foreign Keys in Aurora MySQL \(p. 772\)](#)
- [Related Topics \(p. 773\)](#)

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 an Aurora Replica and `OFF` if you are connected to the primary instance.

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`.

Using T2 Instances

Amazon Aurora MySQL instances that use the `db.t2.small` or `db.t2.medium` DB instance classes are best suited for applications that do not support a high workload for an extended amount of time.

T2 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 db.t2.small and db.t2.medium DB instance classes for development and test servers, or other non-production servers. For more details on T2 instances, see [T2 Instances](#).

Do not enable the MySQL Performance Schema on Amazon Aurora MySQL T2 instances. If the Performance Schema is enabled, the T2 instance might run out of memory.

When you use a T2 instance as a DB instance in an Aurora MySQL DB cluster, we recommend the following:

- If you use a T2 instance as a DB instance class in your DB cluster, then we recommend that all instances in your DB cluster use the same DB instance class. For example, if you use db.t2.medium for your primary instance, then we recommend that you use db.t2.medium for your Aurora Replicas as well.
- 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 R3 DB instance classes (scale compute).

For more information on monitoring metrics, see [Monitoring Amazon Aurora DB Cluster Metrics \(p. 457\)](#).

- For your Aurora MySQL DB clusters using single-master replication, monitor the replica lag (`AuroraReplicaLag`) between the primary instance and the Aurora Replicas.

If an Aurora Replica runs out of CPU credits before the primary instance, the lag behind the primary instance results in the Aurora Replica frequently restarting. This result is common when an application has a heavy load of read operations distributed among Aurora Replicas in an Aurora MySQL DB cluster, at the same time that the primary 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 Aurora Replicas 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 R3 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 R3 DB instance classes (scale compute).

Invoking an AWS Lambda Function

If you are using Amazon Aurora version 1.16 or later, we recommend using the native functions `lambda_sync` and `lambda_async` to invoke Lambda functions.

If you are using the deprecated `mysql.lambda_async` procedure, we recommend that you wrap calls to the `mysql.lambda_async` procedure in a stored procedure. You can call this stored procedure from different sources, such as triggers or client code. This approach can help to avoid impedance mismatch issues and make it easier for your database programmers to invoke Lambda functions.

For more information on invoking Lambda functions from Amazon Aurora, see [Invoking a Lambda Function from an Amazon Aurora MySQL DB Cluster \(p. 752\)](#).

Working with Asynchronous Key Prefetch in Amazon Aurora

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

Amazon Aurora can use AKP to improve the performance of queries that join tables across indexes. This feature improves performance by anticipating the rows needed to run queries in which a JOIN query requires use of the Batched Key Access (BKA) Join algorithm and Multi-Range Read (MRR) optimization features. For more information about BKA and MRR, see [Block Nested-Loop and Batched Key Access Joins](#) and [Multi-Range Read Optimization](#) in the MySQL documentation.

To take advantage of the AKP feature, a query must use both BKA and MRR. Typically, such a query occurs when the JOIN clause of a query uses a secondary index, but also needs some columns from the primary index. For example, you can use AKP when a JOIN clause represents an equijoin on index values between a small outer and large inner table, and the index is highly selective on the larger table. AKP works in concert with BKA and MRR to perform a secondary to primary index lookup during the evaluation of the JOIN clause. AKP identifies the rows required to run the query during the evaluation of the JOIN clause. It then uses a background thread to asynchronously load the pages containing those rows into memory before running the query.

Enabling Asynchronous Key Prefetch

You can enable the AKP feature by setting `aurora_use_key_prefetch`, a MySQL server variable, to `on`. By default, this value is set to `on`. However, AKP cannot be enabled until you also enable the BKA Join algorithm and disable cost-based MRR functionality. To do so, you must set the following values for `optimizer_switch`, a MySQL server variable:

- Set `batched_key_access` to `on`. This value controls the use of the BKA Join algorithm. By default, this value is set to `off`.
- Set `mrr_cost_based` to `off`. This value controls the use of cost-based MRR functionality. By default, this value is set to `on`.

Currently, you can set these values only at the session level. The following example illustrates how to set these values to enable AKP for the current session by executing SET statements.

```
mysql> set @@session.aurora_use_key_prefetch=on;
mysql> set @@session.optimizer_switch='batched_key_access=on,mrr_cost_based=off';
```

Similarly, you can use SET statements to disable AKP and the BKA Join algorithm and re-enable cost-based MRR functionality for the current session, as shown in the following example.

```
mysql> set @@session.aurora_use_key_prefetch=off;
```

```
mysql> set @@session.optimizer_switch='batched_key_access=off,mrr_cost_based=on';
```

For more information about the **batched_key_access** and **mrr_cost_based** optimizer switches, see [Switchable Optimizations](#) in the MySQL documentation.

Optimizing Queries for Asynchronous Key Prefetch

You can confirm whether a query can take advantage of the AKP feature. To do so, use the EXPLAIN statement with the EXTENDED keyword to profile the query before running it. The *EXPLAIN statement* provides information about the execution plan to use for a specified query.

In the output for the EXPLAIN statement, the Extra column describes additional information included with the execution plan. If the AKP feature applies to a table used in the query, this column includes one of the following values:

- Using Key Prefetching
- Using join buffer (Batched Key Access with Key Prefetching)

The following example shows use of EXPLAIN with EXTENDED to view the execution plan for a query that can take advantage of AKP.

```
mysql> explain extended select sql_no_cache
    ->     ps_partkey,
    ->     sum(ps_supplycost * ps_availqty) as value
-> from
->     partsupp,
->     supplier,
->     nation
-> where
->     ps_suppkey = s_suppkey
->     and s_nationkey = n_nationkey
->     and n_name = 'ETHIOPIA'
-> group by
->     ps_partkey having
->     sum(ps_supplycost * ps_availqty) > (
->         select
->             sum(ps_supplycost * ps_availqty) * 0.0000003333
->         from
->             partsupp,
->             supplier,
->             nation
->         where
->             ps_suppkey = s_suppkey
->             and s_nationkey = n_nationkey
->             and n_name = 'ETHIOPIA'
->     )
-> order by
->     value desc;
+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+
+-----+
| id | select_type | table      | type   | possible_keys          | key           | key_len |
| ref          |              |           |        | rows | filtered | Extra
|       |              |           |        |       |          |          |
+-----+-----+-----+-----+-----+
| 1  | PRIMARY     | nation     | ALL    | PRIMARY                | NULL          | NULL      |
| NULL          |              |           |        | 25  | 100.00  | Using where; Using temporary; Using
| filesort      |              |           |        |       |          |          |
+-----+
```

```
| 1 | PRIMARY      | supplier | ref  | PRIMARY,i_s_nationkey | i_s_nationkey | 5      |
| dbt3_scale_10.nation.n_nationkey | 2057 |   100.00 | Using index
|
| 1 | PRIMARY      | partsupp | ref  | i_ps_suppkey          | i_ps_suppkey | 4      |
| dbt3_scale_10.supplier.s_suppkey |   42 |   100.00 | Using join buffer (Batched Key Access
with Key Prefetching) |
| 2 | SUBQUERY     | nation   | ALL   | PRIMARY                | NULL         | NULL    |
| NULL                         |       | 25    |   100.00 | Using where
|
| 2 | SUBQUERY     | supplier | ref  | PRIMARY,i_s_nationkey | i_s_nationkey | 5      |
| dbt3_scale_10.nation.n_nationkey | 2057 |   100.00 | Using index
|
| 2 | SUBQUERY     | partsupp | ref  | i_ps_suppkey          | i_ps_suppkey | 4      |
| dbt3_scale_10.supplier.s_suppkey |   42 |   100.00 | Using join buffer (Batched Key Access
with Key Prefetching) |
+-----+-----+-----+-----+
+-----+-----+
+-----+
6 rows in set, 1 warning (0.00 sec)
```

For more information about the extended EXPLAIN output format, see [Extended EXPLAIN Output Format](#) in the MySQL product documentation.

Working with Multi-Threaded Replication Slaves in Amazon Aurora MySQL

By default, Aurora uses single-threaded replication when an Aurora MySQL DB cluster is used as a replication slave. While Amazon Aurora doesn't prohibit multithreaded replication, Aurora MySQL has inherited several issues regarding multithreaded replication from MySQL. We recommend that you do not use multithreaded replication in production. If you do use multi-threaded replication, we recommend that you test any use thoroughly.

For more information about using replication in Amazon Aurora, see [Replication with Amazon Aurora \(p. 47\)](#).

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 replication slave of your MySQL DB instance. This applies to an Amazon RDS 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 \(p. 100\)](#).

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 master instance until you have verified that they have been applied to the Aurora Replica. This maintenance ensures that you can restore your master 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 \(p. 1007\)](#).

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 Master 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 an Amazon RDS MySQL or MariaDB DB Instance with Reduced Downtime](#) in the *Amazon Relational Database Service User Guide*.

For Linux, OS X, 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, `mydbccluster.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 Master or Slave 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 replication slave before setting up replication. If you need to reestablish replication with the DB cluster as a replication slave, 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.

```
CALL mysql.rds_set_external_master ('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 \(p. 280\)](#).

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 replication slave of your MySQL DB instance. This applies to an Amazon RDS 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 replication slave of your MySQL DB instance, follow the procedure in [Using Amazon Aurora to Scale Reads for Your MySQL Database \(p. 767\)](#).

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 an Amazon RDS 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 Amazon RDS 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.

Using 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.

Working with Hash Joins in Aurora MySQL

When you need to join a large amount of data by using an equijoin, a hash join can improve query performance. You can enable hash joins for Aurora MySQL.

A hash join column can be any complex expression. In a hash join column, you can compare across data types in the following ways:

- You can compare anything across the category of precise numeric data types, such as `int`, `bigint`, `numeric`, and `bit`.
- You can compare anything across the category of approximate numeric data types, such as `float` and `double`.
- You can compare items across string types if the string types have the same character set and collation.
- You can compare items with date and timestamp data types if the types are the same.

Note

Data types in different categories cannot compare.

The following restrictions apply to hash joins for Aurora MySQL:

- Left-right outer joins are not supported.
- Semijoins such as subqueries are not supported, unless the subqueries are materialized first.
- Multiple-table updates or deletes are not supported.

Note

Single-table updates or deletes are supported.

- BLOB and spatial data type columns cannot be join columns in a hash join.

Enabling Hash Joins

To enable hash joins, set the MySQL server variable `optimizer_switch` to `on`. The `optimizer_switch` parameter is set to `on` for hash joins by default. The following example illustrates how to enable hash joins.

```
mysql> SET optimizer_switch='hash_join=on';
```

With this setting, the optimizer chooses to use a hash join based on cost, query characteristics, and resource availability. If the cost estimation is incorrect, you can force the optimizer to choose a hash join. You do so by setting `hash_join_cost_based`, a MySQL server variable, to `off`. The following example illustrates how to force the optimizer to choose a hash join.

```
mysql> SET optimizer_switch='hash_join_cost_based=off';
```

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.

Working with Foreign Keys in Aurora MySQL

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.

Related Topics

- [Managing an Amazon Aurora DB Cluster \(p. 260\)](#)

Amazon Aurora MySQL Reference

This reference includes information about Aurora MySQL parameters and status variables.

Topics

- [Aurora MySQL Parameters \(p. 773\)](#)
- [Inapplicable MySQL Parameters and Status Variables \(p. 785\)](#)
- [Aurora MySQL Events \(p. 785\)](#)
- [Aurora MySQL Isolation Levels \(p. 787\)](#)
- [Aurora MySQL Stored Procedures \(p. 791\)](#)

Aurora MySQL 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.

For more information on DB parameter groups, see [Working with DB Parameter Groups and DB Cluster Parameter Groups \(p. 286\)](#). For rules and restrictions for Aurora Serverless clusters, see [Aurora Serverless and Parameter Groups \(p. 121\)](#).

Topics

- [Cluster-Level Parameters \(p. 774\)](#)
- [Instance-Level Parameters \(p. 776\)](#)

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_enable_replica_log_compress	Yes	Doesn't apply to clusters that are part of an Aurora global database.
aurora_enable_repl_bin_log_filtering	Yes	Doesn't apply to clusters that are part of an Aurora global database.
aurora_enable_zdr	Yes	For more information, see High Availability Considerations for Amazon Aurora MySQL Replication (p. 674) .
aurora_load_from_s3_role	Yes	For more information, see Loading Data into an Amazon Aurora MySQL DB Cluster from Text Files in an Amazon S3 Bucket (p. 739) .
aurora_select_into_s3_role	Yes	For more information, see Saving Data from an Amazon Aurora MySQL DB Cluster into Text Files in an Amazon S3 Bucket (p. 747) .
auto_increment_increment	Yes	
auto_increment_offset	Yes	
aws_default_lambda_role	Yes	For more information, see Invoking a Lambda Function from an Amazon Aurora MySQL DB Cluster (p. 752) .
aws_default_s3_role	Yes	
binlog_checksum	Yes	
binlog_format	Yes	
binlog_row_image	No	
binlog_rows_query_log_events	No	
character-set-client-handshake	Yes	
character_set_client	Yes	
character_set_connection	Yes	
character_set_database	Yes	
character_set_filesystem	Yes	
character_set_results	Yes	
character_set_server	Yes	
collation_connection	Yes	
collation_server	Yes	

Parameter name	Modifiable	Notes
completion_type	Yes	
default_storage_engine	No	Aurora MySQL clusters use the InnoDB storage engine for all of your data.
innodb_autoinc_lock_mode	Yes	
innodb_checksums	No	
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 filesystem directly.
innodb_file_per_table	Yes	
innodb_flush_log_at_trx_commit	Yes	
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	
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 filesystem directly.
innodb_undo_logs	Yes	

Parameter name	Modifiable	Notes
innodb_undo_tablespaces	No	
lc_time_names	Yes	
lower_case_table_names	Yes	
master-info-repository	Yes	
master_verify_checksum	Yes	
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 Publishing Amazon Aurora MySQL Logs to Amazon CloudWatch Logs (p. 759) .
server_id	No	
skip-character-set-client-handshake	Yes	
skip_name_resolve	No	
sync_frm	Yes	
time_zone	Yes	

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
allow-suspicious-udfs	No	
aurora_lab_mode	Yes	For more information, see Amazon Aurora MySQL Lab Mode (p. 762) .
aurora_oom_response	Yes	For more information, see Amazon Aurora MySQL Out of Memory Issues (p. 1006) .
aurora_read_replica_read_committed	Yes	Enables READ COMMITTED isolation level for Aurora Replicas and changes the isolation behavior to reduce purge lag during long-running queries. Enable this setting only if you understand the behavior changes and how they affect your query results. For example, this setting uses less-strict isolation than the MySQL default. When it's enabled, long-running queries might see more than one

Parameter name	Modifiable	Notes
		copy of the same row because Aurora reorganizes the table data while the query is running. For more information, see ??? (p. 787).
<code>autocommit</code>	Yes	
<code>automatic_sp_privileges</code>	Yes	
<code>back_log</code>	Yes	
<code>basedir</code>	No	Aurora MySQL uses managed instances where you don't access the filesystem directly.
<code>binlog_cache_size</code>	Yes	
<code>binlog_max_flush_queue_time</code>	Yes	
<code>binlog_order_commits</code>	Yes	
<code>binlog_stmt_cache_size</code>	Yes	
<code>bulk_insert_buffer_size</code>	Yes	
<code>concurrent_insert</code>	Yes	
<code>connect_timeout</code>	Yes	
<code>core-file</code>	No	Aurora MySQL uses managed instances where you don't access the filesystem directly.
<code>datadir</code>	No	Aurora MySQL uses managed instances where you don't access the filesystem directly.
<code>default_time_zone</code>	No	
<code>default_tmp_storage_engine</code>	Yes	
<code>default_week_format</code>	Yes	
<code>delay_key_write</code>	Yes	
<code>delayed_insert_limit</code>	Yes	
<code>delayed_insert_timeout</code>	Yes	
<code>delayed_queue_size</code>	Yes	
<code>div_precision_increment</code>	Yes	
<code>end_markers_in_json</code>	Yes	
<code>enforce_gtid_consistency</code>	Sometimes	Modifiable in Aurora MySQL version 2.04 and later.
<code>eq_range_index_dive_limit</code>	Yes	

Parameter name	Modifiable	Notes
event_scheduler	Yes	
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 Publishing Amazon Aurora MySQL Logs to Amazon CloudWatch Logs (p. 759) .
general_log_file	No	Aurora MySQL uses managed instances where you don't access the filesystem directly.
group_concat_max_len	Yes	
gtid-mode	Sometimes	Modifiable in Aurora MySQL version 2.04 and later.
host_cache_size	Yes	
init_connect	Yes	
innodb_adaptive_hash_index	Yes	
innodb_adaptive_max_sleep_delay	Yes	
innodb_autoextend_increment	Yes	
innodb_buffer_pool_dump_at_shutdown	No	
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	
innodb_change_buffer_max_size	No	Aurora MySQL doesn't use the InnoDB change buffer at all.
innodb_compression_failure_threshold_pct	Yes	

Parameter name	Modifiable	Notes
innodb_compression_level	Yes	
innodb_compression_pad_pct_max	Yes	
innodb_concurrency_tickets	Yes	
innodb_file_format	Yes	
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	
innodb_lock_wait_timeout	Yes	
innodb_log_compressed_pages	No	
innodb_lru_scan_depth	Yes	
innodb_max_purge_lag	Yes	
innodb_max_purge_lag_delay	Yes	
innodb_monitor_disable	Yes	
innodb_monitor_enable	Yes	
innodb_monitor_reset	Yes	
innodb_monitor_reset_all	Yes	
innodb_old_blocks_pct	Yes	
innodb_old_blocks_time	Yes	
innodb_open_files	Yes	
innodb_print_all_deadlocks	Yes	
innodb_random_read_ahead	Yes	
innodb_read_ahead_threshold	Yes	
innodb_read_io_threads	No	

Parameter name	Modifiable	Notes
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	
interactive_timeout	Yes	
join_buffer_size	Yes	
keep_files_on_create	Yes	
key_buffer_size	Yes	
key_cache_age_threshold	Yes	
key_cache_block_size	Yes	
key_cache_division_limit	Yes	
local_infile	Yes	
lock_wait_timeout	Yes	
log-bin	No	
log_bin_trust_function_creators	Yes	
log_bin_use_v1_row_events	Yes	
log_error	No	
log_output	Yes	
log_queries_not_using_indexes	Yes	
log_slave_updates	No	
log_throttle_queries_not_using_indexes	Yes	

Parameter name	Modifiable	Notes
<code>log_warnings</code>	Yes	
<code>long_query_time</code>	Yes	
<code>low_priority_updates</code>	Yes	
<code>max_allowed_packet</code>	Yes	
<code>max_binlog_cache_size</code>	Yes	
<code>max_binlog_size</code>	No	
<code>max_binlog_stmt_cache_size</code>	Yes	
<code>max_connect_errors</code>	Yes	
<code>max_connections</code>	Yes	For more information, see Maximum Connections to an Aurora MySQL DB Instance (p. 624) .
<code>max_delayed_threads</code>	Yes	
<code>max_error_count</code>	Yes	
<code>max_heap_table_size</code>	Yes	
<code>max_insert_delayed_threads</code>	Yes	
<code>max_join_size</code>	Yes	
<code>max_length_for_sort_data</code>	Yes	
<code>max_prepared_stmt_count</code>	Yes	
<code>max_seeks_for_key</code>	Yes	
<code>max_sort_length</code>	Yes	
<code>max_sp_recursion_depth</code>	Yes	
<code>max_tmp_tables</code>	Yes	
<code>max_user_connections</code>	Yes	
<code>max_write_lock_count</code>	Yes	
<code>metadata_locks_cache_size</code>	Yes	
<code>min_examined_row_limit</code>	Yes	
<code>myisam_data_pointer_size</code>	Yes	
<code>myisam_max_sort_file_size</code>	Yes	
<code>myisam mmap_size</code>	Yes	
<code>myisam_sort_buffer_size</code>	Yes	
<code>myisam_stats_method</code>	Yes	
<code>myisam_use mmap</code>	Yes	

Parameter name	Modifiable	Notes
net_buffer_length	Yes	
net_read_timeout	Yes	
net_retry_count	Yes	
net_write_timeout	Yes	
old-style-user-limits	Yes	
old_passwords	Yes	
optimizer_prune_level	Yes	
optimizer_search_depth	Yes	
optimizer_switch	Yes	
optimizer_trace	Yes	
optimizer_trace_features	Yes	
optimizer_trace_limit	Yes	
optimizer_trace_max_mem_size	Yes	
optimizer_trace_offset	Yes	
performance_schema	Yes	
pid_file	No	
plugin_dir	No	Aurora MySQL uses managed instances where you don't access the filesystem 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	
query_cache_min_res_unit	Yes	
query_cache_size	Yes	
query_cache_type	Yes	
query_cache_wlock_invalidate	Yes	
query_prealloc_size	Yes	
range_alloc_block_size	Yes	
read_buffer_size	Yes	

Parameter name	Modifiable	Notes
<code>read_only</code>	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).
<code>read_rnd_buffer_size</code>	Yes	
<code>relay-log</code>	No	
<code>relay_log_info_repository</code>	Yes	
<code>relay_log_recovery</code>	No	
<code>safe-user-create</code>	Yes	
<code>secure_auth</code>	Yes	
<code>secure_file_priv</code>	No	Aurora MySQL uses managed instances where you don't access the filesystem 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	
<code>slave_checkpoint_period</code>	Yes	
<code>slave_parallel_workers</code>	Yes	
<code>slave_pending_jobs_size_max</code>	Yes	
<code>slave_sql_verify_checksum</code>	Yes	
<code>slow_launch_time</code>	Yes	
<code>slow_query_log</code>	Yes	For instructions on uploading the logs to CloudWatch Logs, see Publishing Amazon Aurora MySQL Logs to Amazon CloudWatch Logs (p. 759) .
<code>slow_query_log_file</code>	No	Aurora MySQL uses managed instances where you don't access the filesystem directly.
<code>socket</code>	No	
<code>sort_buffer_size</code>	Yes	
<code>sql_mode</code>	Yes	
<code>sql_select_limit</code>	Yes	
<code>stored_program_cache</code>	Yes	

Parameter name	Modifiable	Notes
sync_binlog	No	
sync_master_info	Yes	
sync_relay_log	Yes	
sync_relay_log_info	Yes	
sysdate-is-now	Yes	
table_cache_element_entry_ttl	No	
table_definition_cache	Yes	
table_open_cache	Yes	
table_open_cache_instances	Yes	
temp_pool	Yes	
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 filesystem directly.
transaction_alloc_block_size	Yes	
transaction_prealloc_size	Yes	
tx_isolation	Yes	
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	
wait_timeout	Yes	

Inapplicable MySQL Parameters and Status Variables

Because of architectural differences between Aurora MySQL and MySQL, some MySQL parameters and status variables don't apply to Aurora MySQL.

The following MySQL parameters don't apply to Aurora MySQL:

- `innodb_adaptive_flushing`
- `innodb_adaptive_flushing_lwm`
- `innodb_change_buffering`
- `innodb_checksum_algorithm`
- `innodb_doublewrite`
- `innodb_flush_method`
- `innodb_flush_neighbors`
- `innodb_io_capacity`
- `innodb_io_capacity_max`
- `innodb_log_buffer_size`
- `innodb_log_file_size`
- `innodb_log_files_in_group`
- `innodb_max_dirty_pages_pct`
- `innodb_use_native_aio`
- `innodb_write_io_threads`
- `thread_cache_size`

The following MySQL status variables don't apply to Aurora MySQL:

- `innodb_buffer_pool_bytes_dirty`
- `innodb_buffer_pool_pages_dirty`
- `innodb_buffer_pool_pages_flushed`

Note

These lists are not exhaustive.

Aurora MySQL 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.

`io/aurora_redo_log_flush`

In this wait event, a session is writing data to Aurora storage. Typically, this wait event is for a write I/O operation in Aurora MySQL.

`io/aurora_respond_to_client`

In this wait event, a thread is in the process of returning the result set to the client.

`io/file/csv/data`

In this wait event, there are threads 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

In this wait event, there are threads waiting on I/O operations to storage. This event is more prevalent in I/O intensive workloads. SQL statements showing a comparatively large proportion of this wait event might be running disk intensive queries. Or they might be requesting data that can't be satisfied from the InnoDB buffer pool. To find out, check your query plans and cache hit ratios.

For more information, see [Buffering and Caching](#) in the MySQL documentation.

io/file/sql/binlog

In this wait event, there is a thread waiting on a binlog file that is being written to disk.

io/socket/sql/client_connection

In this wait event, a thread is in the process of handling a new connection.

io/table/sql/handler

This is a table I/O wait event. Typically, these types of events can be followed by a nested event such as a file I/O event. For more information about 'atom' and 'molecule' events in the Performance Schema, see [Performance Schema Atom and Molecule Events](#) in the MySQL documentation.

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

In this wait event, threads are suspended when they are waiting on a condition. For example, this event occurs when threads are waiting for table level lock. We recommend that you investigate your workload to see what threads might be acquiring table locks on your DB instance. For more information about table locking in MySQL, see [Table Locking Issues](#) in the MySQL documentation.

synch/cond/sql/MDL_context::COND_wait_status

In this wait event, there are threads waiting on a table metadata lock. For more information, see [Optimizing Locking Operations](#) in the MySQL documentation.

synch/cond/sql/MYSQL_BIN_LOG::COND_done

In this wait event, there is a thread waiting on a binlog file that is being written to disk. Binary logging contention can occur on databases with a very high change rate.

synch/mutex/innodb/aurora_lock_thread_slot_futex

In this wait event, there is a thread that is waiting on an InnoDB record lock. If you see this event, check your database for conflicting workloads. For more information, see [InnoDB Locking](#) in the MySQL documentation.

synch/mutex/innodb/buf_pool_mutex

In this wait event, a thread has acquired a lock on the InnoDB buffer pool.

synch/mutex/sql/LOCK_open

In this wait event, LOCK_open is being used to protect various objects in the data dictionary. This wait event indicates that there are threads waiting to acquire these locks. Typically, this event is caused by data dictionary contention.

synch/mutex/sql/LOCK_table_cache

In this wait event, there are threads waiting to acquire a lock on a table cache instance. For more information, see [How MySQL Opens and Closes Tables](#) in the MySQL documentation.

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 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.

synch/mutex/sql/MYSQL_BIN_LOG::LOCK_log

In this wait event, threads are actively locking the binary log file. 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.

synch/rwlock/innodb/dict

In this wait event, there are threads waiting on an rwlock held on the InnoDB data dictionary.

synch/rwlock/innodb/dict sys RW lock

In this event, there are threads that are waiting on an rwlock held on the InnoDB data dictionary.

synch/rwlock/innodb/dict_operation_lock

In this wait event, there are threads holding locks on InnoDB data dictionary operations.

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, or any DB instance in an Aurora MySQL multi-master cluster. These isolation levels work the same in Aurora MySQL as in RDS 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 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 Amazon RDS 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.

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 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.

You can enable this feature permanently for one or more Aurora Replicas in your cluster. To do so, turn on the `aurora_read_replica_read_committed` setting in the DB parameter group used by the associated DB instances. Enable this setting on the DB instances where you run the long-running queries. In this case, connect to the instance endpoints to ensure the queries run on the intended DB instances.

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 READ COMMITTED	Query on Aurora Replica with READ COMMITTED
T1	<code>INSERT INTO big_table SELECT * FROM other_table LIMIT 1000000; COMMIT;</code>		
T2		Q1: <code>SELECT COUNT(*) FROM big_table;</code>	Q2: <code>SELECT COUNT(*) FROM big_table;</code>
T3	<code>INSERT INTO big_table (c1, c2) VALUES (1, 'one more row'); COMMIT;</code>		
T4		If Q1 finishes now, result is 1,000,000.	If Q2 finishes now, result is 1,000,000 or 1,000,001.

Time	DML Statement on Aurora Primary Instance	Query on Aurora Primary Instance with READ COMMITTED	Query on Aurora Replica with READ COMMITTED
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 = CONCAT(c2,c2,c2); COMMIT;</code>		
T8		If Q1 finishes now, result is 1,000,000.	If Q2 finishes now, result is 1,000,000 or 1,000,001 or 999,999, or possibly some higher number.
T9		Q3: <code>SELECT COUNT(*) FROM big_table;</code>	Q4: <code>SELECT COUNT(*) FROM big_table;</code>
T10		If Q3 finishes now, result is 999,999.	If Q4 finishes now, result is 999,999.
T11		Q5: <code>SELECT COUNT(*) FROM parent_table p JOIN child_table c ON (p.id = c.id) WHERE p.id = 1000;</code>	Q6: <code>SELECT COUNT(*) FROM parent_table p JOIN child_table c ON (p.id = c.id) WHERE p.id = 1000;</code>
T12	<code>INSERT INTO parent_table (id, s) VALUES (1000, 'hello'); INSERT INTO child_table (id, s) VALUES (1000, 'world'); COMMIT;</code>		
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 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 Stored Procedures

You can call the following stored procedures while connected to the primary instance in an Aurora MySQL cluster. These procedures control how transactions are replicated from an external database into Aurora MySQL, or from Aurora MySQL to an external database. To learn how to use replication based on global transaction identifiers (GTIDs) with Aurora MySQL, see [Using GTID-Based Replication for Aurora MySQL \(p. 698\)](#).

Topics

- [mysql.rds_set_master_auto_position \(p. 791\)](#)
- [mysql.rds_set_external_master_with_auto_position \(p. 792\)](#)
- [mysql.rds_skip_transaction_with_gtid \(p. 793\)](#)

[mysql.rds_set_master_auto_position](#)

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_external_master_with_auto_position](#)

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 Amazon RDS 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 Amazon RDS 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.

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 \(p. 793\)](#) stored procedure. For more information about working with GTID-based replication, see [Using GTID-Based Replication for Aurora MySQL \(p. 698\)](#).

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_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. GTID-based replication isn't supported for Aurora MySQL 1.1 or 1.0.

Database Engine Updates for Amazon Aurora MySQL

Amazon Aurora releases updates regularly. Updates are applied to Aurora DB clusters during system maintenance windows. The timing when updates are applied depends on the region and maintenance window setting for the DB cluster, as well as the type of update.

Updates are applied to all instances in a DB cluster at the same time. An update requires a database restart on all instances in a DB cluster, so you experience 20 to 30 seconds of downtime, after which you can resume using your DB cluster or clusters. You can view or change your maintenance window settings from the [AWS Management Console](#).

Aurora MySQL Versions

Although Aurora with MySQL compatibility is compatible with the MySQL database engines, Aurora MySQL includes features that are specific to Aurora MySQL and only available to Aurora MySQL DB clusters.

Aurora versions use the format `<major version>.<minor version>. <patch version>`. You can get the version of your Aurora instance by querying for the `AURORA_VERSION` system variable. To get the Aurora version, use one of the following queries.

```
select AURORA_VERSION();
select @@aurora_version;
```

Aurora MySQL Engine Versions

Starting with Aurora MySQL 2.03.2, Aurora engine versions have the following syntax.

```
<mysql-major-version>.mysql_aurora.<aurora-mysql-version>
```

For example, the engine version for Aurora MySQL 2.03.2 is the following.

```
5.7.mysql_aurora.2.03.2
```

Note

For Aurora MySQL 2.x, the engine version for Aurora MySQL version 2.03.1 and lower is 5.7.12. Currently, the engine version for all Aurora MySQL 1.x versions is 5.6.10a.

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](#).

Prior to Aurora MySQL 2.03.2, the engine version displayed by the AWS Management Console remained the same even after you upgraded the engine in the cluster. In Aurora MySQL 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. 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](#).

Database Upgrades and Patches for Amazon Aurora MySQL

There are two ways to upgrade the minor version of a DB cluster or patch a DB cluster:

- [Modifying the Engine Version \(p. 795\)](#) (upgrading to version 2.03.2 and higher only)
- [Applying Pending Maintenance to an Aurora MySQL DB Cluster \(p. 796\)](#)

Modifying the Engine Version

When upgrading to Amazon Aurora MySQL version 2.03.2 and higher, you upgrade the minor version of a DB cluster. You do so by modifying the engine version of the DB cluster using the AWS Management Console, AWS CLI, or the RDS API.

To modify the engine version of a DB cluster

- **By using the Amazon RDS console** – Complete the following steps:
 1. Sign in to the Amazon RDS console and choose **Databases**.
 2. Choose the DB cluster that you want to modify.
 3. Choose **Modify**.
 4. Change the Aurora MySQL engine version in the **DB engine version** box.
 5. Choose **Continue** and check the summary of modifications.
 6. (Optional) Choose **Apply immediately** to apply the changes immediately.
 7. On the confirmation page, choose **Modify cluster**.
- **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 Amazon 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.

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 Amazon RDS console** – Complete the following steps:
 1. Sign in to the Amazon RDS console and choose **Databases**.
 2. Choose the DB cluster that shows an **available** maintenance upgrade.
 3. 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 Amazon 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 \(p. 408\)](#).

Note

If your current Aurora MySQL version is 1.14.x, but it is 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.

Zero-Downtime Patching

The zero-downtime patching (ZDP) feature attempts, on a **best-effort** basis, to preserve client connections through an engine patch. If ZDP executes successfully, application sessions are preserved and the database engine restarts while patching. The database engine restart can cause a drop in throughput lasting approximately 5 seconds. ZDP is available in Aurora MySQL 1.13 and later, compatible with MySQL 5.6. It's also available in Aurora MySQL 2.07 and later, compatible with MySQL 5.7.

ZDP might not execute 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 cancelled.
- Binary logging is enabled or binary log replication is in-progress.

- Open SSL connections exist.
- Temporary tables or table locks are in use, for example during DDL statements. If Aurora can perform ZDP in this case, any open transactions are cancelled.
- Pending parameter changes exist.

Starting in Aurora MySQL 1.19 and 2.07, the ZDP mechanism is improved. These improvements make ZDP more likely to succeed when there are long-running transactions, binary logging, table locks, or temporary tables.

If no suitable time window for executing ZDP becomes available because of one or more of these conditions, patching reverts to the standard behavior.

Note

- ZDP applies only to the primary instance of a DB cluster. ZDP isn't applicable to Aurora Replicas.
- Prepared statements don't prevent ZDP, but they aren't preserved after ZDP executes.

Aurora MySQL Long-Term Support (LTS) Releases

Each new Aurora MySQL version remains available for a certain amount of time for you to use when you create or upgrade a DB cluster. After this period, you must upgrade any clusters that use that version. You can manually upgrade your cluster before the support period ends, or Aurora can automatically upgrade it for you when its Aurora MySQL version is no longer supported.

Aurora designates certain Aurora MySQL versions as "long-term support" (LTS) releases. DB clusters that use LTS releases can stay on the same version longer and undergo fewer upgrade cycles than clusters that use non-LTS releases. Aurora supports each LTS release for at least one year after that release becomes available. When a DB cluster that's on an LTS release is required to upgrade, Aurora upgrades it to the next LTS release. That way, the cluster doesn't need to be upgraded again for a long time.

During the lifetime of an Aurora MySQL LTS release, new patch levels introduce fixes to important issues. The patch levels don't include any new features. You can choose whether or not 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 that have the following requirements and business needs:

- 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.

As of the publication date for this *Aurora User's Guide*, the current LTS releases for Aurora MySQL are:

- Aurora MySQL version 1.19.5. For more details about this version, see [Aurora MySQL Database Engine Updates 2019-09-19 \(Version 1.19.5\) \(p. 836\)](#).
- Aurora MySQL version 2.04.6. For more details about this version, see [Aurora MySQL Database Engine Updates 2019-09-19 \(Version 2.04.6\) \(p. 808\)](#).

Related Topics

- [Database Engine Updates for Amazon Aurora MySQL 2.0 \(p. 798\)](#)
- [Database Engine Updates for Amazon Aurora MySQL 1.1 \(p. 830\)](#)
- [MySQL Bugs Fixed by Aurora MySQL Database Engine Updates \(p. 868\)](#)
- [Aurora Lab Mode Features \(p. 762\)](#)

Database Engine Updates for Amazon Aurora MySQL 2.0

The following are Amazon Aurora 2.0 database engine updates:

- [Aurora MySQL Database Engine Updates 2019-11-25 \(Version 2.07.0\) \(p. 798\)](#)
- [Aurora MySQL Database Engine Updates 2019-11-22 \(Version 2.06.0\) \(p. 801\)](#)
- [Aurora MySQL Database Engine Updates 2019-11-11 \(Version 2.05.0\) \(p. 803\)](#)
- [Aurora MySQL Database Engine Updates 2019-11-20 \(Version 2.04.8\) \(p. 805\)](#)
- [Aurora MySQL Database Engine Updates 2019-11-14 \(Version 2.04.7\) \(p. 807\)](#)
- [Aurora MySQL Database Engine Updates 2019-09-19 \(Version 2.04.6\) \(p. 808\)](#)
- [Aurora MySQL Database Engine Updates 2019-07-08 \(Version 2.04.5\) \(p. 810\)](#)
- [Aurora MySQL Database Engine Updates 2019-05-29 \(Version 2.04.4\) \(p. 811\)](#)
- [Aurora MySQL Database Engine Updates 2019-05-09 \(Version 2.04.3\) \(p. 813\)](#)
- [Aurora MySQL Database Engine Updates 2019-05-02 \(Version 2.04.2\) \(p. 814\)](#)
- [Aurora MySQL Database Engine Updates 2019-03-25 \(Version 2.04.1\) \(p. 815\)](#)
- [Aurora MySQL Database Engine Updates 2019-03-25 \(Version 2.04\) \(p. 816\)](#)
- [Aurora MySQL Database Engine Updates 2019-02-07 \(p. 817\) \(Version 2.03.4\)](#)
- [Aurora MySQL Database Engine Updates 2019-01-18 \(p. 817\) \(Version 2.03.3\)](#)
- [Aurora MySQL Database Engine Updates 2019-01-09 \(p. 818\) \(Version 2.03.2\)](#)
- [Aurora MySQL Database Engine Updates 2018-10-24 \(p. 819\) \(Version 2.03.1\)](#)
- [Aurora MySQL Database Engine Updates 2018-10-11 \(p. 819\) \(Version 2.03\)](#)
- [Aurora MySQL Database Engine Updates 2018-10-08 \(p. 820\) \(Version 2.02.5\)](#)
- [Aurora MySQL Database Engine Updates 2018-09-21 \(p. 821\) \(Version 2.02.4\)](#)
- [Aurora MySQL Database Engine Updates 2018-08-23 \(p. 821\) \(Version 2.02.3\)](#)
- [Aurora MySQL Database Engine Updates 2018-06-04 \(p. 823\) \(Version 2.02.2\)](#)
- [Aurora MySQL Database Engine Updates 2018-05-03 \(p. 825\) \(Version 2.02\)](#)
- [Aurora MySQL Database Engine Updates 2018-03-13 \(p. 827\) \(Version 2.01.1\)](#)
- [Aurora MySQL Database Engine Updates 2018-02-06 \(p. 828\) \(Version 2.01\)](#)

Aurora MySQL Database Engine Updates 2019-11-25 (Version 2.07.0)

Version: 2.07.0

Aurora MySQL 2.07.0 is generally available. Aurora MySQL 2.x versions are compatible with MySQL 5.7 and Aurora MySQL 1.x versions are compatible with MySQL 5.6.

Currently supported Aurora MySQL releases are 1.14.* , 1.15.* , 1.16.* , 1.17.* , 1.18.* , 1.19.* , 1.20.* , 1.21.* , 1.22.* , 2.01.* , 2.02.* , 2.03.* , 2.04.* , 2.05.* , 2.06.* , and 2.07.*.

You can restore a snapshot from a currently supported Aurora MySQL release into Aurora MySQL 2.07.0. You also have the option to upgrade existing Aurora MySQL 2.* database clusters to Aurora MySQL 2.07.0. You cannot upgrade an existing Aurora MySQL 1.* cluster directly to 2.07.0; however, you can restore its snapshot to Aurora MySQL 2.07.0.

To create a cluster with an older version of Aurora MySQL, please specify the engine version through the AWS Management Console, the AWS CLI, or the RDS API.

Note

This version is currently not available in the following AWS Regions: AWS GovCloud (US-East) [us-gov-east-1], AWS GovCloud (US-West) [us-gov-west-1], China (Ningxia) [cn-northwest-1], Asia Pacific (Hong Kong) [ap-east-1], Middle East (Bahrain) [me-south-1], and South America (São Paulo) [sa-east-1]. There will be a separate announcement once it is made available.

If you have any questions or concerns, AWS Support is available on the community forums and through [AWS Premium Support](#). For more information, see [Maintaining an Amazon Aurora DB Cluster \(p. 408\)](#).

Note

For information on how to upgrade your Aurora MySQL database cluster, see [Database Upgrades and Patches for Amazon Aurora MySQL \(p. 795\)](#).

Improvements

New features:

- Global Databases now allow adding secondary read-only replica regions for database clusters deployed in these AWS Regions: regions: US East (N. Virginia) [us-east-1], US East (Ohio) [us-east-2], US West (N. California) [us-west-1], US West (Oregon) [us-west-2], Europe (Ireland) [eu-west-1], Europe (London) [eu-west-2], Europe (Paris) [eu-west-3], Asia Pacific (Tokyo) [ap-northeast-1], Asia Pacific (Seoul) [ap-northeast-2], Asia Pacific (Singapore) [ap-southeast-1], Asia Pacific (Sydney) [ap-southeast-2], Canada (Central) [ca-central-1], Europe (Frankfurt) [eu-central-1], and Asia Pacific (Mumbai) [ap-south-1].
- Amazon Aurora machine learning is a highly optimized integration between the Aurora MySQL database and AWS machine learning (ML) services. Aurora machine learning allows developers to add a variety of ML-based predictions to their database applications by invoking ML models using the familiar SQL programming language they already use for database development, without having to build custom integrations or learn separate tools. For more information, see [Using Machine Learning \(ML\) Capabilities with Amazon Aurora](#).
- Added support for the ANSI READ COMMITTED isolation level on the read replicas. This isolation level enables long-running queries on the read replica to execute without impacting the high throughput of writes on the writer node. For more information, see [Aurora MySQL Isolation Levels](#).

Critical fixes:

- [CVE-2019-2922](#)
- [CVE-2019-2923](#)
- [CVE-2019-2924](#)
- [CVE-2019-2910](#)

High-priority fixes:

- Fixed an issue in the DDL recovery that resulted in prolonged database downtime. Clusters that become unavailable after executing multi-table drop statement, for example `DROP TABLE t1, t2, t3`, should be updated to this version.

- Fixed an issue in the DDL recovery that resulted in prolonged database downtime. Clusters that become unavailable after executing `INPLACE ALTER TABLE` DDL statements should be updated to this version.

General stability fixes:

- Fixed an issue that generated inconsistent data in the `information_schema.replica_host_status` table.

Integration of MySQL community edition bug fixes

- Bug #26251621: INCORRECT BEHAVIOR WITH TRIGGER AND GCOL
- Bug #22574695: ASSERTION `!TABLE || (!TABLE->READ_SET || BITMAP_IS_SET(TABLE->READ_SET, FIEL
- Bug #25966845: INSERT ON DUPLICATE KEY GENERATE A DEADLOCK
- Bug #23070734: CONCURRENT TRUNCATE TABLES CAUSE STALL
- Bug #26191879: FOREIGN KEY CASCades USE EXCESSIVE MEMORY
- Bug #20989615: INNODB AUTO_INCREMENT PRODUCES SAME VALUE TWICE

Comparison with Aurora MySQL Version 1

The following Amazon Aurora MySQL features are supported in Aurora MySQL Version 1 (compatible with MySQL 5.6), but these features are currently not supported in Aurora MySQL Version 2 (compatible with MySQL 5.7).

- Asynchronous key prefetch (AKP). For more information, see [Working with Asynchronous Key Prefetch in Amazon Aurora \(p. 765\)](#).
- Scan batching. For more information, see [Aurora MySQL Database Engine Updates 2017-12-11 \(p. 846\)](#).
- Migrating data from MySQL using an Amazon S3 bucket. For more information, see [Migrating Data from MySQL by Using an Amazon S3 Bucket \(p. 590\)](#).

MySQL 5.7 compatibility

Aurora MySQL 2.07.0 is wire-compatible with MySQL 5.7 and includes features such as JSON support, spatial indexes, and generated columns. Aurora MySQL uses a native implementation of spatial indexing using z-order curves to deliver >20x better write performance and >10x better read performance than MySQL 5.7 for spatial datasets.

Aurora MySQL 2.07.0 does not currently support the following MySQL 5.7 features:

- Group replication plugin
- Increased page size
- InnoDB buffer pool loading at startup
- InnoDB full-text parser plugin
- Multisource replication
- Online buffer pool resizing
- Password validation plugin
- Query rewrite plugins
- Replication filtering

- The `CREATE TABLESPACE` SQL statement

Aurora MySQL Database Engine Updates 2019-11-22 (Version 2.06.0)

Version: 2.06.0

Aurora MySQL 2.06.0 is generally available. Aurora MySQL 2.x versions are compatible with MySQL 5.7 and Aurora MySQL 1.x versions are compatible with MySQL 5.6.

Currently supported Aurora MySQL releases are 1.14.*, 1.15.*, 1.16.*, 1.17.*, 1.18.*, 1.19.*, 2.01.*, 2.02.*, 2.03.*, 2.04.*, 2.05.*, and 2.06.*.

You can restore a snapshot from a currently supported Aurora MySQL release into Aurora MySQL 2.06.0. You also have the option to upgrade existing Aurora MySQL 2.* database clusters to Aurora MySQL 2.06.0. You cannot upgrade an existing Aurora MySQL 1.* cluster directly to 2.06.0; however, you can restore its snapshot to Aurora MySQL 2.06.0.

To create a cluster with an older version of Aurora MySQL, please specify the engine version through the AWS Management Console, the AWS CLI, or the RDS API.

Note

This version is currently not available in the following AWS Regions: AWS GovCloud (US-East) [us-gov-east-1], AWS GovCloud (US-West) [us-gov-west-1], China (Ningxia) [cn-northwest-1], Asia Pacific (Hong Kong) [ap-east-1], and Middle East (Bahrain) [me-south-1]. There will be a separate announcement once it is made available.

If you have any questions or concerns, AWS Support is available on the community forums and through [AWS Premium Support](#). For more information, see [Maintaining an Amazon Aurora DB Cluster \(p. 408\)](#).

Note

For information on how to upgrade your Aurora MySQL database cluster, see [Database Upgrades and Patches for Amazon Aurora MySQL \(p. 795\)](#).

Improvements

New features:

- Aurora MySQL clusters now support the instance types db.r5.8xlarge, db.r5.16xlarge, and db.r5.24xlarge. For more information about instance types for Aurora MySQL clusters, see [Choosing the DB Instance Class \(p. 76\)](#).
- The hash join feature is now generally available and does not require the Aurora lab mode setting to be ON. This feature can improve query performance when you need to join a large amount of data by using an equi-join. For more information about using this feature, see [Working with Hash Joins in Aurora MySQL \(p. 771\)](#).
- The hot row contention feature is now generally available and does not require the Aurora lab mode setting to be ON. This feature substantially improves throughput for workloads with many transactions contending for rows on the same page.
- Aurora MySQL 2.06 and higher support "rewinding" a DB cluster to a specific time, without restoring data from a backup. This feature, known as Backtrack, provides a quick way to recover from user errors, such as dropping the wrong table or deleting the wrong row. Backtrack completes within seconds, even for large databases. Read the [AWS blog](#) for an overview, and refer to [Backtracking an Aurora DB Cluster \(p. 625\)](#) for more details.

Critical fixes:

None.

High-priority fixes:

CVE fixes

- [CVE-2019-2805](#)
- [CVE-2019-2730](#)
- [CVE-2019-2739](#)
- [CVE-2019-2778](#)
- [CVE-2019-2758](#)
- [CVE-2018-3064](#)
- [CVE-2018-3058](#)
- [CVE-2018-2786](#)
- [CVE-2017-3653](#)
- [CVE-2017-3455](#)
- [CVE-2017-3465](#)
- [CVE-2017-3244](#)
- [CVE-2016-5612](#)

Connection handling

- Database availability has been improved to better service a surge in client connections while executing one or more DDLs. It is handled by temporarily creating additional threads when needed. You are advised to upgrade if the database becomes unresponsive following a surge in connections while processing DDL.

Engine restart

- Fixed an issue of prolonged unavailability while restarting the engine. This addresses an issue in the buffer pool initialization. This issue occurs rarely but can potentially impact any supported release.
- Fixed an issue that causes a database configured as a binlog master to restart while a heavy write workload is running.

General stability fixes:

- Made improvements where queries accessing uncached data could be slower than usual. Customers experiencing unexplained elevated read latency while accessing uncached data are encouraged to upgrade as they may be experiencing this issue.
- Fixed an issue that failed to restore partitioned tables from a database snapshot. Customers who encounter errors when accessing partitioned tables in a database that has been restored from the snapshot of an Aurora MySQL 1.* database are advised to use this version.
- Improved stability of the Aurora Replicas by fixing lock contention between threads serving read queries and the one applying schema changes while a DDL query is in progress on the writer DB instance.
- Fixed a stability issue related to `mysql.innodb_table_stats` table update triggered by DDL operations.
- Fixed an issue that incorrectly reported `ERROR 1836` when a nested query is executed against a temporary table on the Aurora Replica.

Performance enhancements:

- Improved performance of binlog replication by preventing unnecessary API calls to the cache if the query cache has been disabled on the binlog slave.

Comparison with Aurora MySQL Version 1

The following Amazon Aurora MySQL features are supported in Aurora MySQL Version 1 (compatible with MySQL 5.6), but these features are currently not supported in Aurora MySQL Version 2 (compatible with MySQL 5.7).

- Asynchronous key prefetch (AKP). For more information, see [Working with Asynchronous Key Prefetch in Amazon Aurora \(p. 765\)](#).
- Scan batching. For more information, see [Aurora MySQL Database Engine Updates 2017-12-11 \(p. 846\)](#).
- Migrating data from MySQL using an Amazon S3 bucket. For more information, see [Migrating Data from MySQL by Using an Amazon S3 Bucket \(p. 590\)](#).

MySQL 5.7 compatibility

Aurora MySQL 2.06.0 is wire-compatible with MySQL 5.7 and includes features such as JSON support, spatial indexes, and generated columns. Aurora MySQL uses a native implementation of spatial indexing using z-order curves to deliver >20x better write performance and >10x better read performance than MySQL 5.7 for spatial datasets.

Aurora MySQL 2.06.0 does not currently support the following MySQL 5.7 features:

- Group replication plugin
- Increased page size
- InnoDB buffer pool loading at startup
- InnoDB full-text parser plugin
- Multisource replication
- Online buffer pool resizing
- Password validation plugin
- Query rewrite plugins
- Replication filtering
- The `CREATE TABLESPACE` SQL statement

Aurora MySQL Database Engine Updates 2019-11-11 (Version 2.05.0)

Version: 2.05.0

Aurora MySQL 2.05.0 is generally available. Aurora MySQL 2.x versions are compatible with MySQL 5.7 and Aurora MySQL 1.x versions are compatible with MySQL 5.6.

Currently supported Aurora MySQL releases are 1.14.*, 1.15.* , 1.16.* , 1.17.* , 1.18.* , 1.19.* , 2.01.* , 2.02.* , 2.03.* and 2.04.*.

You can restore a snapshot from a currently supported Aurora MySQL release into Aurora MySQL 2.05.0. You also have the option to upgrade existing Aurora MySQL 2.* database clusters to Aurora MySQL 2.05.0. You cannot upgrade an existing Aurora MySQL 1.* cluster directly to 2.05.0; however, you can restore its snapshot to Aurora MySQL 2.05.0.

To create a cluster with an older version of Aurora MySQL, please specify the engine version through the AWS Management Console, the AWS CLI, or the RDS API.

Note

This version is currently not available in the following AWS Regions: AWS GovCloud (US-East) [us-gov-east-1], AWS GovCloud (US-West) [us-gov-west-1], China (Ningxia) [cn-northwest-1], Asia Pacific (Hong Kong) [ap-east-1], Europe (Stockholm) [eu-north-1], and Middle East (Bahrain) [me-south-1]. There will be a separate announcement once it is made available.

If you have any questions or concerns, AWS Support is available on the community forums and through [AWS Premium Support](#). For more information, see [Maintaining an Amazon Aurora DB Cluster \(p. 408\)](#).

Note

For information on how to upgrade your Aurora MySQL database cluster, see [Database Upgrades and Patches for Amazon Aurora MySQL \(p. 795\)](#).

Improvements

Critical fixes:

- [CVE-2018-0734](#)
- [CVE-2019-2534](#)
- [CVE-2018-3155](#)
- [CVE-2018-2612](#)
- [CVE-2017-3599](#)
- [CVE-2018-3056](#)
- [CVE-2018-2562](#)
- [CVE-2017-3329](#)
- [CVE-2018-2696](#)
- Fixed an issue where the events in current binlog file on the master were not replicated on the slave if the value of the parameter `sync_binlog` was not set to 1.

High-priority fixes:

- Customers with database size close to 64 TiB are strongly advised to upgrade to this version to avoid downtime due to stability bugs affecting volumes close to the Aurora storage limit.
- The default value of the parameter `aurora_binlog_replication_max_yield_seconds` has been changed to zero to prevent an increase in replication lag in favor of foreground query performance on the binlog master.

Integration of MySQL Bug Fixes

- Bug#23054591: PURGE BINARY LOGS TO is reading the whole binlog file and causing MySQL to Stall

Comparison with Aurora MySQL Version 1

The following Amazon Aurora MySQL features are supported in Aurora MySQL Version 1 (compatible with MySQL 5.6), but these features are currently not supported in Aurora MySQL Version 2 (compatible with MySQL 5.7).

- Asynchronous key prefetch (AKP). For more information, see [Working with Asynchronous Key Prefetch in Amazon Aurora \(p. 765\)](#).
- Hash joins. For more information, see [Working with Hash Joins in Aurora MySQL \(p. 771\)](#).

- Native functions for synchronously invoking AWS Lambda functions. For more information, see [Invoking a Lambda Function with an Aurora MySQL Native Function \(p. 753\)](#).
- Scan batching. For more information, see [Aurora MySQL Database Engine Updates 2017-12-11 \(p. 846\)](#).
- Migrating data from MySQL using an Amazon S3 bucket. For more information, see [Migrating Data from MySQL by Using an Amazon S3 Bucket \(p. 590\)](#).

MySQL 5.7 compatibility

Aurora MySQL 2.05.0 is wire-compatible with MySQL 5.7 and includes features such as JSON support, spatial indexes, and generated columns. Aurora MySQL uses a native implementation of spatial indexing using z-order curves to deliver >20x better write performance and >10x better read performance than MySQL 5.7 for spatial datasets.

Aurora MySQL 2.05.0 does not currently support the following MySQL 5.7 features:

- Group replication plugin
- Increased page size
- InnoDB buffer pool loading at startup
- InnoDB full-text parser plugin
- Multisource replication
- Online buffer pool resizing
- Password validation plugin
- Query rewrite plugins
- Replication filtering
- The `CREATE TABLESPACE` SQL statement

Aurora MySQL Database Engine Updates 2019-11-20 (Version 2.04.8)

Version: 2.04.8

Aurora MySQL 2.04.8 is generally available. Aurora MySQL 2.x versions are compatible with MySQL 5.7 and Aurora MySQL 1.x versions are compatible with MySQL 5.6.

Currently supported Aurora MySQL releases are 1.14.*, 1.15.*, 1.16.*, 1.17.*, 1.18.*, 1.19.*, 2.01.*, 2.02.*, 2.03.*, 2.04.*, and 2.05.*. You can restore a snapshot of any 2.* Aurora MySQL release into Aurora MySQL 2.04.8. You also have the option to upgrade existing Aurora MySQL 2.* database clusters to Aurora MySQL 2.04.8.

To create a cluster with an older version of Aurora MySQL, please specify the engine version through the AWS Management Console, the AWS CLI, or the RDS API.

If you have any questions or concerns, AWS Support is available on the community forums and through [AWS Premium Support](#). For more information, see [Maintaining an Amazon Aurora DB Cluster \(p. 408\)](#).

Note

This version is currently not available in the following AWS Regions: AWS GovCloud (US-East) [us-gov-east-1], AWS GovCloud (US-West) [us-gov-west-1], Asia Pacific (Hong Kong) [ap-east-1], and Middle East (Bahrain) [me-south-1]. There will be a separate announcement once it is made available.

Note

For information on how to upgrade your Aurora MySQL database cluster, see [Database Upgrades and Patches for Amazon Aurora MySQL \(p. 795\)](#).

Improvements

New features:

- **Read Replica improvements:**

- Reduced network traffic from the writer instance by efficiently transmitting data to reader instances within the Aurora DB cluster. This improvement is enabled by default, because it helps prevent replicas from falling behind and restarting. The parameter for this feature is `aurora_enable_repl_bin_log_filtering`.
- Reduced network traffic from the writer to reader instances within the Aurora DB cluster using compression. This improvement is enabled by default for 8xlarge and 16xlarge instance classes only, because these instances can tolerate additional CPU overhead for compression. The parameter for this feature is `aurora_enable_replica_log_compression`.

High-priority fixes:

- Improved memory management in the Aurora writer instance that prevents restart of writer due to out of memory conditions during heavy workload in presence of reader instances within the Aurora DB cluster.

Comparison with Aurora MySQL Version 1

The following Amazon Aurora MySQL features are supported in Aurora MySQL Version 1 (compatible with MySQL 5.6), but these features are currently not supported in Aurora MySQL Version 2 (compatible with MySQL 5.7).

- Asynchronous key prefetch (AKP). For more information, see [Working with Asynchronous Key Prefetch in Amazon Aurora \(p. 765\)](#).
- Hash joins. For more information, see [Working with Hash Joins in Aurora MySQL \(p. 771\)](#).
- Native functions for synchronously invoking AWS Lambda functions. For more information, see [Invoking a Lambda Function with an Aurora MySQL Native Function \(p. 753\)](#).
- Scan batching. For more information, see [Aurora MySQL Database Engine Updates 2017-12-11 \(p. 846\)](#).
- Migrating data from MySQL using an Amazon S3 bucket. For more information, see [Migrating Data from MySQL by Using an Amazon S3 Bucket \(p. 590\)](#).

MySQL 5.7 compatibility

Aurora MySQL 2.04.8 is wire-compatible with MySQL 5.7 and includes features such as JSON support, spatial indexes, and generated columns. Aurora MySQL uses a native implementation of spatial indexing using z-order curves to deliver >20x better write performance and >10x better read performance than MySQL 5.7 for spatial datasets.

Aurora MySQL 2.04.8 does not currently support the following MySQL 5.7 features:

- Group replication plugin
- Increased page size
- InnoDB buffer pool loading at startup
- InnoDB full-text parser plugin

- Multisource replication
- Online buffer pool resizing
- Password validation plugin
- Query rewrite plugins
- Replication filtering
- The `CREATE TABLESPACE` SQL statement

Aurora MySQL Database Engine Updates 2019-11-14 (Version 2.04.7)

Version: 2.04.7

Aurora MySQL 2.04.7 is generally available. Aurora MySQL 2.x versions are compatible with MySQL 5.7 and Aurora MySQL 1.x versions are compatible with MySQL 5.6.

Currently supported Aurora MySQL releases are 1.14.* , 1.15.* , 1.16.* , 1.17.* , 1.18.* , 1.19.* , 2.01.* , 2.02.* , 2.03.* and 2.04.*.

You can restore a snapshot from a currently supported Aurora MySQL release into Aurora MySQL 2.04.7. You also have the option to upgrade existing Aurora MySQL 2.* database clusters to Aurora MySQL 2.04.7. You can't upgrade an existing Aurora MySQL 1.* cluster directly to 2.04.7; however, you can restore its snapshot to Aurora MySQL 2.04.7.

To create a cluster with an older version of Aurora MySQL, please specify the engine version through the AWS Management Console, the AWS CLI, or the RDS API.

If you have any questions or concerns, AWS Support is available on the community forums and through [AWS Premium Support](#). For more information, see [Maintaining an Amazon Aurora DB Cluster \(p. 408\)](#).

Note

This version is currently not available in the following AWS Regions: AWS GovCloud (US-East) [us-gov-east-1], AWS GovCloud (US-West) [us-gov-west-1], Asia Pacific (Hong Kong) [ap-east-1], and Middle East (Bahrain) [me-south-1]. There will be a separate announcement once it is made available.

Note

For information on how to upgrade your Aurora MySQL database cluster, see [Database Upgrades and Patches for Amazon Aurora MySQL \(p. 795\)](#).

Improvements

High-priority fixes:

Connection Handling

- Database availability has been improved to better service a surge in client connections while executing one or more DDLs. It is handled by temporarily creating additional threads when needed. You are advised to upgrade if the database becomes unresponsive following a surge in connections while processing DDL.

Engine Restart

- Fixed an issue of prolonged unavailability while restarting the engine. This addresses an issue in the buffer pool initialization. This issue occurs rarely but can potentially impact any supported release.

General stability fixes:

- Made improvements where queries accessing uncached data could be slower than usual. Customers experiencing unexplained elevated read latencies while accessing uncached data are encouraged to upgrade as they may be experiencing this issue.

Comparison with Aurora MySQL Version 1

The following Amazon Aurora MySQL features are supported in Aurora MySQL Version 1 (compatible with MySQL 5.6), but these features are currently not supported in Aurora MySQL Version 2 (compatible with MySQL 5.7).

- Asynchronous key prefetch (AKP). For more information, see [Working with Asynchronous Key Prefetch in Amazon Aurora \(p. 765\)](#).
- Hash joins. For more information, see [Working with Hash Joins in Aurora MySQL \(p. 771\)](#).
- Native functions for synchronously invoking AWS Lambda functions. For more information, see [Invoking a Lambda Function with an Aurora MySQL Native Function \(p. 753\)](#).
- Scan batching. For more information, see [Aurora MySQL Database Engine Updates 2017-12-11 \(p. 846\)](#).
- Migrating data from MySQL using an Amazon S3 bucket. For more information, see [Migrating Data from MySQL by Using an Amazon S3 Bucket \(p. 590\)](#).

MySQL 5.7 compatibility

Aurora MySQL 2.04.7 is wire-compatible with MySQL 5.7 and includes features such as JSON support, spatial indexes, and generated columns. Aurora MySQL uses a native implementation of spatial indexing using z-order curves to deliver >20x better write performance and >10x better read performance than MySQL 5.7 for spatial datasets.

Aurora MySQL 2.04.7 does not currently support the following MySQL 5.7 features:

- Group replication plugin
- Increased page size
- InnoDB buffer pool loading at startup
- InnoDB full-text parser plugin
- Multisource replication
- Online buffer pool resizing
- Password validation plugin
- Query rewrite plugins
- Replication filtering
- The `CREATE TABLESPACE` SQL statement

Aurora MySQL Database Engine Updates 2019-09-19 (Version 2.04.6)

Version: 2.04.6

Aurora MySQL 2.04.6 is generally available. Aurora MySQL 2.x versions are compatible with MySQL 5.7 and Aurora MySQL 1.x versions are compatible with MySQL 5.6.

You have the option to upgrade existing Aurora MySQL 2.* database clusters to Aurora MySQL 2.04.6. We do not allow in-place upgrade of Aurora MySQL 1.* clusters. This restriction will be lifted in a later Aurora MySQL 2.* release. You can restore snapshots of Aurora MySQL 1.14.*, 1.15.*, 1.16.*, 1.17.*, 1.18.*, 1.19.*, 2.01.*, 2.02.*, 2.03.* and 2.04.* into Aurora MySQL 2.04.6.

To use an older version of Aurora MySQL, you can create new database clusters by specifying the engine version through the AWS Management Console, the AWS CLI, or the Amazon RDS API.

If you have any questions or concerns, AWS Support is available on the community forums and through [AWS Premium Support](#). For more information, see [Maintaining an Amazon Aurora DB Cluster \(p. 408\)](#).

Note

This version is currently not available in the following AWS Regions: Europe (London) [eu-west-2], AWS GovCloud (US-East) [us-gov-east-1], AWS GovCloud (US-West) [us-gov-west-1], China (Ningxia) [cn-northwest-1], and Asia Pacific (Hong Kong) [ap-east-1]. There will be a separate announcement once it is made available.

Note

For information on how to upgrade your Aurora MySQL database cluster, see [Database Upgrades and Patches for Amazon Aurora MySQL \(p. 795\)](#).

Improvements

- Fixed an issue where the events in current binlog file on the master were not replicated on the slave if the value of the parameter `sync_binlog` was not set to 1.
- The default value of the parameter `aurora_binlog_replication_max_yield_seconds` has been changed to zero to prevent an increase in replication lag in favor of foreground query performance on the binlog master.

Integration of MySQL Bug Fixes

- Bug#23054591: PURGE BINARY LOGS TO is reading the whole binlog file and causing MySql to Stall

Comparison with Aurora MySQL Version 1

The following Amazon Aurora MySQL features are supported in Aurora MySQL Version 1 (compatible with MySQL 5.6), but these features are currently not supported in Aurora MySQL Version 2 (compatible with MySQL 5.7).

- Asynchronous key prefetch (AKP). For more information, see [Working with Asynchronous Key Prefetch in Amazon Aurora \(p. 765\)](#).
- Hash joins. For more information, see [Working with Hash Joins in Aurora MySQL \(p. 771\)](#).
- Native functions for synchronously invoking AWS Lambda functions. For more information, see [Invoking a Lambda Function with an Aurora MySQL Native Function \(p. 753\)](#).
- Scan batching. For more information, see [Aurora MySQL Database Engine Updates 2017-12-11 \(p. 846\)](#).
- Migrating data from MySQL using an Amazon S3 bucket. For more information, see [Migrating Data from MySQL by Using an Amazon S3 Bucket \(p. 590\)](#).

MySQL 5.7 compatibility

Aurora MySQL 2.04.6 is wire-compatible with MySQL 5.7 and includes features such as JSON support, spatial indexes, and generated columns. Aurora MySQL uses a native implementation of spatial indexing using z-order curves to deliver >20x better write performance and >10x better read performance than MySQL 5.7 for spatial datasets.

Aurora MySQL 2.04.6 does not currently support the following MySQL 5.7 features:

- Group replication plugin
- Increased page size
- InnoDB buffer pool loading at startup
- InnoDB full-text parser plugin
- Multisource replication
- Online buffer pool resizing
- Password validation plugin
- Query rewrite plugins
- Replication filtering
- The `CREATE TABLESPACE` SQL statement

Aurora MySQL Database Engine Updates 2019-07-08 (Version 2.04.5)

Version: 2.04.5

Aurora MySQL 2.04.5 is generally available. Aurora MySQL 2.x versions are compatible with MySQL 5.7 and Aurora MySQL 1.x versions are compatible with MySQL 5.6.

You have the option to upgrade existing Aurora MySQL 2.* database clusters to Aurora MySQL 2.04.5. We do not allow in-place upgrade of Aurora MySQL 1.* clusters. This restriction will be lifted in a later Aurora MySQL 2.* release. You can restore snapshots of Aurora MySQL 1.14.*, 1.15.*, 1.16.*, 1.17.*, 1.18.*, 1.19.*, 2.01.*, 2.02.*, 2.03.* and 2.04.* into Aurora MySQL 2.04.5.

If you have any questions or concerns, AWS Support is available on the community forums and through [AWS Premium Support](#). For more information, see [Maintaining an Amazon Aurora DB Cluster \(p. 408\)](#).

Note

For information on how to upgrade your Aurora MySQL database cluster, see [Database Upgrades and Patches for Amazon Aurora MySQL \(p. 795\)](#).

Improvements

- Fixed a race condition during storage volume growth that caused the database to restart.
- Fixed an internal communication failure during volume open that caused the database to restart.
- Added DDL recovery support for `ALTER TABLE ALGORITHM=INPLACE` on partitioned tables.
- Fixed an issue with DDL recovery of `ALTER TABLE ALGORITHM=COPY` that caused the database to restart.
- Improved Aurora Replica stability under heavy delete workload on the writer.
- Fixed a database restart caused by a deadlock between the thread performing full-text search index sync and the thread performing eviction of full-text search table from dictionary cache.
- Fixed a stability issue on the binlog slave during DDL replication while the connection to the binlog master is unstable.
- Fixed an out-of-memory issue in the full-text search code that caused the database to restart.
- Fixed an issue on the Aurora Writer that caused it to restart when the entire 64 TiB volume is used.
- Fixed a race condition in the Performance Schema feature that caused the database to restart.
- Fixed an issue that caused aborted connections when handling an error in network protocol management.

Comparison with Aurora MySQL Version 1

The following Amazon Aurora MySQL features are supported in Aurora MySQL Version 1 (compatible with MySQL 5.6), but these features are currently not supported in Aurora MySQL Version 2 (compatible with MySQL 5.7).

- Asynchronous key prefetch (AKP). For more information, see [Working with Asynchronous Key Prefetch in Amazon Aurora \(p. 765\)](#).
- Hash joins. For more information, see [Working with Hash Joins in Aurora MySQL \(p. 771\)](#).
- Native functions for synchronously invoking AWS Lambda functions. For more information, see [Invoking a Lambda Function with an Aurora MySQL Native Function \(p. 753\)](#).
- Scan batching. For more information, see [Aurora MySQL Database Engine Updates 2017-12-11 \(p. 846\)](#).
- Migrating data from MySQL using an Amazon S3 bucket. For more information, see [Migrating Data from MySQL by Using an Amazon S3 Bucket \(p. 590\)](#).

MySQL 5.7 compatibility

Aurora MySQL 2.04.5 is wire-compatible with MySQL 5.7 and includes features such as JSON support, spatial indexes, and generated columns. Aurora MySQL uses a native implementation of spatial indexing using z-order curves to deliver >20x better write performance and >10x better read performance than MySQL 5.7 for spatial datasets.

Aurora MySQL 2.04.5 does not currently support the following MySQL 5.7 features:

- Group replication plugin
- Increased page size
- InnoDB buffer pool loading at startup
- InnoDB full-text parser plugin
- Multisource replication
- Online buffer pool resizing
- Password validation plugin
- Query rewrite plugins
- Replication filtering
- The `CREATE TABLESPACE` SQL statement

Aurora MySQL Database Engine Updates 2019-05-29 (Version 2.04.4)

Version: 2.04.4

Aurora MySQL 2.04.4 is generally available. Aurora MySQL 2.x versions are compatible with MySQL 5.7 and Aurora MySQL 1.x versions are compatible with MySQL 5.6.

When creating a new Aurora MySQL DB cluster (including restoring a snapshot), you have the option of choosing compatibility with either MySQL 5.7 or MySQL 5.6. We do not allow in-place upgrade of Aurora MySQL 1.* clusters or restore of Aurora MySQL 1.* clusters from an Amazon S3 backup into Aurora MySQL 2.04.4. We plan to remove these restrictions in a later Aurora MySQL 2.* release.

You can restore snapshots of Aurora MySQL 1.14.*, 1.15.*, 1.16.*, 1.17.*, 1.18.*, 1.19.*, 2.01.*, 2.02.*, 2.03.*, and 2.04.* into Aurora MySQL 2.04.4.

If you have any questions or concerns, AWS Support is available on the community forums and through [AWS Premium Support](#). For more information, see [Maintaining an Amazon Aurora DB Cluster \(p. 408\)](#).

Note

This version is currently not available in the AWS GovCloud (US-West) [us-gov-west-1], Europe (Stockholm) [eu-north-1], China (Ningxia) [cn-northwest-1], and Asia Pacific (Hong Kong) [ap-east-1] AWS Regions. There will be a separate announcement once it is made available.

Note

For information on how to upgrade your Aurora MySQL database cluster, see [Database Upgrades and Patches for Amazon Aurora MySQL \(p. 795\)](#).

Improvements

- Fixed an issue that could cause failures when loading data into Aurora from S3.
- Fixed an issue that could cause failures when upload data from Aurora to S3.
- Fixed an issue that caused aborted connections when handling an error in network protocol management.
- Fixed an issue that could cause a crash when dealing with partitioned tables.
- Fixed an issue with the Performance Insights feature being unavailable in some regions.

Comparison with Aurora MySQL Version 1

The following Amazon Aurora MySQL features are supported in Aurora MySQL Version 1 (compatible with MySQL 5.6), but these features are currently not supported in Aurora MySQL Version 2 (compatible with MySQL 5.7).

- Asynchronous key prefetch (AKP). For more information, see [Working with Asynchronous Key Prefetch in Amazon Aurora \(p. 765\)](#).
- Hash joins. For more information, see [Working with Hash Joins in Aurora MySQL \(p. 771\)](#).
- Native functions for synchronously invoking AWS Lambda functions. For more information, see [Invoking a Lambda Function with an Aurora MySQL Native Function \(p. 753\)](#).
- Scan batching. For more information, see [Aurora MySQL Database Engine Updates 2017-12-11 \(p. 846\)](#).
- Migrating data from MySQL using an Amazon S3 bucket. For more information, see [Migrating Data from MySQL by Using an Amazon S3 Bucket \(p. 590\)](#).

MySQL 5.7 compatibility

Aurora MySQL 2.04.4 is wire-compatible with MySQL 5.7 and includes features such as JSON support, spatial indexes, and generated columns. Aurora MySQL uses a native implementation of spatial indexing using z-order curves to deliver >20x better write performance and >10x better read performance than MySQL 5.7 for spatial datasets.

Aurora MySQL 2.04.4 does not currently support the following MySQL 5.7 features:

- Group replication plugin
- Increased page size
- InnoDB buffer pool loading at startup
- InnoDB full-text parser plugin
- Multisource replication
- Online buffer pool resizing
- Password validation plugin

- Query rewrite plugins
- Replication filtering
- The `CREATE TABLESPACE` SQL statement

Aurora MySQL Database Engine Updates 2019-05-09 (Version 2.04.3)

Version: 2.04.3

Aurora MySQL 2.04.3 is generally available. Aurora MySQL 2.x versions are compatible with MySQL 5.7 and Aurora MySQL 1.x versions are compatible with MySQL 5.6.

When creating a new Aurora MySQL DB cluster (including restoring a snapshot), you have the option of choosing compatibility with either MySQL 5.7 or MySQL 5.6. We do not allow in-place upgrade of Aurora MySQL 1.* clusters or restore of Aurora MySQL 1.* clusters from an Amazon S3 backup into Aurora MySQL 2.04.3. We plan to remove these restrictions in a later Aurora MySQL 2.* release.

You can restore snapshots of Aurora MySQL 1.14.*, 1.15.*, 1.16.*, 1.17.*, 1.18.*, 1.19.*, 2.01.*, 2.02.*, 2.03.*, and 2.04.* into Aurora MySQL 2.04.3.

If you have any questions or concerns, AWS Support is available on the community forums and through [AWS Premium Support](#). For more information, see [Maintaining an Amazon Aurora DB Cluster \(p. 408\)](#).

Note

This version is currently not available in the AWS GovCloud (US-West) [us-gov-west-1] and China (Ningxia) [cn-northwest-1] AWS Regions. There will be a separate announcement once it is made available.

Note

For information on how to upgrade your Aurora MySQL database cluster, see [Database Upgrades and Patches for Amazon Aurora MySQL \(p. 795\)](#).

Improvements

- Fixed a bug in binlog replication that can cause an issue on Aurora instances configured as binlog slave.
- Fixed an out-of-memory condition when handling large stored routines.
- Fixed an error in handling certain kinds of `ALTER TABLE` commands.
- Fixed an issue with aborted connections because of an error in network protocol management.

Comparison with Aurora MySQL Version 1

The following Amazon Aurora MySQL features are supported in Aurora MySQL Version 1 (compatible with MySQL 5.6), but these features are currently not supported in Aurora MySQL Version 2 (compatible with MySQL 5.7).

- Asynchronous key prefetch (AKP). For more information, see [Working with Asynchronous Key Prefetch in Amazon Aurora \(p. 765\)](#).
- Hash joins. For more information, see [Working with Hash Joins in Aurora MySQL \(p. 771\)](#).
- Native functions for synchronously invoking AWS Lambda functions. For more information, see [Invoking a Lambda Function with an Aurora MySQL Native Function \(p. 753\)](#).
- Scan batching. For more information, see [Aurora MySQL Database Engine Updates 2017-12-11 \(p. 846\)](#).
- Migrating data from MySQL using an Amazon S3 bucket. For more information, see [Migrating Data from MySQL by Using an Amazon S3 Bucket \(p. 590\)](#).

MySQL 5.7 compatibility

Aurora MySQL 2.04.3 is wire-compatible with MySQL 5.7 and includes features such as JSON support, spatial indexes, and generated columns. Aurora MySQL uses a native implementation of spatial indexing using z-order curves to deliver >20x better write performance and >10x better read performance than MySQL 5.7 for spatial datasets.

Aurora MySQL 2.04.3 does not currently support the following MySQL 5.7 features:

- Group replication plugin
- Increased page size
- InnoDB buffer pool loading at startup
- InnoDB full-text parser plugin
- Multisource replication
- Online buffer pool resizing
- Password validation plugin
- Query rewrite plugins
- Replication filtering
- The `CREATE TABLESPACE` SQL statement

Aurora MySQL Database Engine Updates 2019-05-02 (Version 2.04.2)

Version: 2.04.2

Aurora MySQL 2.04.2 is generally available. Aurora MySQL 2.x versions are compatible with MySQL 5.7 and Aurora MySQL 1.x versions are compatible with MySQL 5.6.

When creating a new Aurora MySQL DB cluster (including restoring a snapshot), you have the option of choosing compatibility with either MySQL 5.7 or MySQL 5.6. We do not allow in-place upgrade of Aurora MySQL 1.* clusters or restore of Aurora MySQL 1.* clusters from an Amazon S3 backup into Aurora MySQL 2.04.2. We plan to remove these restrictions in a later Aurora MySQL 2.* release.

You can restore snapshots of Aurora MySQL 1.14.*, 1.15.*, 1.16.*, 1.17.*, 1.18.*, 1.19.*, 2.01.*, 2.02.*, 2.03.*, 2.04.0, and 2.04.1 into Aurora MySQL 2.04.2.

If you have any questions or concerns, AWS Support is available on the community forums and through [AWS Premium Support](#). For more information, see [Maintaining an Amazon Aurora DB Cluster \(p. 408\)](#).

Note

This version is currently not available in the AWS GovCloud (US-West) [us-gov-west-1] and China (Ningxia) [cn-northwest-1] AWS Regions. There will be a separate announcement once it is made available.

Note

For information on how to upgrade your Aurora MySQL database cluster, see [Database Upgrades and Patches for Amazon Aurora MySQL \(p. 795\)](#).

Improvements

- Added support for SSL binlog replication using custom certificates. For information on using SSL binlog replication in Aurora MySQL, see [mysql_rds_import_binlog_ssl_material](#).
- Fixed a deadlock on the Aurora primary instance that occurs when a table with a Full Text Search index is being optimized.

- Fixed an issue on the Aurora Replicas where performance of certain queries using `SELECT(*)` could be impacted on tables that have secondary indexes.
- Fixed a condition that resulted in Error 1032 being posted.
- Improved the stability of Aurora Replicas by fixing multiple deadlocks.

Integration of MySQL Bug Fixes

- Bug #24829050 - INDEX_MERGE_INTERSECTION OPTIMIZATION CAUSES WRONG QUERY RESULTS

Comparison with Aurora MySQL Version 1

The following Amazon Aurora MySQL features are supported in Aurora MySQL Version 1 (compatible with MySQL 5.6), but these features are currently not supported in Aurora MySQL Version 2 (compatible with MySQL 5.7).

- Asynchronous key prefetch (AKP). For more information, see [Working with Asynchronous Key Prefetch in Amazon Aurora \(p. 765\)](#).
- Hash joins. For more information, see [Working with Hash Joins in Aurora MySQL \(p. 771\)](#).
- Native functions for synchronously invoking AWS Lambda functions. For more information, see [Invoking a Lambda Function with an Aurora MySQL Native Function \(p. 753\)](#).
- Scan batching. For more information, see [Aurora MySQL Database Engine Updates 2017-12-11 \(p. 846\)](#).
- Migrating data from MySQL using an Amazon S3 bucket. For more information, see [Migrating Data from MySQL by Using an Amazon S3 Bucket \(p. 590\)](#).

MySQL 5.7 compatibility

Aurora MySQL 2.04.2 is wire-compatible with MySQL 5.7 and includes features such as JSON support, spatial indexes, and generated columns. Aurora MySQL uses a native implementation of spatial indexing using z-order curves to deliver >20x better write performance and >10x better read performance than MySQL 5.7 for spatial datasets.

Aurora MySQL 2.04.2 does not currently support the following MySQL 5.7 features:

- Group replication plugin
- Increased page size
- InnoDB buffer pool loading at startup
- InnoDB full-text parser plugin
- Multisource replication
- Online buffer pool resizing
- Password validation plugin
- Query rewrite plugins
- Replication filtering
- The `CREATE TABLESPACE` SQL statement

Aurora MySQL Database Engine Updates 2019-03-25 (Version 2.04.1)

Version: 2.04.1

Aurora MySQL 2.04.1 is generally available. Aurora MySQL 2.x versions are compatible with MySQL 5.7 and Aurora MySQL 1.x versions are compatible with MySQL 5.6.

When creating a new Aurora MySQL DB cluster (including restoring a snapshot), you have the option of choosing compatibility with either MySQL 5.7 or MySQL 5.6. We do not allow in-place upgrade of Aurora MySQL 1.* clusters or restore of Aurora MySQL 1.* clusters from an Amazon S3 backup into Aurora MySQL 2.04.1. We plan to remove these restrictions in a later Aurora MySQL 2.* release.

You can restore snapshots of Aurora MySQL 1.14.*, 1.15.*, 1.16.*, 1.17.*, 1.18.*, 1.19.*, 2.01.*, 2.02.*, 2.03.*, 2.04.0 into Aurora MySQL 2.04.1.

If you have any questions or concerns, AWS Support is available on the community forums and through [AWS Premium Support](#). For more information, see [Maintaining an Amazon Aurora DB Cluster \(p. 408\)](#).

Note

This version is currently not available in the AWS GovCloud (US-West) [us-gov-west-1] region. There will be a separate announcement once it is made available.

Note

The procedure to upgrade your DB cluster has changed. For more information, see [Database Upgrades and Patches for Amazon Aurora MySQL \(p. 795\)](#).

Improvements

- Fixed an issue where an Aurora MySQL 5.6 snapshot for versions lower than 1.16 could not be restored to the latest Aurora MySQL 5.7 cluster.

Aurora MySQL Database Engine Updates 2019-03-25 (Version 2.04)

Version: 2.04

Aurora MySQL 2.04 is generally available. Aurora MySQL 2.x versions are compatible with MySQL 5.7 and Aurora MySQL 1.x versions are compatible with MySQL 5.6.

When creating a new Aurora MySQL DB cluster (including restoring a snapshot), you have the option of choosing compatibility with either MySQL 5.7 or MySQL 5.6. We do not allow in-place upgrade of Aurora MySQL 1.* clusters or restore of Aurora MySQL 1.* clusters from an Amazon S3 backup into Aurora MySQL 2.04.0. We plan to remove these restrictions in a later Aurora MySQL 2.* release.

You can restore snapshots of Aurora MySQL 1.19.*, 2.01.*, 2.02.*, and 2.03.* into Aurora MySQL 2.04.0. You cannot restore snapshots of Aurora MySQL 1.14.* or lower, 1.15.*, 1.16.*, 1.17.*, 1.18.* into Aurora MySQL 2.04.0. This restriction is removed in Aurora MySQL 2.04.1.

If you have any questions or concerns, AWS Support is available on the community forums and through [AWS Premium Support](#). For more information, see [Maintaining an Amazon Aurora DB Cluster \(p. 408\)](#).

Note

This version is currently not available in the AWS GovCloud (US-West) [us-gov-west-1] region. There will be a separate announcement once it is made available.

Note

The procedure to upgrade your DB cluster has changed. For more information, see [Database Upgrades and Patches for Amazon Aurora MySQL \(p. 795\)](#).

Improvements

- Supports GTID-based replication. For information about using GTID-based replication with Aurora MySQL, see [Using GTID-Based Replication for Aurora MySQL \(p. 698\)](#).

- Fixed an issue where an Aurora Replica incorrectly throws a `running in read-only mode` error when a statement deleting or updating rows in a temporary table contains an InnoDB subquery.

Integration of MySQL Bug Fixes

- Bug #26225783: MYSQL CRASH ON CREATE TABLE (REPRODUCABLE) -> INNODB: ALONG SEMAPHORE WAIT.

Aurora MySQL Database Engine Updates 2019-02-07

Version: 2.03.4

Aurora MySQL 2.03.4 is generally available. Aurora MySQL 2.x versions are compatible with MySQL 5.7 and Aurora MySQL 1.x versions are compatible with MySQL 5.6.

When creating a new Aurora MySQL DB cluster (including restoring a snapshot), you can choose compatibility with either MySQL 5.7 or MySQL 5.6.

We don't allow in-place upgrade of Aurora MySQL 1.* clusters into Aurora MySQL 2.03.4 or restore to Aurora MySQL 2.03.4 from an Amazon S3 backup. We plan to remove these restrictions in a later Aurora MySQL 2.* release.

If you have any questions or concerns, AWS Support is available on the community forums and through [AWS Premium Support](#). For more information, see [Maintaining an Amazon Aurora DB Cluster \(p. 408\)](#).

Note

This version is currently not available in the AWS GovCloud (US-West) [us-gov-west-1] and China (Beijing) [cn-north-1] regions. There will be a separate announcement once it is made available.

Note

The procedure to upgrade your DB cluster has changed. For more information, see [Database Upgrades and Patches for Amazon Aurora MySQL \(p. 795\)](#).

Improvements

- Support for UTF8MB4 Unicode 9.0 accent-sensitive and case-insensitive collation, `utf8mb4_0900_as_ci`.

Aurora MySQL Database Engine Updates 2019-01-18

Version: 2.03.3

Aurora MySQL 2.03.3 is generally available. Aurora MySQL 2.x versions are compatible with MySQL 5.7 and Aurora MySQL 1.x versions are compatible with MySQL 5.6.

When creating a new Aurora MySQL DB cluster (including restoring a snapshot), you can choose compatibility with either MySQL 5.7 or MySQL 5.6.

We don't allow in-place upgrade of Aurora MySQL 1.* clusters into Aurora MySQL 2.03.3 or restore to Aurora MySQL 2.03.3 from an Amazon S3 backup. We plan to remove these restrictions in a later Aurora MySQL 2.* release.

If you have any questions or concerns, AWS Support is available on the community forums and through [AWS Premium Support](#). For more information, see [Maintaining an Amazon Aurora DB Cluster \(p. 408\)](#).

Note

This version is currently not available in the AWS GovCloud (US-West) [us-gov-west-1] and China (Beijing) [cn-north-1] regions. There will be a separate announcement once it is made available.

Note

The procedure to upgrade your DB cluster has changed. For more information, see [Database Upgrades and Patches for Amazon Aurora MySQL \(p. 795\)](#).

Improvements

- Fixed an issue where an Aurora Replica might become dead-latched when running a backward scan on an index.
- Fixed an issue where an Aurora Replica might restart when the Aurora primary instance runs in-place DDL operations on partitioned tables.
- Fixed an issue where an Aurora Replica might restart during query cache invalidation after a DDL operation on the Aurora primary instance.
- Fixed an issue where an Aurora Replica might restart during a `SELECT` query on a table while the Aurora primary instance runs truncation on that table.
- Fixed a wrong result issue with MyISAM temporary tables where only indexed columns are accessed.
- Fixed an issue in slow logs that generated incorrect large values for `query_time` and `lock_time` periodically after approximately 40,000 queries.
- Fixed an issue where a schema named "tmp" could cause migration from RDS MySQL to Aurora MySQL to become stuck.
- Fixed an issue where the audit log might have missing events during log rotation.
- Fixed an issue where the Aurora primary instance restored from an Aurora 5.6 snapshot might restart when the Fast Online DDL feature in the lab mode is enabled.
- Fixed an issue where the CPU usage is 100% caused by the dictionary stats thread.
- Fixed an issue where an Aurora Replica might restart when running a `CHECK TABLE` statement.

Integration of MySQL Bug Fixes

- Bug #25361251: INCORRECT BEHAVIOR WITH INSERT ON DUPLICATE KEY IN SP
- Bug #26734162: INCORRECT BEHAVIOR WITH INSERT OF BLOB + ON DUPLICATE KEY UPDATE
- Bug #27460607: INCORRECT BEHAVIOR OF IODKU WHEN INSERT SELECT's SOURCE TABLE IS EMPTY
- SELECT DISTINCT NOT RETURNING DISTINCT ROWS IN 5.7 (Bug #22343910)
- Delete from joined tables with where using derived table fails with error 1093 (Bug #23074801).
- GCOLS: INCORRECT BEHAVIOR WITH CHARSET CHANGES (Bug #25287633).

Aurora MySQL Database Engine Updates 2019-01-09

Version: 2.03.2

Aurora MySQL 2.03.2 is generally available. Aurora MySQL 2.x versions are compatible with MySQL 5.7 and Aurora MySQL 1.x versions are compatible with MySQL 5.6.

When creating a new Aurora MySQL DB cluster (including restoring a snapshot), you can choose compatibility with either MySQL 5.7 or MySQL 5.6.

We don't allow in-place upgrade of Aurora MySQL 1.* clusters into Aurora MySQL 2.03.2 or restore to Aurora MySQL 2.03.2 from an Amazon S3 backup. We plan to remove these restrictions in a later Aurora MySQL 2.* release.

If you have any questions or concerns, AWS Support is available on the community forums and through [AWS Premium Support](#). For more information, see [Maintaining an Amazon Aurora DB Cluster \(p. 408\)](#).

Note

This version is currently not available in the AWS GovCloud (US-West) [us-gov-west-1] and China (Beijing) [cn-north-1] regions. There will be a separate announcement once it is made available.

Note

The procedure to upgrade your DB cluster has changed. For more information, see [Database Upgrades and Patches for Amazon Aurora MySQL \(p. 795\)](#).

Improvements

- **Aurora Version Selector** – Starting with Aurora MySQL 2.03.2, you can choose from among multiple versions of MySQL 5.7-compatible Aurora on the AWS Management Console. For more information, see [Aurora MySQL Engine Versions \(p. 794\)](#).

Aurora MySQL Database Engine Updates 2018-10-24

Version: 2.03.1

Aurora MySQL 2.03.1 is generally available. Aurora MySQL 2.x versions are compatible with MySQL 5.7 and Aurora MySQL 1.x versions are compatible with MySQL 5.6.

When creating a new Aurora MySQL DB cluster, you can choose compatibility with either MySQL 5.7 or MySQL 5.6. When restoring a MySQL 5.6-compatible snapshot, you can choose compatibility with either MySQL 5.7 or MySQL 5.6.

You can restore snapshots of Aurora MySQL 1.14.*, 1.15.*, 1.16.*, 1.17.*, 1.18.*, 2.01.*, 2.02.*, and 2.03 into Aurora MySQL 2.03.1.

We don't allow in-place upgrade of Aurora MySQL 1.* clusters into Aurora MySQL 2.03.1 or restore to Aurora MySQL 2.03.1 from an Amazon S3 backup. We plan to remove these restrictions in a later Aurora MySQL 2.* release.

If you have any questions or concerns, AWS Support is available on the community forums and through [AWS Premium Support](#). For more information, see [Maintaining an Amazon Aurora DB Cluster \(p. 408\)](#).

Note

This version is currently not available in the AWS GovCloud (US-West) [us-gov-west-1] and China (Beijing) [cn-north-1] regions. There will be a separate announcement once it is made available.

Improvements

- Fix an issue where the Aurora Writer might restart when running transaction deadlock detection.

Aurora MySQL Database Engine Updates 2018-10-11

Version: 2.03

Aurora MySQL 2.03 is generally available. Aurora MySQL 2.x versions are compatible with MySQL 5.7 and Aurora MySQL 1.x versions are compatible with MySQL 5.6.

When creating a new Aurora MySQL DB cluster, you can choose compatibility with either MySQL 5.7 or MySQL 5.6. When restoring a MySQL 5.6-compatible snapshot, you can choose compatibility with either MySQL 5.7 or MySQL 5.6.

You can restore snapshots of Aurora MySQL 1.14.* , 1.15.* , 1.16.* , 1.17.* , 1.18.* , 2.01.* , and 2.02.* into Aurora MySQL 2.03.

We don't allow in-place upgrade of Aurora MySQL 1.* clusters into Aurora MySQL 2.03 or restore to Aurora MySQL 2.03 from an Amazon S3 backup. We plan to remove these restrictions in a later Aurora MySQL 2.* release.

Note

This version is currently not available in the AWS GovCloud (US-West) [us-gov-west-1] and China (Beijing) [cn-north-1] regions. There will be a separate announcement once it is made available.

If you have any questions or concerns, AWS Support is available on the community forums and through [AWS Premium Support](#). For more information, see [Maintaining an Amazon Aurora DB Cluster \(p. 408\)](#).

Improvements

- Performance schema is available.
- Fixed an issue where zombie sessions with killed state might consume more CPU.
- Fixed a dead latch issue when a read-only transaction is acquiring a lock on a record on the Aurora Writer.
- Fixed an issue where the Aurora Replica without customer workload might have high CPU utilization.
- Multiple fixes on issues that might cause the Aurora Replica or the Aurora writer to restart.
- Added capability to skip diagnostic logging when the disk throughput limit is reached.
- Fixed a memory leak issue when binlog is enabled on the Aurora Writer.

Integration of MySQL community edition bug fixes

- REVERSE SCAN ON A PARTITIONED TABLE DOES ICP - ORDER BY DESC (Bug #24929748).
- JSON_OBJECT CREATES INVALID JSON CODE (Bug#26867509).
- INSERTING LARGE JSON DATA TAKES AN INORDINATE AMOUNT OF TIME (Bug #22843444).
- PARTITIONED TABLES USE MORE MEMORY IN 5.7 THAN 5.6 (Bug #25080442).

Aurora MySQL Database Engine Updates 2018-10-08

Version: 2.02.5

Aurora MySQL 2.02.5 is generally available. Aurora MySQL 2.x versions are compatible with MySQL 5.7 and Aurora MySQL 1.x versions are compatible with MySQL 5.6.

When creating a new Aurora MySQL DB cluster, you can choose compatibility with either MySQL 5.7 or MySQL 5.6. When restoring a MySQL 5.6-compatible snapshot, you can choose compatibility with either MySQL 5.7 or MySQL 5.6.

You can restore snapshots of Aurora MySQL 1.14.* , 1.15.* , 1.16.* , 1.17.* , 1.18.* , 2.01.* , and 2.02.* into Aurora MySQL 2.02.5. You can also perform an in-place upgrade from Aurora MySQL 2.01.* or 2.02.* to Aurora MySQL 2.02.5.

We don't allow in-place upgrade of Aurora MySQL 1.* clusters into Aurora MySQL 2.02.5 or restore to Aurora MySQL 2.02.5 from an Amazon S3 backup. We plan to remove these restrictions in a later Aurora MySQL 2.* release.

The performance schema is disabled for this release of Aurora MySQL 5.7. Upgrade to Aurora 2.03 for performance schema support.

Note

This version is currently not available in the AWS GovCloud (US-West) [us-gov-west-1] and China (Beijing) [cn-north-1] regions. There will be a separate announcement once it is made available.

If you have any questions or concerns, AWS Support is available on the community forums and through [AWS Premium Support](#). For more information, see [Maintaining an Amazon Aurora DB Cluster \(p. 408\)](#).

Improvements

- Fix an issue where an Aurora Replica might restart when it is doing a reverse scan on a table.

Aurora MySQL Database Engine Updates 2018-09-21

Version: 2.02.4

Aurora MySQL 2.02.4 is generally available. Aurora MySQL 2.x versions are compatible with MySQL 5.7 and Aurora MySQL 1.x versions are compatible with MySQL 5.6.

When creating a new Aurora MySQL DB cluster, you can choose compatibility with either MySQL 5.7 or MySQL 5.6. When restoring a MySQL 5.6-compatible snapshot, you can choose compatibility with either MySQL 5.7 or MySQL 5.6.

You can restore snapshots of Aurora MySQL 1.14.*, 1.15.*, 1.16.*, 1.17.*, 1.18.*, 2.01.*, and 2.02.* into Aurora MySQL 2.02.4. You can also perform an in-place upgrade from Aurora MySQL 2.01.* or 2.02.* to Aurora MySQL 2.02.4.

We don't allow in-place upgrade of Aurora MySQL 1.* clusters into Aurora MySQL 2.02.4 or restore to Aurora MySQL 2.02.4 from an Amazon S3 backup. We plan to remove these restrictions in a later Aurora MySQL 2.* release.

The performance schema is disabled for this release of Aurora MySQL 5.7. Upgrade to Aurora 2.03 for performance schema support.

If you have any questions or concerns, AWS Support is available on the community forums and through [AWS Premium Support](#). For more information, see [Maintaining an Amazon Aurora DB Cluster \(p. 408\)](#).

Improvements

- Fixed a stability issue related to Full Text Search indexes on tables restored from an Aurora MySQL 5.6 snapshot.

Aurora MySQL Database Engine Updates 2018-08-23

Version: 2.02.3

Aurora MySQL 2.02.3 is generally available. Aurora MySQL 2.x versions are compatible with MySQL 5.7 and Aurora MySQL 1.x versions are compatible with MySQL 5.6.

When creating a new Aurora MySQL DB cluster, you can choose compatibility with either MySQL 5.7 or MySQL 5.6. When restoring a MySQL 5.6-compatible snapshot, you can choose compatibility with either MySQL 5.7 or MySQL 5.6.

You can restore snapshots of Aurora MySQL 1.14.*, 1.15.*, 1.16.*, 1.17.*, 2.01.*, and 2.02.* into Aurora MySQL 2.02.3. You can also perform an in-place upgrade from Aurora MySQL 2.01.* or 2.02.* to Aurora MySQL 2.02.3.

We don't allow in-place upgrade of Aurora MySQL 1.* clusters into Aurora MySQL 2.02.3 or restore to Aurora MySQL 2.02.3 from an Amazon S3 backup. We plan to remove these restrictions in a later Aurora MySQL 2.* release.

The performance schema is disabled for this release of Aurora MySQL 5.7. Upgrade to Aurora 2.03 for performance schema support.

Note

This version is currently not available in the AWS GovCloud (US-West) [us-gov-west-1] and China (Beijing) [cn-north-1] regions. There will be a separate announcement once it is made available.

If you have any questions or concerns, AWS Support is available on the community forums and through [AWS Premium Support](#). For more information, see [Maintaining an Amazon Aurora DB Cluster \(p. 408\)](#).

Comparison with Aurora MySQL 5.6

The following Amazon Aurora MySQL features are supported in Aurora MySQL 5.6, but these features are currently not supported in Aurora MySQL 5.7.

- Asynchronous key prefetch (AKP). For more information, see [Working with Asynchronous Key Prefetch in Amazon Aurora \(p. 765\)](#).
- Hash joins. For more information, see [Working with Hash Joins in Aurora MySQL \(p. 771\)](#).
- Native functions for synchronously invoking AWS Lambda functions. These functions are available for MySQL 5.7-compatible clusters in Aurora MySQL 2.06 and higher. For more information, see [Invoking a Lambda Function with an Aurora MySQL Native Function \(p. 753\)](#).
- Scan batching. For more information, see [Aurora MySQL Database Engine Updates 2017-12-11 \(p. 846\)](#).
- Migrating data from MySQL using an Amazon S3 bucket. For more information, see [Migrating Data from MySQL by Using an Amazon S3 Bucket \(p. 590\)](#).

Currently, Aurora MySQL 2.01 does not support features added in Aurora MySQL version 1.16 and later. For information about Aurora MySQL version 1.16, see [Aurora MySQL Database Engine Updates 2017-12-11 \(p. 846\)](#).

MySQL 5.7 compatibility

Aurora MySQL 2.01 is wire-compatible with MySQL 5.7 and includes features such as JSON support, spatial indexes, and generated columns. Aurora MySQL uses a native implementation of spatial indexing using z-order curves to deliver >20x better write performance and >10x better read performance than MySQL 5.7 for spatial datasets.

Aurora MySQL 2.01 does not currently support the following MySQL 5.7 features:

- Global transaction identifiers (GTIDs). Aurora MySQL supports GTIDs in version 2.04 and higher.
- Group replication plugin
- Increased page size
- InnoDB buffer pool loading at startup
- InnoDB full-text parser plugin
- Multisource replication
- Online buffer pool resizing
- Password validation plugin
- Query rewrite plugins
- Replication filtering
- The `CREATE TABLESPACE` SQL statement

- X Protocol

CLI differences between Aurora MySQL 2.x and Aurora MySQL 1.x

- The engine name for Aurora MySQL 2.x is `aurora-mysql`; the engine name for Aurora MySQL 1.x continues to be `aurora`.
- The engine version for Aurora MySQL 2.x is `5.7.12`; the engine version for Aurora MySQL 1.x continues to be `5.6.10ann`.
- The default parameter group for Aurora MySQL 2.x is `default.aurora-mysql5.7`; the default parameter group for Aurora MySQL 1.x continues to be `default.aurora5.6`.
- The DB cluster parameter group family name for Aurora MySQL 2.x is `aurora-mysql5.7`; the DB cluster parameter group family name for Aurora MySQL 1.x continues to be `aurora5.6`.

Refer to the Aurora documentation for the full set of CLI commands and differences between Aurora MySQL 2.x and Aurora MySQL 1.x.

Improvements

- Fixed an issue where an Aurora Replica can restart when using optimistic cursor restores while reading records.
- Updated the default value of the parameter `innodb_stats_persistent_sample_pages` to 128 to improve index statistics.
- Fixed an issue where an Aurora Replica might restart when it accesses a small table that is being concurrently modified on the Aurora primary instance.
- Fixed `ANALYZE TABLE` to stop flushing the table definition cache.
- Fixed an issue where the Aurora primary instance or an Aurora Replica might restart when converting a point query for geospatial to a search range.

Aurora MySQL Database Engine Updates 2018-06-04

Version: 2.02.2

Aurora MySQL 2.02.2 is generally available. Aurora MySQL 2.x versions are compatible with MySQL 5.7 and Aurora MySQL 1.x versions are compatible with MySQL 5.6.

When creating a new Aurora MySQL DB cluster, you can choose compatibility with either MySQL 5.7 or MySQL 5.6. When restoring a MySQL 5.6-compatible snapshot, you can choose compatibility with either MySQL 5.7 or MySQL 5.6.

You can restore snapshots of Aurora MySQL 1.14*, 1.15*, 1.16*, 1.17*, 2.01*, and 2.02 into Aurora MySQL 2.02.2. You can also perform an in-place upgrade from Aurora MySQL 2.01* or 2.02 to Aurora MySQL 2.02.2.

We don't allow in-place upgrade of Aurora MySQL 1.* clusters into Aurora MySQL 2.02.2 or restore to Aurora MySQL 2.02.2 from an Amazon S3 backup. We plan to remove these restrictions in a later Aurora MySQL 2.* release.

The performance schema is disabled for this release of Aurora MySQL 5.7. Upgrade to Aurora 2.03 for performance schema support.

Note

This version is currently not available in the AWS GovCloud (US-West) [us-gov-west-1] and China (Beijing) [cn-north-1] regions. There will be a separate announcement once it is made available.

If you have any questions or concerns, AWS Support is available on the community forums and through [AWS Premium Support](#). For more information, see [Maintaining an Amazon Aurora DB Cluster \(p. 408\)](#).

Comparison with Aurora MySQL 5.6

The following Amazon Aurora MySQL features are supported in Aurora MySQL 5.6, but these features are currently not supported in Aurora MySQL 5.7.

- Asynchronous key prefetch (AKP). For more information, see [Working with Asynchronous Key Prefetch in Amazon Aurora \(p. 765\)](#).
- Hash joins. For more information, see [Working with Hash Joins in Aurora MySQL \(p. 771\)](#).
- Native functions for synchronously invoking AWS Lambda functions. For more information, see [Invoking a Lambda Function with an Aurora MySQL Native Function \(p. 753\)](#).
- Scan batching. For more information, see [Aurora MySQL Database Engine Updates 2017-12-11 \(p. 846\)](#).
- Migrating data from MySQL using an Amazon S3 bucket. For more information, see [Migrating Data from MySQL by Using an Amazon S3 Bucket \(p. 590\)](#).

Currently, Aurora MySQL 2.01 does not support features added in Aurora MySQL version 1.16 and later. For information about Aurora MySQL version 1.16, see [Aurora MySQL Database Engine Updates 2017-12-11 \(p. 846\)](#).

MySQL 5.7 compatibility

Aurora MySQL 2.01 is wire-compatible with MySQL 5.7 and includes features such as JSON support, spatial indexes, and generated columns. Aurora MySQL uses a native implementation of spatial indexing using z-order curves to deliver >20x better write performance and >10x better read performance than MySQL 5.7 for spatial datasets.

Aurora MySQL 2.01 does not currently support the following MySQL 5.7 features:

- Global transaction identifiers (GTIDs). Aurora MySQL supports GTIDs in version 2.04 and higher.
- Group replication plugin
- Increased page size
- InnoDB buffer pool loading at startup
- InnoDB full-text parser plugin
- Multisource replication
- Online buffer pool resizing
- Password validation plugin
- Query rewrite plugins
- Replication filtering
- The `CREATE TABLESPACE` SQL statement
- X Protocol

CLI differences between Aurora MySQL 2.x and Aurora MySQL 1.x

- The engine name for Aurora MySQL 2.x is `aurora-mysql`; the engine name for Aurora MySQL 1.x continues to be `aurora`.
- The engine version for Aurora MySQL 2.x is `5.7.12`; the engine version for Aurora MySQL 1.x continues to be `5.6.10ann`.
- The default parameter group for Aurora MySQL 2.x is `default.aurora-mysql5.7`; the default parameter group for Aurora MySQL 1.x continues to be `default.aurora5.6`.

- The DB cluster parameter group family name for Aurora MySQL 2.x is `aurora-mysql5.7`; the DB cluster parameter group family name for Aurora MySQL 1.x continues to be `aurora5.6`.

Refer to the Aurora documentation for the full set of CLI commands and differences between Aurora MySQL 2.x and Aurora MySQL 1.x.

Improvements

- Fixed an issue where an Aurora Writer can occasionally restart when tracking Aurora Replica progress.
- Fixed an issue where an Aurora Replica restarts or throws an error when a partitioned table is accessed after running index create or drop statements on the table on the Aurora Writer.
- Fixed an issue where a table on an Aurora Replica is inaccessible while it is applying the changes caused by running ALTER table ADD/DROP column statements on the Aurora Writer.

Aurora MySQL Database Engine Updates 2018-05-03

Version: 2.02

Aurora MySQL 2.02 is generally available. Aurora MySQL 2.x versions are compatible with MySQL 5.7 and Aurora MySQL 1.x versions are compatible with MySQL 5.6.

When creating a new Aurora MySQL DB cluster, you can choose compatibility with either MySQL 5.7 or MySQL 5.6. When restoring a MySQL 5.6-compatible snapshot, you can choose compatibility with either MySQL 5.7 or MySQL 5.6.

You can restore snapshots of Aurora MySQL 1.14*, 1.15*, 1.16*, 1.17* and 2.01* into Aurora MySQL 2.02. You can also perform an in-place upgrade from Aurora MySQL 2.01* to Aurora MySQL 2.02.

We don't allow in-place upgrade of Aurora MySQL 1.x clusters into Aurora MySQL 2.02 or restore to Aurora MySQL 2.02 from an Amazon S3 backup. We plan to remove these restrictions in a later Aurora MySQL 2.x release.

The performance schema is disabled for this release of Aurora MySQL 5.7. Upgrade to Aurora 2.03 for performance schema support.

If you have any questions or concerns, AWS Support is available on the community forums and through [AWS Premium Support](#). For more information, see [Maintaining an Amazon Aurora DB Cluster \(p. 408\)](#).

Comparison with Aurora MySQL 5.6

The following Amazon Aurora MySQL features are supported in Aurora MySQL 5.6, but these features are currently not supported in Aurora MySQL 5.7.

- Asynchronous key prefetch (AKP). For more information, see [Working with Asynchronous Key Prefetch in Amazon Aurora \(p. 765\)](#).
- Hash joins. For more information, see [Working with Hash Joins in Aurora MySQL \(p. 771\)](#).
- Native functions for synchronously invoking AWS Lambda functions. For more information, see [Invoking a Lambda Function with an Aurora MySQL Native Function \(p. 753\)](#).
- Scan batching. For more information, see [Aurora MySQL Database Engine Updates 2017-12-11 \(p. 846\)](#).
- Migrating data from MySQL using an Amazon S3 bucket. For more information, see [Migrating Data from MySQL by Using an Amazon S3 Bucket \(p. 590\)](#).

Currently, Aurora MySQL 2.01 does not support features added in Aurora MySQL version 1.16 and later. For information about Aurora MySQL version 1.16, see [Aurora MySQL Database Engine Updates 2017-12-11 \(p. 846\)](#).

MySQL 5.7 compatibility

Aurora MySQL 2.01 is wire-compatible with MySQL 5.7 and includes features such as JSON support, spatial indexes, and generated columns. Aurora MySQL uses a native implementation of spatial indexing using z-order curves to deliver >20x better write performance and >10x better read performance than MySQL 5.7 for spatial datasets.

Aurora MySQL 2.01 does not currently support the following MySQL 5.7 features:

- Global transaction identifiers (GTIDs). Aurora MySQL supports GTIDs in version 2.04 and higher.
- Group replication plugin
- Increased page size
- InnoDB buffer pool loading at startup
- InnoDB full-text parser plugin
- Multisource replication
- Online buffer pool resizing
- Password validation plugin
- Query rewrite plugins
- Replication filtering
- The `CREATE TABLESPACE` SQL statement
- X Protocol

CLI differences between Aurora MySQL 2.x and Aurora MySQL 1.x

- The engine name for Aurora MySQL 2.x is `aurora-mysql`; the engine name for Aurora MySQL 1.x continues to be `aurora`.
- The engine version for Aurora MySQL 2.x is `5.7.12`; the engine version for Aurora MySQL 1.x continues to be `5.6.10ann`.
- The default parameter group for Aurora MySQL 2.x is `default.aurora-mysql5.7`; the default parameter group for Aurora MySQL 1.x continues to be `default.aurora5.6`.
- The DB cluster parameter group family name for Aurora MySQL 2.x is `aurora-mysql5.7`; the DB cluster parameter group family name for Aurora MySQL 1.x continues to be `aurora5.6`.

Refer to the Aurora documentation for the full set of CLI commands and differences between Aurora MySQL 2.x and Aurora MySQL 1.x.

Improvements

- Fixed an issue where an Aurora Writer restarts when running `INSERT` statements and exploiting the Fast Insert optimization.
- Fixed an issue where an Aurora Replica restarts when running `ALTER DATABASE` statements on the Aurora Replica.
- Fixed an issue where an Aurora Replica restarts when running queries on tables that have just been dropped on the Aurora Writer.
- Fixed an issue where an Aurora Replica restarts when setting `innodb_adaptive_hash_index` to `OFF` on the Aurora Replica.
- Fixed an issue where an Aurora Replica restarts when running `TRUNCATE TABLE` queries on the Aurora Writer.
- Fixed an issue where the Aurora Writer freezes in certain circumstances when running `INSERT` statements. On a multi-node cluster, this can result in a failover.
- Fixed a memory leak associated with setting session variables.

- Fixed an issue where the Aurora Writer freezes in certain circumstances associated with purging undo for tables with generated columns.
- Fixed an issue where the Aurora Writer can sometimes restart when binary logging is enabled.

Integration of MySQL Bug Fixes

- Left join returns incorrect results on the outer side (Bug #22833364).

Aurora MySQL Database Engine Updates 2018-03-13

Version: 2.01.1

Aurora MySQL 2.01.1 is generally available. Aurora MySQL 2.x versions are compatible with MySQL 5.7 or Aurora MySQL 1.x versions are compatible with MySQL 5.6.

When creating a new Aurora MySQL DB cluster, you can choose compatibility with either MySQL 5.7 or MySQL 5.6. When restoring a MySQL 5.6-compatible snapshot, you can choose compatibility with either MySQL 5.7 or MySQL 5.6.

You can restore snapshots of Aurora MySQL 1.14*, 1.15*, 1.16*, and 1.17* into Aurora MySQL 2.01.1.

We don't allow in-place upgrade of Aurora MySQL 1.x clusters into Aurora MySQL 2.01.1 or restore to Aurora MySQL 2.01.1 from an Amazon S3 backup. We plan to remove these restrictions in a later Aurora MySQL 2.x release.

The performance schema is disabled for this release of Aurora MySQL 5.7. Upgrade to Aurora 2.03 for performance schema support.

Comparison with Aurora MySQL 5.6

The following Amazon Aurora MySQL features are supported in Aurora MySQL 5.6, but these features are currently not supported in Aurora MySQL 5.7.

- Asynchronous key prefetch (AKP). For more information, see [Working with Asynchronous Key Prefetch in Amazon Aurora \(p. 765\)](#).
- Hash joins. For more information, see [Working with Hash Joins in Aurora MySQL \(p. 771\)](#).
- Native functions for synchronously invoking AWS Lambda functions. For more information, see [Invoking a Lambda Function with an Aurora MySQL Native Function \(p. 753\)](#).
- Scan batching. For more information, see [Aurora MySQL Database Engine Updates 2017-12-11 \(p. 846\)](#).
- Migrating data from MySQL using an Amazon S3 bucket. For more information, see [Migrating Data from MySQL by Using an Amazon S3 Bucket \(p. 590\)](#).

Currently, Aurora MySQL 2.01.1 does not support features added in Aurora MySQL version 1.16 and later. For information about Aurora MySQL version 1.16, see [Aurora MySQL Database Engine Updates 2017-12-11 \(p. 846\)](#).

MySQL 5.7 compatibility

Aurora MySQL 2.01.1 is wire-compatible with MySQL 5.7 and includes features such as JSON support, spatial indexes, and generated columns. Aurora MySQL uses a native implementation of spatial indexing using z-order curves to deliver >20x better write performance and >10x better read performance than MySQL 5.7 for spatial datasets.

Aurora MySQL 2.01.1 does not currently support the following MySQL 5.7 features:

- Global transaction identifiers (GTIDs). Aurora MySQL supports GTIDs in version 2.04 and higher.
- Group replication plugin
- Increased page size
- InnoDB buffer pool loading at startup
- InnoDB full-text parser plugin
- Multisource replication
- Online buffer pool resizing
- Password validation plugin
- Query rewrite plugins
- Replication filtering
- The `CREATE TABLESPACE` SQL statement
- X Protocol

CLI differences between Aurora MySQL 2.x and Aurora MySQL 1.x

- The engine name for Aurora MySQL 2.x is `aurora-mysql`; the engine name for Aurora MySQL 1.x continues to be `aurora`.
- The engine version for Aurora MySQL 2.x is `5.7.12`; the engine version for Aurora MySQL 1.x continues to be `5.6.10ann`.
- The default parameter group for Aurora MySQL 2.x is `default.aurora-mysql5.7`; the default parameter group for Aurora MySQL 1.x continues to be `default.aurora5.6`.
- The DB cluster parameter group family name for Aurora MySQL 2.x is `aurora-mysql5.7`; the DB cluster parameter group family name for Aurora MySQL 1.x continues to be `aurora5.6`.

Refer to the Aurora documentation for the full set of CLI commands and differences between Aurora MySQL 2.x and Aurora MySQL 1.x.

Improvements

- Fixed an issue with snapshot restore where Aurora-specific database privileges were created incorrectly when a MySQL 5.6-compatible snapshot was restored with MySQL 5.7 compatibility.
- Added support for 1.17 snapshot restores.

Aurora MySQL Database Engine Updates 2018-02-06

Version: 2.01

Aurora MySQL 2.01 is generally available. Going forward, Aurora MySQL 2.x versions will be compatible with MySQL 5.7 and Aurora MySQL 1.x versions will be compatible with MySQL 5.6.

When creating a new Aurora MySQL DB cluster, including those restored from snapshots, you can choose compatibility with either MySQL 5.7 or MySQL 5.6.

You can restore snapshots of Aurora MySQL 1.14*, 1.15*, and 1.16* into Aurora MySQL 2.01.

We don't allow in-place upgrade of Aurora MySQL 1.x clusters into Aurora MySQL 2.01 or restore to Aurora MySQL 2.01 from an Amazon S3 backup. We plan to remove these restrictions in a later Aurora MySQL 2.x release.

The performance schema is disabled for this release of Aurora MySQL 5.7. Upgrade to Aurora 2.03 for performance schema support.

Comparison with Aurora MySQL 5.6

The following Amazon Aurora MySQL features are supported in Aurora MySQL 5.6, but these features are currently not supported in Aurora MySQL 5.7.

- Asynchronous key prefetch (AKP). For more information, see [Working with Asynchronous Key Prefetch in Amazon Aurora \(p. 765\)](#).
- Hash joins. For more information, see [Working with Hash Joins in Aurora MySQL \(p. 771\)](#).
- Native functions for synchronously invoking AWS Lambda functions. For more information, see [Invoking a Lambda Function with an Aurora MySQL Native Function \(p. 753\)](#).
- Scan batching. For more information, see [Aurora MySQL Database Engine Updates 2017-12-11 \(p. 846\)](#).
- Migrating data from MySQL using an Amazon S3 bucket. For more information, see [Migrating Data from MySQL by Using an Amazon S3 Bucket \(p. 590\)](#).

Currently, Aurora MySQL 2.01 does not support features added in Aurora MySQL version 1.16 and later. For information about Aurora MySQL version 1.16, see [Aurora MySQL Database Engine Updates 2017-12-11 \(p. 846\)](#).

MySQL 5.7 compatibility

Aurora MySQL 2.01 is wire-compatible with MySQL 5.7 and includes features such as JSON support, spatial indexes, and generated columns. Aurora MySQL uses a native implementation of spatial indexing using z-order curves to deliver >20x better write performance and >10x better read performance than MySQL 5.7 for spatial datasets.

Aurora MySQL 2.01 does not currently support the following MySQL 5.7 features:

- Global transaction identifiers (GTIDs). Aurora MySQL supports GTIDs in version 2.04 and higher.
- Group replication plugin
- Increased page size
- InnoDB buffer pool loading at startup
- InnoDB full-text parser plugin
- Multisource replication
- Online buffer pool resizing
- Password validation plugin
- Query rewrite plugins
- Replication filtering
- The CREATE TABLESPACE SQL statement
- X Protocol

CLI differences between Aurora MySQL 2.x and Aurora MySQL 1.x

- The engine name for Aurora MySQL 2.x is `aurora-mysql`; the engine name for Aurora MySQL 1.x continues to be `aurora`.
- The engine version for Aurora MySQL 2.x is `5.7.12`; the engine version for Aurora MySQL 1.x continues to be `5.6.10ann`.
- The default parameter group for Aurora MySQL 2.x is `default.aurora-mysql5.7`; the default parameter group for Aurora MySQL 1.x continues to be `default.aurora5.6`.
- The DB cluster parameter group family name for Aurora MySQL 2.x is `aurora-mysql5.7`; the DB cluster parameter group family name for Aurora MySQL 1.x continues to be `aurora5.6`.

Refer to the Aurora documentation for the full set of CLI commands and differences between Aurora MySQL 2.x and Aurora MySQL 1.x.

Database Engine Updates for Amazon Aurora MySQL 1.1

The following are Amazon Aurora 1.1 database engine updates:

- [Aurora MySQL Database Engine Updates 2019-11-25 \(Version 1.22.0\) \(p. 831\)](#)
- [Aurora MySQL Database Engine Updates 2019-11-25 \(Version 1.21.0\) \(p. 833\)](#)
- [Aurora MySQL Database Engine Updates 2019-11-11 \(Version 1.20.0\) \(p. 835\)](#)
- [Aurora MySQL Database Engine Updates 2019-09-19 \(Version 1.19.5\) \(p. 836\)](#)
- [Aurora MySQL Database Engine Updates 2019-06-05 \(Version 1.19.2\) \(p. 837\)](#)
- [Aurora MySQL Database Engine Updates 2019-05-09 \(Version 1.19.1\) \(p. 838\)](#)
- [Aurora MySQL Database Engine Updates 2019-02-07 \(Version 1.19.0\) \(p. 838\)](#)
- [Aurora MySQL Database Engine Updates 2018-09-20 \(p. 839\) \(Version 1.18.0\)](#)
- [Aurora MySQL Database Engine Updates 2019-01-17 \(p. 840\) \(Version 1.17.8\)](#)
- [Aurora MySQL Database Engine Updates 2018-10-08 \(p. 841\) \(Version 1.17.7\)](#)
- [Aurora MySQL Database Engine Updates 2018-09-06 \(p. 842\) \(Version 1.17.6\)](#)
- [Aurora MySQL Database Engine Updates 2018-08-14 \(p. 842\) \(Version 1.17.5\)](#)
- [Aurora MySQL Database Engine Updates 2018-08-07 \(p. 843\) \(Version 1.17.4\)](#)
- [Aurora MySQL Database Engine Updates 2018-06-05 \(p. 844\) \(Version 1.17.3\)](#)
- [Aurora MySQL Database Engine Updates 2018-04-27 \(p. 844\) \(Version 1.17.2\)](#)
- [Aurora MySQL Database Engine Updates 2018-03-23 \(p. 845\) \(Version 1.17.1\)](#)
- [Aurora MySQL Database Engine Updates 2018-03-13 \(p. 845\) \(Version 1.17\)](#)
- [Aurora MySQL Database Engine Updates 2017-12-11 \(p. 846\) \(Version 1.16\)](#)
- [Aurora MySQL Database Engine Updates 2017-11-20 \(p. 847\) \(Version 1.15.1\)](#)
- [Aurora MySQL Database Engine Updates 2017-10-24 \(p. 848\) \(Version 1.15\)](#)
- [Aurora MySQL Database Engine Updates: 2018-03-13 \(p. 850\) \(Version 1.14.4\)](#)
- [Aurora MySQL Database Engine Updates: 2017-09-22 \(p. 850\) \(Version 1.14.1\)](#)
- [Aurora MySQL Database Engine Updates: 2017-08-07 \(p. 851\) \(Version 1.14\)](#)
- [Aurora MySQL Database Engine Updates: 2017-05-15 \(p. 852\) \(Version 1.13\)](#)
- [Aurora MySQL Database Engine Updates: 2017-04-05 \(p. 853\) \(Version 1.12\)](#)
- [Aurora MySQL Database Engine Updates: 2017-02-23 \(p. 855\) \(Version 1.11\)](#)
- [Aurora MySQL Database Engine Updates: 2017-01-12 \(p. 857\) \(Version 1.10.1\)](#)
- [Aurora MySQL Database Engine Updates: 2016-12-14 \(p. 857\) \(Version 1.10\)](#)
- [Aurora MySQL Database Engine Updates: 2016-11-10 \(p. 858\) \(Versions 1.9.0, 1.9.1\)](#)
- [Aurora MySQL Database Engine Updates: 2016-10-26 \(p. 859\) \(Version 1.8.1\)](#)
- [Aurora MySQL Database Engine Updates: 2016-10-18 \(p. 859\) \(Version 1.8\)](#)
- [Aurora MySQL Database Engine Updates: 2016-09-20 \(p. 861\) \(Version 1.7.1\)](#)
- [Aurora MySQL Database Engine Updates: 2016-08-30 \(p. 861\) \(Version 1.7\)](#)
- [Aurora MySQL Database Engine Updates: 2016-06-01 \(p. 862\) \(Version 1.6.5\)](#)
- [Aurora MySQL Database Engine Updates: 2016-04-06 \(p. 862\) \(Version 1.6\)](#)
- [Aurora MySQL Database Engine Updates: 2016-01-11 \(p. 864\) \(Version 1.5\)](#)
- [Aurora MySQL Database Engine Updates: 2015-12-03 \(p. 864\) \(Version 1.4\)](#)
- [Aurora MySQL Database Engine Updates: 2015-10-16 \(p. 866\) \(Versions 1.2, 1.3\)](#)

- [Aurora MySQL Database Engine Updates: 2015-08-24 \(p. 868\)](#) (Version 1.1)

Aurora MySQL Database Engine Updates 2019-11-25 (Version 1.22.0)

Version: 1.22.0

Aurora MySQL 1.22.0 is generally available. Aurora MySQL 1.* versions are compatible with MySQL 5.6 and Aurora MySQL 2.* versions are compatible with MySQL 5.7.

Currently supported Aurora MySQL releases are 1.14.*, 1.15.*, 1.16.*, 1.17.*, 1.18.*, 1.19.*, 1.20.*, 1.21.*, 1.22.*, 2.01.*, 2.02.*, 2.03.*, 2.04.*, 2.05.*, 2.06.*, and 2.07.*. To create a cluster with an older version of Aurora MySQL, please specify the engine version through the AWS Management Console, the AWS CLI or the RDS API. You have the option to upgrade existing Aurora MySQL 1.* database clusters to Aurora MySQL 1.22.0.

Note

This version is currently not available in the following AWS Regions: AWS GovCloud (US-East) [us-gov-east-1], AWS GovCloud (US-West) [us-gov-west-1], China (Ningxia) [cn-northwest-1], Asia Pacific (Hong Kong) [ap-east-1], Middle East (Bahrain) [me-south-1], and South America (São Paulo) [sa-east-1]. There will be a separate announcement once it is made available.

If you have any questions or concerns, AWS Support is available on the community forums and through [AWS Premium Support](#). For more information, see [Maintaining an Amazon Aurora DB Cluster \(p. 408\)](#).

Note

The procedure to upgrade your DB cluster has changed. For more information, see [Database Upgrades and Patches for Amazon Aurora MySQL \(p. 795\)](#).

Improvements

New features:

- Aurora MySQL clusters now support the instance types r5.8xlarge, r5.16xlarge and r5.24xlarge.
- Binlog has new enhancements for improved commit time latency when very large transactions are involved.
- Aurora MySQL now has a mechanism to minimize the time window during which events of a large transaction are written to binlog on commit. This effectively prevents lengthy offline recovery incurred when database crashes occur during that time window. This feature also fixes the issue where a large transaction blocks small transactions on binlog commit. This feature is off by default and can be enabled by the service team if needed for your workload. When enabled, it will be triggered when a transaction size is > 500MB.
- Added support for the ANSI READ COMMITTED isolation level on the read replicas. This isolation level enables long-running queries on the read replica to execute without impacting the high throughput of writes on the writer node. For more information, see [Aurora MySQL Isolation Levels](#).
- Global Databases now allow adding secondary read-only replica regions for database clusters deployed in these AWS Regions: regions: US East (N. Virginia) [us-east-1], US East (Ohio) [us-east-2], US West (N. California) [us-west-1], US West (Oregon) [us-west-2], Europe (Ireland) [eu-west-1], Europe (London) [eu-west-2], Europe (Paris) [eu-west-3], Asia Pacific (Tokyo) [ap-northeast-1], Asia Pacific (Seoul) [ap-northeast-2], Asia Pacific (Singapore) [ap-southeast-1], Asia Pacific (Sydney) [ap-southeast-2], Canada (Central) [ca-central-1], Europe (Frankfurt) [eu-central-1], and Asia Pacific (Mumbai) [ap-south-1].
- The hot row contention feature is now generally available and does not require the Aurora lab mode setting to be ON. This feature substantially improves throughput for workloads with many transactions contending for rows on the same page.
- This version has updated timezone files to support the latest Brazil timezone update for new clusters.

Critical fixes:

- [CVE-2019-2922](#)
- [CVE-2019-2923](#)
- [CVE-2019-2924](#)
- [CVE-2019-2910](#)

High priority fixes:

- [CVE-2019-2805](#)
- [CVE-2019-2730](#)
- [CVE-2019-2740](#)
- [CVE-2018-3064](#)
- [CVE-2018-3058](#)
- [CVE-2017-3653](#)
- [CVE-2017-3464](#)
- [CVE-2017-3244](#)
- [CVE-2016-5612](#)
- [CVE-2016-5439](#)
- [CVE-2016-0606](#)
- [CVE-2015-4904](#)
- [CVE-2015-4879](#)
- [CVE-2015-4864](#)
- [CVE-2015-4830](#)
- [CVE-2015-4826](#)
- [CVE-2015-2620](#)
- [CVE-2015-0382](#)
- [CVE-2015-0381](#)
- [CVE-2014-6555](#)
- [CVE-2014-4258](#)
- [CVE-2014-4260](#)
- [CVE-2014-2444](#)
- [CVE-2014-2436](#)
- [CVE-2013-5881](#)
- [CVE-2014-0393](#)
- [CVE-2013-5908](#)
- [CVE-2013-5807](#)
- [CVE-2013-3806](#)
- [CVE-2013-3811](#)
- [CVE-2013-3804](#)
- [CVE-2013-3807](#)
- [CVE-2013-2378](#)
- [CVE-2013-2375](#)
- [CVE-2013-1523](#)
- [CVE-2013-2381](#)
- [CVE-2012-5615](#)
- [CVE-2014-6489](#)

- Fixed an issue in the DDL recovery component that resulted in prolonged database downtime. Clusters that become unavailable after executing `TRUNCATE TABLE` query on a table with an `AUTO_INCREMENT` column should be updated.
- Fixed an issue in the DDL recovery component that resulted in prolonged database downtime. Clusters that become unavailable after executing `DROP TABLE` query on multiple tables in parallel should be updated.

General stability fixes:

- Fixed an issue that caused read replicas to restart during a long-running transaction. Customers who encounter replica restarts that coincide with an accelerated drop in freeable memory should consider upgrading to this version.
- Fixed an issue that incorrectly reported `ERROR 1836` when a nested query is executed against a temporary table on the read replica.
- Fixed a parallel query abort error on an Aurora reader instance while a heavy write workload is running on the Aurora writer instance.
- Fixed an issue that causes a database configured as a Binlog Master to restart while a heavy write workload is running.
- Fixed an issue of prolonged unavailability while restarting the engine. This addresses an issue in the buffer pool initialization. This issue occurs rarely but can potentially impact any supported release.
- Fixed an issue that generated inconsistent data in the `information_schema.replica_host_status` table.
- Fixed a race condition between the parallel query and the standard execution paths that caused the Reader nodes to restart intermittently.
- Improved stability of the database when the number of client connections exceeds the `max_connections` parameter value.
- Improved stability of the reader instances by blocking unsupported DDL and `LOAD FROM S3` queries.

Integration of MySQL community edition bug fixes

- Bug#16346241 - SERVER CRASH IN ITEM_PARAM::QUERY_VAL_STR
- Bug#17733850 - NAME_CONST() CRASH IN ITEM_NAME_CONST::ITEM_NAME_CONST()
- Bug #20989615 - INNODB AUTO_INCREMENT PRODUCES SAME VALUE TWICE
- Bug #20181776 - ACCESS CONTROL DOESN'T MATCH MOST SPECIFIC HOST WHEN IT CONTAINS WILDCARD
- Bug #27326796 - MYSQL CRASH WITH INNODB ASSERTION FAILURE IN FILE PARSOPARS.CC
- Bug #20590013 - IF YOU HAVE A FULLTEXT INDEX AND DROP IT YOU CAN NO LONGER PERFORM ONLINE DDL

Aurora MySQL Database Engine Updates 2019-11-25 (Version 1.21.0)

Version: 1.21.0

Aurora MySQL 1.21.0 is generally available. Aurora MySQL 1.* versions are compatible with MySQL 5.6 and Aurora MySQL 2.* versions are compatible with MySQL 5.7.

Currently supported Aurora MySQL releases are 1.14.*, 1.15.*, 1.16.*, 1.17.*, 1.18.*, 1.19.*, 1.20.*, 1.21.*, 2.01.*, 2.02.*, 2.03.* and 2.04.*. To create a cluster with an older version of Aurora MySQL, please specify the engine version through the AWS Management Console, the AWS CLI or the RDS API. You have the option to upgrade existing Aurora MySQL 1.* database clusters to Aurora MySQL 1.21.0.

Note

This version is currently not available in the following AWS Regions: AWS GovCloud (US-East) [us-gov-east-1], AWS GovCloud (US-West) [us-gov-west-1], China (Ningxia) [cn-northwest-1], Asia Pacific (Hong Kong) [ap-east-1], Europe (Stockholm) [eu-north-1], and Middle East (Bahrain) [me-south-1]. There will be a separate announcement once it is made available.

If you have any questions or concerns, AWS Support is available on the community forums and through [AWS Premium Support](#). For more information, see [Maintaining an Amazon Aurora DB Cluster \(p. 408\)](#).

Note

The procedure to upgrade your DB cluster has changed. For more information, see [Database Upgrades and Patches for Amazon Aurora MySQL \(p. 795\)](#).

Improvements

Critical fixes:

- [CVE-2018-0734](#)
- [CVE-2019-2534](#)
- [CVE-2018-2612](#)
- [CVE-2017-3599](#)
- [CVE-2018-2562](#)
- [CVE-2017-3329](#)
- [CVE-2018-2696](#)
- [CVE-2015-4737](#)

High priority fixes:

- Customers with database size close to 64 TiB are strongly advised to upgrade to this version to avoid downtime due to stability bugs affecting volumes close to the Aurora storage limit.

General stability fixes:

- Fixed a parallel query abort error on Aurora reader instances while a heavy write workload is running on the Aurora writer instance.
- Fixed an issue on Aurora reader instances that reduced free memory during long-running transactions while there is a heavy transaction commit traffic on the writer instance.
- The value of the parameter `aurora_disable_hash_join` is now persisted after database restart or host replacement.
- Fixed an issue related to the Full Text Search cache that caused the Aurora instance to run out of memory. Customers using Full Text Search should upgrade.
- Improved stability of the database when the hash join feature is enabled and the instance is low on memory. Customers using hash join should upgrade.
- Fixed an issue in the query cache where the "Too many connections" error could cause a reboot.
- Fixed the free memory calculation on T2 instances to include swap memory space to prevent unnecessary reboots.

Integration of MySQL community edition bug fixes

- Bug #19929406: `HANDLE_FATAL_SIGNAL (SIG=11) IN __MEMMOVE_SSSE3_BACK FROM STRING::COPY`
- Bug #17059925: For `UNION` statements, the rows-examined value was calculated incorrectly. This was manifested as too-large values for the `ROWS_EXAMINED` column of Performance Schema statement tables (such as `events_statements_current`).

- Bug #11827369: Some queries with `SELECT ... FROM DUAL` nested subqueries raised an assertion.
- Bug #16311231: Incorrect results were returned if a query contained a subquery in an `IN` clause that contained an `XOR` operation in the `WHERE` clause.

Aurora MySQL Database Engine Updates 2019-11-11 (Version 1.20.0)

Version: 1.20.0

Aurora MySQL 1.20.0 is generally available. Aurora MySQL 1.* versions are compatible with MySQL 5.6 and Aurora MySQL 2.* versions are compatible with MySQL 5.7.

Currently supported Aurora MySQL releases are 1.14.*, 1.15.*, 1.16.*, 1.17.*, 1.18.*, 1.19.*, 1.20.*, 2.01.*, 2.02.*, 2.03.* and 2.04.*. To create a cluster with an older version of Aurora MySQL, please specify the engine version through the AWS Management Console, the AWS CLI or the RDS API. You have the option to upgrade existing Aurora MySQL 1.* database clusters to Aurora MySQL 1.20.0.

Note

This version is currently not available in the following AWS Regions: AWS GovCloud (US-East) [us-gov-east-1], AWS GovCloud (US-West) [us-gov-west-1], China (Ningxia) [cn-northwest-1], Asia Pacific (Hong Kong) [ap-east-1], Europe (Stockholm) [eu-north-1], and Middle East (Bahrain) [me-south-1]. There will be a separate announcement once it is made available.

If you have any questions or concerns, AWS Support is available on the community forums and through [AWS Premium Support](#). For more information, see [Maintaining an Amazon Aurora DB Cluster \(p. 408\)](#).

Note

The procedure to upgrade your DB cluster has changed. For more information, see [Database Upgrades and Patches for Amazon Aurora MySQL \(p. 795\)](#).

Improvements

Critical fixes:

- [CVE-2018-0734](#)
- [CVE-2019-2534](#)
- [CVE-2018-2612](#)
- [CVE-2017-3599](#)
- [CVE-2018-2562](#)
- [CVE-2017-3329](#)
- [CVE-2018-2696](#)
- [CVE-2015-4737](#)

High priority fixes:

- Customers with database size close to 64 TiB are strongly advised to upgrade to this version to avoid downtime due to stability bugs affecting volumes close to the Aurora storage limit.

General stability fixes:

- Fixed a parallel query abort error on Aurora reader instances while a heavy write workload is running on the Aurora writer instance.
- Fixed an issue on Aurora reader instances that reduced free memory during long-running transactions while there is a heavy transaction commit traffic on the writer instance.

- The value of the parameter `aurora_disable_hash_join` is now persisted after database restart or host replacement.
- Fixed an issue related to the Full Text Search cache that caused the Aurora instance to run out of memory. Customers using Full Text Search should upgrade.
- Improved stability of the database when the hash join feature is enabled and the instance is low on memory. Customers using hash join should upgrade.
- Fixed an issue in the query cache where the "Too many connections" error could cause a reboot.
- Fixed the free memory calculation on T2 instances to include swap memory space to prevent unnecessary reboots.

Integration of MySQL community edition bug fixes

- Bug #19929406: `HANDLE_FATAL_SIGNAL (SIG=11) IN __MEMMOVE_SSSE3_BACK FROM STRING::COPY`
- Bug #17059925: For `UNION` statements, the `rows-examined` value was calculated incorrectly. This was manifested as too-large values for the `ROWS_EXAMINED` column of Performance Schema statement tables (such as `events_statements_current`).
- Bug #11827369: Some queries with `SELECT ... FROM DUAL` nested subqueries raised an assertion.
- Bug #16311231: Incorrect results were returned if a query contained a subquery in an `IN` clause that contained an `XOR` operation in the `WHERE` clause.

Aurora MySQL Database Engine Updates 2019-09-19 (Version 1.19.5)

Version: 1.19.5

Aurora MySQL 1.19.5 is generally available. Aurora MySQL 1.* versions are compatible with MySQL 5.6 and Aurora MySQL 2.* versions are compatible with MySQL 5.7.

You have the option to upgrade existing database clusters to Aurora MySQL 1.19.5. You can restore snapshots of Aurora MySQL 1.14.*, 1.15.*, 1.16.*, 1.17.*, 1.18.*, 1.19.1, and 1.19.2 into Aurora MySQL 1.19.5.

To use an older version of Aurora MySQL, you can create new database clusters by specifying the engine version through the AWS Management Console, the AWS CLI, or the RDS API.

If you have any questions or concerns, AWS Support is available on the community forums and through [AWS Premium Support](#). For more information, see [Maintaining an Amazon Aurora DB Cluster \(p. 408\)](#).

Note

This version is currently not available in the following AWS Regions: Europe (London) [eu-west-2], AWS GovCloud (US-East) [us-gov-east-1], AWS GovCloud (US-West) [us-gov-west-1], China (Ningxia) [cn-northwest-1], and Asia Pacific (Hong Kong) [ap-east-1]. There will be a separate announcement once it is made available.

Note

The procedure to upgrade your DB cluster has changed. For more information, see [Database Upgrades and Patches for Amazon Aurora MySQL \(p. 795\)](#).

Improvements

- Fixed an issue on Aurora reader instances that reduced free memory during long-running transactions while there is a heavy transaction commit traffic on the writer instance.
- Fixed a parallel query abort error on Aurora reader instances while a heavy write workload is running on the Aurora writer instance.

- The value of the parameter `aurora_disable_hash_join` is now persisted after database restart or host replacement.
- Fixed an issue related to the Full Text Search cache that caused the Aurora instance to run out of memory.
- Improved stability of the database when the volume size is close to the 64 TiB volume limit by reserving 160 GB of space for the recovery workflow to complete without a failover.
- Improved stability of the database when the hash join feature is enabled and the instance is low on memory.
- Fixed the free memory calculation to include swap memory space on T2 instances that caused them to reboot prematurely.
- Fixed an issue in the query cache where the "Too many connections" error could cause a reboot.

Integration of MySQL community edition bug fixes

- [CVE-2018-2696](#)
- [CVE-2015-4737](#)
- Bug #19929406: `HANDLE_FATAL_SIGNAL (SIG=11) IN __MEMMOVE_SSSE3_BACK FROM STRING::COPY`
- Bug #17059925: For `UNION` statements, the `rows_examined` value was calculated incorrectly. This was manifested as too-large values for the `ROWS_EXAMINED` column of Performance Schema statement tables (such as `events_statements_current`).
- Bug #11827369: Some queries with `SELECT ... FROM DUAL` nested subqueries raised an assertion.
- Bug #16311231: Incorrect results were returned if a query contained a subquery in an `IN` clause that contained an `XOR` operation in the `WHERE` clause.

Aurora MySQL Database Engine Updates 2019-06-05 (Version 1.19.2)

Version: 1.19.2

Aurora MySQL 1.19.2 is generally available. All new Aurora MySQL database clusters with MySQL 5.6 compatibility, including those restored from snapshots, can be created with 1.17.8, 1.19.0, 1.19.1, or 1.19.2. You have the option, but are not required, to upgrade existing database clusters to Aurora MySQL 1.19.2. To use an older version, you can create new database clusters in Aurora MySQL 1.14.4, Aurora MySQL 1.15.1, Aurora MySQL 1.16, Aurora MySQL 1.17.8, or Aurora MySQL 1.18. You can do so using the AWS CLI or the Amazon RDS API and specifying the engine version.

If you have any questions or concerns, AWS Support is available on the community forums and through [AWS Premium Support](#). For more information, see [Maintaining an Amazon Aurora DB Cluster \(p. 408\)](#).

Note

This version is currently not available in the AWS GovCloud (US-West) [us-gov-west-1], Europe (Stockholm) [eu-north-1], China (Ningxia) [cn-northwest-1], and Asia Pacific (Hong Kong) [ap-east-1] AWS Regions. There will be a separate announcement once it is made available.

Note

The procedure to upgrade your DB cluster has changed. For more information, see [Database Upgrades and Patches for Amazon Aurora MySQL \(p. 795\)](#).

Improvements

- Fixed an issue that could cause failures when loading data into Aurora from Amazon S3.
- Fixed an issue that could cause failures when uploading data from Aurora to Amazon S3.
- Fixed an issue that created zombie sessions left in a killed state.

- Fixed an issue that caused aborted connections when handling an error in network protocol management.
- Fixed an issue that could cause a crash when dealing with partitioned tables.
- Fixed an issue related to binlog replication of trigger creation.

Aurora MySQL Database Engine Updates 2019-05-09 (Version 1.19.1)

Version: 1.19.1

Aurora MySQL 1.19.1 is generally available. All new Aurora MySQL database clusters with MySQL 5.6 compatibility, including those restored from snapshots, can be created with 1.17.8, 1.19.0, or 1.19.1. You have the option, but are not required, to upgrade existing database clusters to Aurora MySQL 1.19.1. To use an older version, you can create new database clusters in Aurora MySQL 1.14.4, Aurora MySQL 1.15.1, Aurora MySQL 1.16, Aurora MySQL 1.17.8, or Aurora MySQL 1.18. You can do so using the AWS CLI or the Amazon RDS API and specifying the engine version.

If you have any questions or concerns, AWS Support is available on the community forums and through [AWS Premium Support](#). For more information, see [Maintaining an Amazon Aurora DB Cluster \(p. 408\)](#).

Note

This version is currently not available in the AWS GovCloud (US-West) [us-gov-west-1] and China (Beijing) [cn-north-1] regions. There will be a separate announcement once it is made available.

Note

The procedure to upgrade your DB cluster has changed. For more information, see [Database Upgrades and Patches for Amazon Aurora MySQL \(p. 795\)](#).

Improvements

- Fixed a bug in binlog replication that can cause an issue on Aurora instances configured as binlog slave.
- Fixed an error in handling certain kinds of `ALTER TABLE` commands.
- Fixed an issue with aborted connections because of an error in network protocol management.

Aurora MySQL Database Engine Updates 2019-02-07 (Version 1.19.0)

Version: 1.19.0

Aurora MySQL 1.19.0 is generally available. All new Aurora MySQL database clusters with MySQL 5.6 compatibility, including those restored from snapshots, can be created with 1.17.8 or 1.19.0. You have the option, but are not required, to upgrade existing database clusters to Aurora MySQL 1.19.0. To use an older version, you can create new database clusters in Aurora MySQL 1.14.4, Aurora MySQL 1.15.1, Aurora MySQL 1.16, Aurora MySQL 1.17.8, or Aurora MySQL 1.18.0. You can do so using the AWS CLI or the Amazon RDS API and specifying the engine version.

If you have any questions or concerns, AWS Support is available on the community forums and through [AWS Premium Support](#). For more information, see [Maintaining an Amazon Aurora DB Cluster \(p. 408\)](#).

Note

This version is currently not available in the AWS GovCloud (US-West) [us-gov-west-1] and China (Beijing) [cn-north-1] regions. There will be a separate announcement once it is made available.

Note

The procedure to upgrade your DB cluster has changed. For more information, see [Database Upgrades and Patches for Amazon Aurora MySQL \(p. 795\)](#).

Features

- **Aurora Version Selector** - Starting with Aurora MySQL 1.19.0, you can choose from among multiple versions of MySQL 5.6 compatible Aurora on the Amazon RDS console. For more information, see [Aurora MySQL Engine Versions \(p. 794\)](#).

Improvements

- Fixed a stability issue related to the `CHECK TABLE` query on an Aurora Replica.
- Introduced a new global user variable `aurora_disable_hash_join` to disable Hash Join.
- Fixed a stability issue when generating the output row during multiple table hash join.
- Fixed an issue that returned a wrong result because of a plan change during Hash Join applicability check.
- Zero Downtime Patching is supported with long running transactions. This enhancement will come into effect when upgrading from version 1.19 to a higher one.
- Zero Downtime Patching is now supported when binlog is enabled. This enhancement will come into effect when upgrading from version 1.19 to a higher one.
- Fixed an issue that caused a spike in CPU utilization on the Aurora Replica unrelated to the workload.
- Fixed a race condition in the lock manager that resulted in a database restart.
- Fixed a race condition in the lock manager component to improve stability of Aurora instances.
- Improved stability of the deadlock detector inside the lock manager component.
- `INSERT` operation on a table is prohibited if InnoDB detects that the index is corrupted.
- Fixed a stability issue in Fast DDL.
- Improved Aurora stability by reducing the memory consumption in scan batching for single-row subquery.
- Fixed a stability issue that occurred after a foreign key was dropped while the system variable `foreign_key_checks` is set to 0.
- Fixed an issue in the Out Of Memory Avoidance feature that erroneously overrode changes to the `table_definition_cache` value made by the user.
- Fixed stability issues in the Out Of Memory Avoidance feature.
- Fixed an issue that set `query_time` and `lock_time` in `slow_query_log` to garbage values.
- Fixed a parallel query stability issue triggered by improper handling of string collation internally.
- Fixed a parallel query stability issue triggered by a secondary index search.
- Fixed a parallel query stability issue triggered by a multi-table update.

Integration of MySQL community edition bug fixes

- BUG #32917: DETECT ORPHAN TEMP-POOL FILES, AND HANDLE GRACEFULLY
- BUG #63144 CREATE TABLE IF NOT EXISTS METADATA LOCK IS TOO RESTRICTIVE

Aurora MySQL Database Engine Updates 2018-09-20

Version: 1.18.0

Aurora MySQL 1.18.0 is generally available. All new Aurora MySQL database clusters with MySQL 5.6 compatibility, including those restored from snapshots, will be created in Aurora MySQL 1.18.0. You have the option, but are not required, to upgrade existing database clusters to Aurora MySQL 1.18.0. You can create new DB clusters in Aurora MySQL 1.14.4, Aurora MySQL 1.15.1, Aurora MySQL 1.16, or Aurora MySQL 1.17.6. You can do so using the AWS CLI or the Amazon RDS API and specifying the engine version.

With version 1.18.0 of Aurora MySQL, we are using a cluster patching model where all nodes in an Aurora DB cluster are patched at the same time.

If you have any questions or concerns, AWS Support is available on the community forums and through [AWS Premium Support](#). For more information, see [Maintaining an Amazon Aurora DB Cluster \(p. 408\)](#).

Features

- **Parallel Query** is available with this release, for new clusters and restored snapshots. Aurora MySQL parallel query is an optimization that parallelizes some of the I/O and computation involved in processing data-intensive queries. The work that is parallelized includes retrieving rows from storage, extracting column values, and determining which rows match the conditions in the `WHERE` clause and join clauses. This data-intensive work is delegated (in database optimization terms, pushed down) to multiple nodes in the Aurora distributed storage layer. Without parallel query, each query brings all the scanned data to a single node within the Aurora MySQL cluster (the head node) and performs all the query processing there.
 - When the parallel query feature is enabled, the Aurora MySQL engine automatically determines when queries can benefit, without requiring SQL changes such as hints or table attributes.

For more information, see [Working with Parallel Query for Amazon Aurora MySQL \(p. 644\)](#).

- **OOM Avoidance:** This feature monitors the system memory and tracks memory consumed by various components of the database. Once the system runs low on memory, it performs a list of actions to release memory from various tracked components in an attempt to save the database from running into Out of Memory (OOM) and thereby avoiding a database restart. This best-effort feature is enabled by default for t2 instances and can be enabled on other instance classes via a new instance parameter named `aurora_oom_response`. The instance parameter takes a string of comma separated actions that an instance should take when its memory is low. Valid actions include "print", "tune", "decline", "kill_query" or any combination of these. Any empty string means there should be no actions taken and effectively renders the feature to be disabled. Note that the default actions for the feature is "print, tune". Usage examples:

- "print" – Only prints the queries taking high amount of memory.
- "tune" – Tunes the internal table caches to release some memory back to the system.
- "decline" – Declines new queries once the instance is low on memory.
- "kill_query" – Kills the queries in descending order of memory consumption until the instance memory surfaces above the low threshold. Data definition language (DDL) statements are not killed.
- "print, tune" – Performs actions described for both "print" and "tune".
- "tune, decline, kill_query" – Performs the actions described for "tune", "decline", and "kill_query".

For information about handling out-of-memory conditions and other troubleshooting advice, see [Amazon Aurora MySQL Out of Memory Issues \(p. 1006\)](#).

Aurora MySQL Database Engine Updates 2019-01-17

Version: 1.17.8

Aurora MySQL 1.17.8 is generally available. All new Aurora MySQL database clusters with MySQL 5.6 compatibility, including those restored from snapshots, will be created in Aurora MySQL 1.17.8. You have the option, but are not required, to upgrade existing database clusters to Aurora MySQL 1.17.8. To use

an older version, you can create new database clusters in Aurora MySQL 1.14.4, 1.15.1, 1.16, or 1.17.7. You can do so using the AWS CLI or the Amazon RDS API and specifying the engine version.

With version 1.17.8 of Aurora MySQL, we are using a cluster patching model where all nodes in an Aurora DB cluster are patched at the same time.

Note

This version is currently not available in the AWS GovCloud (US-West) [us-gov-west-1] and China (Beijing) [cn-north-1] regions. There will be a separate announcement once it is made available.

If you have any questions or concerns, AWS Support is available on the community forums and through [AWS Premium Support](#). For more information, see [Maintaining an Amazon Aurora DB Cluster \(p. 408\)](#).

Improvements

- Fixed a performance issue that increased the CPU utilization on an Aurora Replica after a restart.
- Fixed a stability issue for `SELECT` queries that used hash join.

Integration of MySQL community edition bug fixes

- BUG #13418638: CREATE TABLE IF NOT EXISTS METADATA LOCK IS TOO RESTRICTIVE

Aurora MySQL Database Engine Updates 2018-10-08

Version: 1.17.7

Aurora MySQL 1.17.7 is generally available. All new Aurora MySQL database clusters with MySQL 5.6 compatibility, including those restored from snapshots, will be created in Aurora MySQL 1.17.7. You have the option, but are not required, to upgrade existing database clusters to Aurora MySQL 1.17.7. To use an older version, you can create new database clusters in Aurora MySQL 1.14.4, 1.15.1, 1.16, or 1.17.6. You can do so using the AWS CLI or the Amazon RDS API and specifying the engine version.

With version 1.17.7 of Aurora MySQL, we are using a cluster patching model where all nodes in an Aurora DB cluster are patched at the same time.

Note

This version is currently not available in the AWS GovCloud (US-West) [us-gov-west-1] and China (Beijing) [cn-north-1] regions. There will be a separate announcement once it is made available.

If you have any questions or concerns, AWS Support is available on the community forums and through [AWS Premium Support](#). For more information, see [Maintaining an Amazon Aurora DB Cluster \(p. 408\)](#).

Improvements

- The InnoDB status variable `innodb_buffer_pool_size` has been made publicly visible for the customers to modify.
- Fixed a stability issue on the Aurora cluster that occurred during failovers.
- Improved cluster availability by fixing a DDL recovery issue that occurred after an unsuccessful `TRUNCATE` operation.
- Fixed a stability issue related to the `mysql.innodb_table_stats` table update, triggered by DDL operations.
- Fixed Aurora Replica stability issues triggered during query cache invalidation after a DDL operation.
- Fixed a stability issue triggered by invalid memory access during periodic dictionary cache eviction in the background.

Integration of MySQL community edition bug fixes

- Bug #16208542: Drop index on a foreign key column leads to missing table.
- Bug #76349: memory leak in add_derived_key().
- Bug #16862316: For partitioned tables, queries could return different results depending on whether Index Merge was used.
- Bug #17588348: Queries using the index_merge optimization (see [Index Merge Optimization](#)) could return invalid results when run against tables that were partitioned by HASH.

Aurora MySQL Database Engine Updates 2018-09-06

Version: 1.17.6

Aurora MySQL 1.17.6 is generally available. All new Aurora MySQL database clusters with MySQL 5.6 compatibility, including those restored from snapshots, will be created in Aurora MySQL 1.17.6. You have the option, but are not required, to upgrade existing database clusters to Aurora MySQL 1.17.6. To use an older version, you can create new database clusters in Aurora MySQL 1.14.4, 1.15.1, 1.16, or 1.17.5. You can do so using the AWS CLI or the Amazon RDS API and specifying the engine version.

With version 1.17.6 of Aurora MySQL, we are using a cluster patching model where all nodes in an Aurora DB cluster are patched at the same time.

Note

This version is currently not available in the AWS GovCloud (US-West) [us-gov-west-1] and China (Beijing) [cn-north-1] regions. There will be a separate announcement once it is made available.

If you have any questions or concerns, AWS Support is available on the community forums and through [AWS Premium Support](#). For more information, see [Maintaining an Amazon Aurora DB Cluster \(p. 408\)](#).

Improvements

- Fixed a stability issue on the Aurora Reader for `SELECT` queries while the Aurora Writer is performing DDL operations on the same table.
- Fixed a stability issue caused by the creation and deletion of DDL logs for temporary tables that use Heap/Memory engine.
- Fixed a stability issue on the Binlog Slave when DDL statements are being replicated while the connection to the Binlog Master is unstable.
- Fixed a stability issue encountered while writing to the slow query log.
- Fixed an issue with the replica status table that exposed incorrect Aurora Reader lag information.

Integration of MySQL community edition bug fixes

- For an `ALTER TABLE` statement that renamed or changed the default value of a `BINARY` column, the alteration was done using a table copy and not in place. (Bug #67141, Bug #14735373, Bug #69580, Bug #17024290)
- An outer join between a regular table and a derived table that is implicitly groups could cause a server exit. (Bug #16177639)

Aurora MySQL Database Engine Updates 2018-08-14

Version: 1.17.5

Aurora MySQL 1.17.5 is generally available. All new Aurora MySQL database clusters with MySQL 5.6 compatibility, including those restored from snapshots, will be created in Aurora MySQL 1.17.5. You have the option, but are not required, to upgrade existing database clusters to Aurora MySQL 1.17.5. To use an older version, you can create new database clusters in Aurora MySQL 1.14.4, 1.15.1, 1.16, or 1.17.4. You can do so using the AWS CLI or the Amazon RDS API and specifying the engine version.

With version 1.17.5 of Aurora MySQL, we are using a cluster patching model where all nodes in an Aurora DB cluster are patched at the same time.

Note

This version is currently not available in the AWS GovCloud (US-West) [us-gov-west-1] and China (Beijing) [cn-north-1] regions. There will be a separate announcement once it is made available.

If you have any questions or concerns, AWS Support is available on the community forums and through [AWS Premium Support](#). For more information, see [Maintaining an Amazon Aurora DB Cluster \(p. 408\)](#).

Improvements

- Fixed an issue where an Aurora Writer might experience a restart after an Aurora cluster is patched using the Zero-Downtime Patching feature.

Aurora MySQL Database Engine Updates 2018-08-07

Version: 1.17.4

Aurora MySQL 1.17.4 is generally available. All new Aurora MySQL database clusters with MySQL 5.6 compatibility, including those restored from snapshots, will be created in Aurora MySQL 1.17.4. You have the option, but are not required, to upgrade existing database clusters to Aurora MySQL 1.17.4. To use an older version, you can create new database clusters in Aurora MySQL 1.14.4, 1.15.1, 1.16, or 1.17.3. You can do so using the AWS CLI or the Amazon RDS API and specifying the engine version.

With version 1.17.4 of Aurora MySQL, we are using a cluster patching model where all nodes in an Aurora DB cluster are patched at the same time.

Note

This version is currently not available in the AWS GovCloud (US-West) [us-gov-west-1] and China (Beijing) [cn-north-1] regions. There will be a separate announcement once it is made available.

If you have any questions or concerns, AWS Support is available on the community forums and through [AWS Premium Support](#). For more information, see [Maintaining an Amazon Aurora DB Cluster \(p. 408\)](#).

Improvements

- Replication improvements:
 - Reduced network traffic by not transmitting binlog records to cluster replicas. This improvement is enabled by default.
 - Reduced network traffic by compressing replication messages. This improvement is enabled by default for 8xlarge and 16xlarge instance classes. Such large instances can sustain a heavy volume of write traffic that results in substantial network traffic for replication messages.
- Fixes to the replica query cache.
- Fixed an issue where `ORDER BY LOWER(col_name)` could produce incorrect ordering while using the `utf8_bin` collation.
- Fixed an issue where DDL statements (especially `TRUNCATE TABLE`) could cause problems on Aurora replicas, including instability or missing tables.

- Fixed an issue where sockets are left in a half-open state when storage nodes are restarted.
- The following new DB cluster parameters are available:
 - `aurora_enable_zdr` – Allow connections opened on an Aurora Replica to stay active on replica restart.
 - `aurora_enable_replica_log_compression` – Enable compression of replication payloads to improve network bandwidth utilization between the master and Aurora Replicas.
 - `aurora_enable_repl_bin_log_filtering` – Enable filtering of replication records that are unusable by Aurora Replicas on the master.

Aurora MySQL Database Engine Updates 2018-06-05

Version: 1.17.3

Aurora MySQL 1.17.3 is generally available. All new Aurora MySQL database clusters with MySQL 5.6 compatibility, including those restored from snapshots, will be created in Aurora MySQL 1.17.3. You have the option, but are not required, to upgrade existing database clusters to Aurora MySQL 1.17.3. You can create new database clusters in Aurora MySQL 1.14.4, Aurora MySQL 1.15.1, or Aurora MySQL 1.16. You can do so using the AWS CLI or the Amazon RDS API and specifying the engine version.

With version 1.17.3 of Aurora MySQL, we are using a cluster patching model where all nodes in an Aurora DB cluster are patched at the same time.

Note

This version is currently not available in the AWS GovCloud (US-West) [us-gov-west-1] and China (Beijing) [cn-north-1] regions. There will be a separate announcement once it is made available.

If you have any questions or concerns, AWS Support is available on the community forums and through [AWS Premium Support](#). For more information, see [Maintaining an Amazon Aurora DB Cluster \(p. 408\)](#).

Improvements

- Fixed an issue where an Aurora Replica can restart when using optimistic cursor restores while reading records.
- Fixed an issue where an Aurora Writer restarts when trying to kill a MySQL session (`kill "<session id>"`) with performance schema enabled.
- Fixed an issue where an Aurora Writer restarts when computing a threshold for garbage collection.
- Fixed an issue where an Aurora Writer can occasionally restart when tracking Aurora Replica progress in log application.
- Fixed an issue with the Query Cache when auto-commit is off and that could potentially cause stale reads.

Aurora MySQL Database Engine Updates 2018-04-27

Version: 1.17.2

Aurora MySQL 1.17.2 is generally available. All new Aurora MySQL database clusters with MySQL 5.6 compatibility, including those restored from snapshots, will be created in Aurora MySQL 1.17.2. You have the option, but are not required, to upgrade existing database clusters to Aurora MySQL 1.17.2. You can create new database clusters in Aurora MySQL 1.14.4, Aurora MySQL 1.15.1, or Aurora MySQL 1.16. You can do so using the AWS CLI or the Amazon RDS API and specifying the engine version.

With version 1.17.2 of Aurora MySQL, we are using a cluster patching model where all nodes in an Aurora DB cluster are patched at the same time.

If you have any questions or concerns, AWS Support is available on the community forums and through [AWS Premium Support](#). For more information, see [Maintaining an Amazon Aurora DB Cluster \(p. 408\)](#).

Improvements

- Fixed an issue which was causing restarts during certain DDL partition operations.
- Fixed an issue which was causing support for invocation of AWS Lambda functions via native Aurora MySQL functions to be disabled.
- Fixed an issue with cache invalidation which was causing restarts on Aurora Replicas.
- Fixed an issue in lock manager which was causing restarts.

Aurora MySQL Database Engine Updates 2018-03-23

Version: 1.17.1

Aurora MySQL 1.17.1 is generally available. All new database clusters, including those restored from snapshots, will be created in Aurora MySQL 1.17.1. You have the option, but are not required, to upgrade existing database clusters to Aurora MySQL 1.17.1. You can create new DB clusters in Aurora MySQL 1.15.1, Aurora MySQL 1.16, or Aurora MySQL 1.17. You can do so using the AWS CLI or the Amazon RDS API and specifying the engine version.

With version 1.17.1 of Aurora MySQL, we are using a cluster patching model where all nodes in an Aurora DB cluster are patched at the same time. This release fixes some known engine issues as well as regressions.

If you have any questions or concerns, AWS Support is available on the community forums and through [AWS Premium Support](#). For more information, see [Maintaining an Amazon Aurora DB Cluster \(p. 408\)](#).

Note

There is an issue in the latest version of the Aurora MySQL engine. After upgrading to 1.17.1, the engine version is reported incorrectly as 1.17. If you upgraded to 1.17.1, you can confirm the upgrade by checking the **Maintenance** column for the DB cluster in the AWS Management Console. If it displays none, then the engine is upgraded to 1.17.1.

Improvements

- Fixed an issue in binary log recovery that resulted in longer recovery times for situations with large binary log index files which can happen if binary logs rotate very often.
- Fixed an issue in the query optimizer that generated an inefficient query plan for partitioned tables.
- Fixed an issue in the query optimizer due to which a range query resulted in a restart of the database engine.

Aurora MySQL Database Engine Updates 2018-03-13

Version: 1.17

Aurora MySQL 1.17 is generally available. Aurora MySQL 1.x versions are only compatible with MySQL 5.6, and not MySQL 5.7. All new 5.6-compatible database clusters, including those restored from snapshots, will be created in Aurora 1.17. You have the option, but are not required, to upgrade existing database clusters to Aurora 1.17. You can create new DB clusters in Aurora 1.14.1, Aurora 1.15.1, or Aurora 1.16. You can do so using the AWS CLI or the Amazon RDS API and specifying the engine version.

With version 1.17 of Aurora, we are using a cluster patching model where all nodes in an Aurora DB cluster are patched at the same time. We support zero-downtime patching, which works on a best-effort

basis to preserve client connections through the patching process. For more information, see [Maintaining an Amazon Aurora DB Cluster \(p. 408\)](#).

If you have any questions or concerns, AWS Support is available on the community forums and through [AWS Premium Support](#).

Zero-Downtime Patching

The zero-downtime patching (ZDP) feature attempts, on a best-effort basis, to preserve client connections through an engine patch. For more information about ZDP, see [Zero-Downtime Patching \(p. 796\)](#).

New Features

- Aurora MySQL now supports lock compression, which optimizes the lock manager's memory usage. Starting in version 1.17, you can use this feature without enabling lab mode.

Improvements

- Fixed an issue predominantly seen on instances with fewer cores where a single core might have 100% CPU utilization even when the database is idle.
- Improved the performance of fetching binary logs from Aurora clusters.
- Fixed an issue where Aurora Replicas attempt to write table statistics to persistent storage, and crash.
- Fixed an issue where query cache did not work as expected on Aurora Replicas.
- Fixed a race condition in lock manager that resulted in an engine restart.
- Fixed an issue where locks taken by read-only, auto-commit transactions resulted in an engine restart.
- Fixed an issue where some queries are not written to the audit logs.
- Fixed an issue with recovery of certain partition maintenance operations on failover.

Integration of MySQL Bug Fixes

- LAST_INSERT_ID is replicated incorrectly if replication filters are used (Bug #69861)
- Query returns different results depending on whether INDEX_MERGE setting (Bug #16862316)
- Query proc re-execute of stored routine, inefficient query plan (Bug #16346367)
- INNODB FTS : Assert in FTS_CACHE_APPEND_DELETED_DOC_IDS (BUG #18079671)
- Assert RBT_EMPTY(INDEX_CACHE->WORDS) in ALTER TABLE CHANGE COLUMN (BUG #17536995)
- INNODB fulltext search doesn't find records when savepoints are involved (BUG #70333, BUG #17458835)

Aurora MySQL Database Engine Updates 2017-12-11

Version: 1.16

Aurora MySQL 1.16 is generally available. All new database clusters, including those restored from snapshots, will be created in Aurora 1.16. You have the option, but are not required, to upgrade existing database clusters to Aurora 1.16. You can create new DB clusters in Aurora 1.14.1 or Aurora 1.15.1. You can do so using the AWS CLI or the Amazon RDS API and specifying the engine version.

With version 1.16 of Aurora, we are using a cluster patching model where all nodes in an Aurora DB cluster are patched at the same time. We are enabling zero-downtime patching, which works on a best-effort basis to preserve client connections through the patching process. For more information, see [Maintaining an Amazon Aurora DB Cluster \(p. 408\)](#).

If you have any questions or concerns, AWS Support is available on the community forums and through AWS Premium Support at <http://aws.amazon.com/support>.

Zero-Downtime Patching

The zero-downtime patching (ZDP) feature attempts, on a best-effort basis, to preserve client connections through an engine patch. For more information about ZDP, see [Zero-Downtime Patching \(p. 796\)](#).

New Features

- Aurora MySQL now supports synchronous AWS Lambda invocations via the native function `lambda_sync()`. Also available is native function `lambda_async()`, which can be used as an alternative to the existing stored procedure for asynchronous Lambda invocation. For more information, see [Invoking a Lambda Function from an Amazon Aurora MySQL DB Cluster \(p. 752\)](#).
- Aurora MySQL now supports hash joins to speed up equijoin queries. Aurora's cost-based optimizer can automatically decide when to use hash joins; you can also force their use in a query plan. For more information, see [Working with Hash Joins in Aurora MySQL \(p. 771\)](#).
- Aurora MySQL now supports scan batching to speed up in-memory scan-oriented queries significantly. The feature boosts the performance of table full scans, index full scans, and index range scans by batch processing.

Improvements

- Fixed an issue where read replicas crashed when running queries on tables that have just been dropped on the master.
- Fixed an issue where restarting the writer on a database cluster with a very large number of `FULLTEXT` indexes results in longer than expected recovery.
- Fixed an issue where flushing binary logs causes `LOST_EVENTS` incidents in binlog events.
- Fixed stability issues with the scheduler when performance schema is enabled.
- Fixed an issue where a subquery that uses temporary tables could return partial results.

Integration of MySQL Bug Fixes

None

Aurora MySQL Database Engine Updates 2017-11-20

Version: 1.15.1

Aurora MySQL 1.15.1 is generally available. All new database clusters, including those restored from snapshots, will be created in Aurora 1.15.1. You have the option, but are not required, to upgrade existing DB clusters to Aurora 1.15.1. You can create new DB clusters in Aurora 1.14.1. You can do so using the AWS CLI or the Amazon RDS API and specifying the engine version.

With version 1.15.1 of Aurora, we are using a cluster patching model where all nodes in an Aurora DB cluster are patched at the same time. We are enabling zero-downtime patching, which works on a best-effort basis to preserve client connections through the patching process. For more information, see [Maintaining an Amazon Aurora DB Cluster \(p. 408\)](#).

If you have any questions or concerns, AWS Support is available on the community forums and through AWS Premium Support at <http://aws.amazon.com/support>. For more information, see [Maintaining an Amazon Aurora DB Cluster \(p. 408\)](#).

Zero-Downtime Patching

The zero-downtime patching (ZDP) feature attempts, on a best-effort basis, to preserve client connections through an engine patch. For more information about ZDP, see [Zero-Downtime Patching \(p. 796\)](#).

Improvements

- Fixed an issue in the adaptive segment selector for a read request that would cause it to choose the same segment twice causing a spike in read latency under certain conditions.
- Fixed an issue that stems from an optimization in Aurora MySQL for the thread scheduler. This problem manifests itself into what are spurious errors while writing to the slow log, while the associated queries themselves perform fine.
- Fixed an issue with stability of read replicas on large (> 5 TB) volumes.
- Fixed an issue where worker thread count increases continuously due to a bogus outstanding connection count.
- Fixed an issue with table locks that caused long semaphore waits during insert workloads.
- Reverted the following MySQL bug fixes included in Aurora MySQL 1.15:
 - MySQL instance stalling “doing SYNC index” (Bug #73816)
 - Assert RBT_EMPTY(INDEX_CACHE->WORDS) in ALTER TABLE CHANGE COLUMN (Bug #17536995)
 - InnoDB Fulltext search doesn't find records when savepoints are involved (Bug #70333)

Integration of MySQL Bug Fixes

None

Aurora MySQL Database Engine Updates 2017-10-24

Version: 1.15

Aurora MySQL 1.15 is generally available. All new database clusters, including those restored from snapshots, will be created in Aurora 1.15. You have the option, but are not required, to upgrade existing DB clusters to Aurora 1.15. You can create new DB clusters in Aurora 1.14.1. You can do so using the AWS CLI or the Amazon RDS API and specifying the engine version.

With version 1.15 of Aurora, we are using a cluster patching model where all nodes in an Aurora DB cluster are patched at the same time. Updates require a database restart, so you will experience 20 to 30 seconds of downtime, after which you can resume using your DB cluster or clusters. If your DB clusters are currently running Aurora 1.14 or Aurora 1.14.1, the zero-downtime patching feature in Aurora MySQL might allow client connections to your Aurora MySQL primary instance to persist through the upgrade, depending on your workload.

If you have any questions or concerns, AWS Support is available on the community forums and through AWS Premium Support at <http://aws.amazon.com/support>. For more information, see [Maintaining an Amazon Aurora DB Cluster \(p. 408\)](#).

Zero-Downtime Patching

The zero-downtime patching (ZDP) feature attempts, on a best-effort basis, to preserve client connections through an engine patch. For more information about ZDP, see [Zero-Downtime Patching \(p. 796\)](#).

New Features

- **Asynchronous Key Prefetch** – Asynchronous key prefetch (AKP) is a feature targeted to improve the performance of non-cached index joins, by prefetching keys in memory ahead of when they

are needed. The primary use case targeted by AKP is an index join between a small outer and large inner table, where the index is highly selective on the larger table. Also, when the Multi-Range Read (MRR) interface is enabled, AKP will be leveraged for a secondary to primary index lookup. Smaller instances which have memory constraints might in some cases be able to leverage AKP, given the right key cardinality. For more information, see [Working with Asynchronous Key Prefetch in Amazon Aurora \(p. 765\)](#).

- **Fast DDL** – We have extended the feature that was released in [Aurora 1.13 \(p. 852\)](#) to operations that include default values. With this extension, Fast DDL is applicable for operations that add a nullable column at the end of a table, with or without default values. The feature remains under Aurora lab mode. For more information, see [Altering Tables in Amazon Aurora Using Fast DDL \(p. 642\)](#).

Improvements

- Fixed a calculation error during optimization of WITHIN/CONTAINS spatial queries which previously resulted in an empty result set.
- Fixed SHOW VARIABLE command to show the updated `innodb_buffer_pool_size` parameter value whenever it is changed in the parameter group.
- Improved stability of primary instance during bulk insert into a table altered using Fast DDL when adaptive hash indexing is disabled and the record to be inserted is the first record of a page.
- Improved stability of Aurora when the user attempts to set `server_audit_events` DB cluster parameter value to `default`.
- Fixed an issue in which a database character set change for an ALTER TABLE statement that ran on the Aurora primary instance was not being replicated on the Aurora Replicas until they were restarted.
- Improved stability by fixing a race condition on the primary instance which previously allowed it to register an Aurora Replica even if the primary instance had closed its own volume.
- Improved performance of the primary instance during index creation on a large table by changing the locking protocol to enable concurrent data manipulation language (DML) statements during index build.
- Fixed InnoDB metadata inconsistency during ALTER TABLE RENAME query which improved stability. Example: When columns of table t1(c1, c2) are renamed cyclically to t1(c2,c3) within the same ALTER statement.
- Improved stability of Aurora Replicas for the scenario where an Aurora Replica has no active workload and the primary instance is unresponsive.
- Improved availability of Aurora Replicas for a scenario in which the Aurora Replica holds an explicit lock on a table and blocks the replication thread from applying any DDL changes received from the primary instance.
- Improved stability of the primary instance when a foreign key and a column are being added to a table from two separate sessions at the same time and Fast DDL has been enabled.
- Improved stability of the purge thread on the primary instance during a heavy write workload by blocking truncate of undo records until they have been purged.
- Improved stability by fixing the lock release order during commit process of transactions which drop tables.
- Fixed a defect for Aurora Replicas in which the DB instance could not complete startup and complained that port 3306 was already in use.
- Fixed a race condition in which a SELECT query run on certain information_schema tables (`innodb trx`, `innodb lock`, `innodb lock waits`) increased cluster instability.

Integration of MySQL Bug Fixes

- CREATE USER accepts plugin and password hash, but ignores the password hash (Bug #78033)

- The partitioning engine adds fields to the read bit set to be able to return entries sorted from a partitioned index. This leads to the join buffer will try to read unneeded fields. Fixed by not adding all partitioning fields to the read_set, but instead only sort on the already set prefix fields in the read_set. Added a DBUG_ASSERT that if doing key_cmp, at least the first field must be read (Bug #16367691).
- MySQL instance stalling “doing SYNC index” (Bug #73816)
- Assert RBT_EMPTY(INDEX_CACHE->WORDS) in ALTER TABLE CHANGE COLUMN (Bug #17536995)
- InnoDB Fulltext search doesn't find records when savepoints are involved (Bug #70333)

Aurora MySQL Database Engine Updates: 2018-03-13

Version: 1.14.4

Aurora MySQL 1.14.4 is generally available. You can create new DB clusters in Aurora 1.14.4, using the AWS CLI or the Amazon RDS API and specifying the engine version. You have the option, but are not required, to upgrade existing 1.14.x DB clusters to Aurora 1.14.4.

With version 1.14.4 of Aurora, we are using a cluster-patching model where all nodes in an Aurora DB cluster are patched at the same time. We support zero-downtime patching, which works on a best-effort basis to preserve client connections through the patching process. For more information, see [Maintaining an Amazon Aurora DB Cluster \(p. 408\)](#).

If you have any questions or concerns, AWS Support is available on the community forums and through AWS Premium Support at <http://aws.amazon.com/support>. For more information, see [Maintaining an Amazon Aurora DB Cluster \(p. 408\)](#).

Zero-Downtime Patching

The zero-downtime patching (ZDP) feature attempts, on a best-effort basis, to preserve client connections through an engine patch. For more information about ZDP, see [Zero-Downtime Patching \(p. 796\)](#).

New Features

- Aurora MySQL now supports db.r4 instance classes.

Improvements

- Fixed an issue where LOST_EVENTS were generated when writing large binlog events.

Integration of MySQL Bug Fixes

- Ignorable events don't work and are not tested (Bug #74683)
- NEW->OLD ASSERT FAILURE 'GTID_MODE > 0' (Bug #20436436)

Aurora MySQL Database Engine Updates: 2017-09-22

Version: 1.14.1

Aurora MySQL 1.14.1 is generally available. All new database clusters, including those restored from snapshots, will be created in Aurora MySQL 1.14.1. Aurora MySQL 1.14.1 is also a mandatory upgrade for existing Aurora MySQL DB clusters. For more information, see [Announcement: Extension to Mandatory Upgrade Schedule for Amazon Aurora](#) on the AWS Developer Forums website.

With version 1.14.1 of Aurora MySQL, we are using a cluster patching model where all nodes in an Aurora MySQL DB cluster are patched at the same time. Updates require a database restart, so you will experience 20 to 30 seconds of downtime, after which you can resume using your DB cluster or clusters. If your DB clusters are currently running version 1.13 or greater, the zero-downtime patching feature in Aurora MySQL might allow client connections to your Aurora MySQL primary instance to persist through the upgrade, depending on your workload.

If you have any questions or concerns, AWS Support is available on the community forums and through AWS Premium Support at <http://aws.amazon.com/support>.

Improvements

- Fixed race conditions associated with inserts and purge to improve the stability of the Fast DDL feature, which remains in Aurora MySQL lab mode.

Aurora MySQL Database Engine Updates: 2017-08-07

Version: 1.14

Aurora MySQL 1.14 is generally available. All new database clusters, including those restored from snapshots, will be created in Aurora MySQL 1.14. Aurora MySQL 1.14 is also a mandatory upgrade for existing Aurora MySQL DB clusters. We will send a separate announcement with the timeline for deprecating earlier versions of Aurora MySQL.

With version 1.14 of Aurora MySQL, we are using a cluster patching model where all nodes in an Aurora DB cluster are patched at the same time. Updates require a database restart, so you will experience 20 to 30 seconds of downtime, after which you can resume using your DB cluster or clusters. If your DB clusters are currently running version 1.13, Aurora's zero-downtime patching feature may allow client connections to your Aurora primary instance to persist through the upgrade, depending on your workload.

If you have any questions or concerns, AWS Support is available on the community forums and through AWS Premium Support at <http://aws.amazon.com/support>.

Zero-Downtime Patching

The zero-downtime patching (ZDP) feature attempts, on a best-effort basis, to preserve client connections through an engine patch. For more information about ZDP, see [Zero-Downtime Patching \(p. 796\)](#).

Improvements

- Fixed an incorrect "record not found" error when a record is found in the secondary index but not in the primary index.
- Fixed a stability issue that can occur due to a defensive assertion (added in 1.12) that was too strong in the case when an individual write spans over 32 pages. Such a situation can occur, for instance, with large BLOB values.
- Fixed a stability issue because of inconsistencies between the tablespace cache and the dictionary cache.
- Fixed an issue in which an Aurora Replica becomes unresponsive after it has exceeded the maximum number of attempts to connect to the primary instance. An Aurora Replica now restarts if the period of inactivity is more than the heartbeat time period used for health check by the primary instance.
- Fixed a livelock that can occur under very high concurrency when one connection tries to acquire an exclusive meta data lock (MDL) while issuing a command, such as `ALTER TABLE`.
- Fixed a stability issue in an Aurora Read Replica in the presence of logical/parallel read ahead.
- Improved `LOAD FROM S3` in two ways:

1. Better handling of Amazon S3 timeout errors by using the SDK retry in addition to the existing retry.
 2. Performance optimization when loading very big files or large numbers of files by caching and reusing client state.
- Fixed the following stability issues with Fast DDL for `ALTER TABLE` operations:
 1. When the `ALTER TABLE` statement has multiple `ADD COLUMN` commands and the column names are not in ascending order.
 2. When the name string of the column to be updated and its corresponding name string, fetched from the internal system table, differs by a null terminating character (/0).
 3. Under certain B-tree split operations.
 4. When the table has a variable length primary key.
 - Fixed a stability issue with Aurora Replicas when it takes too long to make its Full Text Search (FTS) index cache consistent with that of the primary instance. This can happen if a large portion of the newly created FTS index entries on the primary instance have not yet been flushed to disk.
 - Fixed a stability issue that can happen during index creation.
 - New infrastructure that tracks memory consumption per connection and associated telemetry that will be used for building out Out-Of-Memory (OOM) avoidance strategies.
 - Fixed an issue where `ANALYZE TABLE` was incorrectly allowed on Aurora Replicas. This has now been blocked.
 - Fixed a stability issue caused by a rare deadlock as a result of a race condition between logical read-ahead and purge.

Integration of MySQL Bug Fixes

- A full-text search combined with derived tables (subqueries in the `FROM` clause) caused a server exit. Now, if a full-text operation depends on a derived table, the server produces an error indicating that a full-text search cannot be done on a materialized table. (Bug #68751, Bug #16539903)

Aurora MySQL Database Engine Updates: 2017-05-15

Version: 1.13

Note

We enabled a new feature - `SELECT INTO OUTFILE S3` - in Aurora MySQL version 1.13 after the initial release, and have updated the release notes to reflect that change.

Aurora MySQL 1.13 is generally available. All new database clusters, including those restored from snapshots, will be created in Aurora MySQL 1.13. You have the option, but are not required, to upgrade existing database clusters to Aurora MySQL 1.13. With version 1.13 of Aurora, we are using a cluster patching model where all nodes in an Aurora DB cluster are patched at the same time. We are enabling zero-downtime patching, which works on a best-effort basis to preserve client connections through the patching process. For more information, see [Maintaining an Amazon Aurora DB Cluster \(p. 408\)](#).

Zero-Downtime Patching

The zero-downtime patching (ZDP) feature attempts, on a best-effort basis, to preserve client connections through an engine patch. For more information about ZDP, see [Zero-Downtime Patching \(p. 796\)](#).

New Features:

- **`SELECT INTO OUTFILE S3`** – Aurora MySQL now allows you to upload the results of a query to one or more files in an Amazon S3 bucket. For more information, see [Saving Data from an Amazon Aurora MySQL DB Cluster into Text Files in an Amazon S3 Bucket \(p. 747\)](#).

Improvements:

- Implemented truncation of CSV format log files at engine startup to avoid long recovery time. The `general_log_backup`, `general_log`, `slow_log_backup`, and `slow_log` tables now don't survive a database restart.
- Fixed an issue where migration of a database named `test` would fail.
- Improved stability in the lock manager's garbage collector by reusing the correct lock segments.
- Improved stability of the lock manager by removing invalid assertions during deadlock detection algorithm.
- Re-enabled asynchronous replication, and fixed an associated issue which caused incorrect replica lag to be reported under no-load or read-only workload. The replication pipeline improvements that were introduced in version 1.10. These improvements were introduced in order to apply log stream updates to the buffer cache of an Aurora Replica, which helps to improve read performance and stability on Aurora Replicas.
- Fixed an issue where `autocommit=OFF` leads to scheduled events being blocked and long transactions being held open until the server reboots.
- Fixed an issue where general, audit, and slow query logs could not log queries handled by asynchronous commit.
- Improved the performance of the logical read ahead (LRA) feature by up to 2.5 times. This was done by allowing pre-fetches to continue across intermediate pages in a B-tree.
- Added parameter validation for audit variables to trim unnecessary spaces.
- Fixed a regression, introduced in Aurora MySQL version 1.11, in which queries can return incorrect results when using the `SQL_CALC_FOUND_ROWS` option and invoking the `FOUND_ROWS()` function.
- Fixed a stability issue when the Metadata Lock list was incorrectly formed.
- Improved stability when `sql_mode` is set to `PAD_CHAR_TO_FULL_LENGTH` and the command `SHOW FUNCTION STATUS WHERE Db='string'` is executed.
- Fixed a rare case when instances would not come up after Aurora version upgrade because of a false volume consistency check.
- Fixed the performance issue, introduced in Aurora MySQL version 1.12, where the performance of the Aurora writer was reduced when users have a large number of tables.
- Improved stability issue when the Aurora writer is configured as a binlog slave and the number of connections approaches 16,000.
- Fixed a rare issue where an Aurora Replica could restart when a connection gets blocked waiting for Metadata Lock when running DDL on the Aurora master.

Integration of MySQL Bug Fixes

- With an empty InnoDB table, it's not possible to decrease the `auto_increment` value using an `ALTER TABLE` statement, even when the table is empty. (Bug #69882)
- `MATCH() ... AGAINST` queries that use a long string as an argument for `AGAINST()` could result in an error when run on an InnoDB table with a full-text search index. (Bug #17640261)
- Handling of `SQL_CALC_FOUND_ROWS` in combination with `ORDER BY` and `LIMIT` could lead to incorrect results for `FOUND_ROWS()`. (Bug #68458, Bug # 16383173)
- `ALTER TABLE` does not allow to change nullability of the column if foreign key exists. (Bug #77591)

Aurora MySQL Database Engine Updates: 2017-04-05

Version: 1.12

Aurora MySQL 1.12 is now the preferred version for the creation of new DB clusters, including restores from snapshots.

This is not a mandatory upgrade for existing clusters. You will have the option to upgrade existing clusters to version 1.12 after we complete the fleet-wide patch to 1.11 (see [Aurora 1.11 release notes \(p. 855\)](#) and corresponding [forum announcement](#)). With version 1.12 of Aurora, we are using a cluster patching model where all nodes in an Aurora DB cluster are patched at the same time. For more information, see [Maintaining an Amazon Aurora DB Cluster \(p. 408\)](#).

New Features

- **Fast DDL** – Aurora MySQL now allows you to execute an ALTER TABLE *tbl_name* ADD COLUMN *col_name column_definition* operation nearly instantaneously. The operation completes without requiring the table to be copied and without materially impacting other DML statements. Since it does not consume temporary storage for a table copy, it makes DDL statements practical even for large tables on small instance classes. Fast DDL is currently only supported for adding a nullable column, without a default value, at the end of a table. This feature is currently available in Aurora lab mode. For more information, see [Altering Tables in Amazon Aurora Using Fast DDL \(p. 642\)](#).
- **Show volume status** – We have added a new monitoring command, SHOW VOLUME STATUS, to display the number of nodes and disks in a volume. For more information, see [Displaying Volume Status for an Aurora DB Cluster \(p. 643\)](#).

Improvements

- Implemented changes to lock compression to further reduce memory allocated per lock object. This improvement is available in lab mode.
- Fixed an issue where the `trx_active_transactions` metric decrements rapidly even when the database is idle.
- Fixed an invalid error message regarding fault injection query syntax when simulating failure in disks and nodes.
- Fixed multiple issues related to race conditions and dead latches in the lock manager.
- Fixed an issue causing a buffer overflow in the query optimizer.
- Fixed a stability issue in Aurora read replicas when the underlying storage nodes experience low available memory.
- Fixed an issue where idle connections persisted past the `wait_timeout` parameter setting.
- Fixed an issue where `query_cache_size` returns an unexpected value after reboot of the instance.
- Fixed a performance issue that is the result of a diagnostic thread probing the network too often in the event that writes are not progressing to storage.

Integration of MySQL Bug Fixes

- Reloading a table that was evicted while empty caused an AUTO_INCREMENT value to be reset. ([Bug #21454472](#), [Bug #77743](#))
- An index record was not found on rollback due to inconsistencies in the `purge_node_t` structure. The inconsistency resulted in warnings and error messages such as "error in sec index entry update", "unable to purge a record", and "tried to purge sec index entry not marked for deletion". ([Bug #19138298](#), [Bug #70214](#), [Bug #21126772](#), [Bug #21065746](#))
- Wrong stack size calculation for `qsort` operation leads to stack overflow. ([Bug #73979](#))
- Record not found in an index upon rollback. ([Bug #70214](#), [Bug #72419](#))
- ALTER TABLE add column TIMESTAMP on update CURRENT_TIMESTAMP inserts ZERO-datas ([Bug #17392](#))

Aurora MySQL Database Engine Updates: 2017-02-23

Version: 1.11

We will patch all Aurora MySQL DB clusters with the latest version over a short period following the release. DB clusters are patched using the legacy procedure with a downtime of about 5-30 seconds.

Patching occurs during the system maintenance window that you have specified for each of your database instances. You can view or change this window using the AWS Management Console. For more information, see [Maintaining an Amazon Aurora DB Cluster \(p. 408\)](#).

Alternatively, you can apply the patch immediately in the AWS Management Console by choosing a DB cluster, choosing **Cluster Actions**, and then choosing **Upgrade Now**.

With version 1.11 of Aurora MySQL, we are using a cluster patching model where all nodes in an Aurora DB cluster are patched at the same time.

New Features

- **MANIFEST option for LOAD DATA FROM S3** – LOAD DATA FROM S3 was released in version 1.8. The options for this command have been expanded, and you can now specify a list of files to be loaded into an Aurora DB cluster from Amazon S3 by using a manifest file. This makes it easy to load data from specific files in one or more locations, as opposed to loading data from a single file by using the FILE option or loading data from multiple files that have the same location and prefix by using the PREFIX option. The manifest file format is the same as that used by Amazon Redshift. For more information about using LOAD DATA FROM S3 with the MANIFEST option, see [Using a Manifest to Specify Data Files to Load \(p. 742\)](#).
- **Spatial indexing enabled by default** – This feature was released in lab mode in version 1.10, and is now turned on by default. Spatial indexing improves query performance on large datasets for queries that use spatial data. For more information about using spatial indexing, see [Amazon Aurora MySQL and Spatial Data \(p. 579\)](#).
- **Advanced Auditing timing change** – This feature was released in version 1.10.1 to provide a high-performance facility for auditing database activity. In this release, the precision of audit log timestamps has been changed from one second to one microsecond. The more accurate timestamps allow you to better understand when an audit event happened. For more information about audit, see [Using Advanced Auditing with an Amazon Aurora MySQL DB Cluster \(p. 669\)](#).

Improvements

- Modified the `thread_handling` parameter to prevent you from setting it to anything other than `multiple-connections-per-thread`, which is the only supported model with Aurora's thread pool.
- Fixed an issue caused when you set either the `buffer_pool_size` or the `query_cache_size` parameter to be larger than the DB cluster's total memory. In this circumstance, Aurora sets the modified parameter to the default value, so the DB cluster can start up and not crash.
- Fixed an issue in the query cache where a transaction gets stale read results if the table is invalidated in another transaction.
- Fixed an issue where binlog files marked for deletion are removed after a small delay rather than right away.
- Fixed an issue where a database created with the name `tmp` is treated as a system database stored on ephemeral storage and not persisted to Aurora distributed storage.
- Modified the behavior of SHOW TABLES to exclude certain internal system tables. This change helps avoid an unnecessary failover caused by mysqldump locking all files listed in SHOW TABLES, which in turn prevents writes on the internal system table, causing the failover.

- Fixed an issue where an Aurora Replica incorrectly restarts when creating a temporary table from a query that invokes a function whose argument is a column of an InnoDB table.
- Fixed an issue related to a metadata lock conflict in an Aurora Replica node that causes the Aurora Replica to fall behind the primary DB cluster and eventually get restarted.
- Fixed a dead latch in the replication pipeline in reader nodes, which causes an Aurora Replica to fall behind and eventually get restarted.
- Fixed an issue where an Aurora Replica lags too much with encrypted volumes larger than 1 terabyte (TB).
- Improved Aurora Replica dead latch detection by using an improved way to read the system clock time.
- Fixed an issue where an Aurora Replica can restart twice instead of once following de-registration by the writer.
- Fixed a slow query performance issue on Aurora Replicas that occurs when transient statistics cause statistics discrepancy on non-unique index columns.
- Fixed an issue where an Aurora Replica can crash when a DDL statement is replicated on the Aurora Replica at the same time that the Aurora Replica is processing a related query.
- Changed the replication pipeline improvements that were introduced in version 1.10 from enabled by default to disabled by default. These improvements were introduced in order to apply log stream updates to the buffer cache of an Aurora Replica, and although this feature helps to improve read performance and stability on Aurora Replicas, it increases replica lag in certain workloads.
- Fixed an issue where the simultaneous occurrence of an ongoing DDL statement and pending Parallel Read Ahead on the same table causes an assertion failure during the commit phase of the DDL transaction.
- Enhanced the general log and slow query log to survive DB cluster restart.
- Fixed an out-of-memory issue for certain long running queries by reducing memory consumption in the ACL module.
- Fixed a restart issue that occurs when a table has non-spatial indexes, there are spatial predicates in the query, the planner chooses to use a non-spatial index, and the planner incorrectly pushes the spatial condition down to the index.
- Fixed an issue where the DB cluster restarts when there is a delete, update, or purge of very large geospatial objects that are stored externally (like LOBs).
- Fixed an issue where fault simulation using ALTER SYSTEM SIMULATE ... FOR INTERVAL isn't working properly.
- Fixed a stability issue caused by an invalid assertion on an incorrect invariant in the lock manager.
- Disabled the following two improvements to InnoDB Full-Text Search that were introduced in version 1.10 because they introduce stability issues for some demanding workloads:
 - Updating the cache only after a read request to an Aurora Replica in order to improve full-text search index cache replication speed.
 - Offloading the cache sync task to a separate thread as soon as the cache size crosses 10% of the total size, in order to avoid MySQL queries stalling for too long during FTS cache sync to disk. (Bugs #22516559, #73816).

Integration of MySQL Bug Fixes

- Running ALTER table DROP foreign key simultaneously with another DROP operation causes the table to disappear. (Bug #16095573)
- Some INFORMATION_SCHEMA queries that used ORDER BY did not use a filesort optimization as they did previously. (Bug #16423536)
- FOUND_ROWS () returns the wrong count of rows on a table. (Bug #68458)
- The server fails instead of giving an error when too many temp tables are open. (Bug #18948649)

Aurora MySQL Database Engine Updates: 2017-01-12

Version: 1.10.1

Version 1.10.1 of Aurora MySQL is an opt-in version and is not used to patch your database instances. It is available for creating new Aurora instances and for upgrading existing instances. You can apply the patch by choosing a cluster in the [Amazon RDS console](#), choosing **Cluster Actions**, and then choosing **Upgrade Now**. Patching requires a database restart with downtime typically lasting 5-30 seconds, after which you can resume using your DB clusters. This patch is using a cluster patching model where all nodes in an Aurora cluster are patched at the same time.

New Features

- **Advanced Auditing** – Aurora MySQL provides a high-performance Advanced Auditing feature, which you can use to audit database activity. For more information about enabling and using Advanced Auditing, see [Using Advanced Auditing with an Amazon Aurora MySQL DB Cluster \(p. 669\)](#).

Improvements

- Fixed an issue with spatial indexing when creating a column and adding an index on it in the same statement.
- Fixed an issue where spatial statistics aren't persisted across DB cluster restart.

Aurora MySQL Database Engine Updates: 2016-12-14

Version: 1.10

New Features

- **Zero downtime patch** – This feature allows a DB instance to be patched without any downtime. That is, database upgrades are performed without disconnecting client applications, or rebooting the database. This approach increases the availability of your Aurora DB clusters during the maintenance window. Note that temporary data like that in the performance schema is reset during the upgrade process. This feature applies to service-delivered patches during a maintenance window as well as user-initiated patches.

When a patch is initiated, the service ensures there are no open locks, transactions or temporary tables, and then waits for a suitable window during which the database can be patched and restarted. Application sessions are preserved, although there is a drop in throughput while the patch is in progress (for approximately 5 seconds). If no suitable window can be found, then patching defaults to the standard patching behavior.

Zero downtime patching takes place on a best-effort basis, subject to certain limitations as described following:

- This feature is currently applicable for patching single-node DB clusters or writer instances in multi-node DB clusters.
- SSL connections are not supported in conjunction with this feature. If there are active SSL connections, Amazon Aurora MySQL won't perform a zero downtime patch, and instead will retry periodically to see if the SSL connections have terminated. If they have, zero downtime patching proceeds. If the SSL connections persist after more than a couple seconds, standard patching with downtime proceeds.
- The feature is available in Aurora release 1.10 and beyond. Going forward, we will identify any releases or patches that can't be applied by using zero downtime patching.
- This feature is not applicable if replication based on binary logging is active.

- **Spatial indexing** – Spatial indexing improves query performance on large datasets for queries that use spatial data. For more information about using spatial indexing, see [Amazon Aurora MySQL and Spatial Data \(p. 579\)](#).

This feature is disabled by default and can be activated by enabling Aurora lab mode. For information, see [Amazon Aurora MySQL Lab Mode \(p. 762\)](#).

- **Replication pipeline improvements** – Aurora MySQL now uses an improved mechanism to apply log stream updates to the buffer cache of an Aurora Replica. This feature improves the read performance and stability on Aurora Replicas when there is a heavy write load on the master as well as a significant read load on the Replica. This feature is enabled by default.
- **Throughput improvement for workloads with cached reads** – Aurora MySQL now uses a latch-free concurrent algorithm to implement read views, which leads to better throughput for read queries served by the buffer cache. As a result of this and other improvements, Amazon Aurora MySQL can achieve throughput of up to 625K reads per second compared to 164K reads per second by MySQL 5.7 for a SysBench SELECT-only workload.
- **Throughput improvement for workloads with hot row contention** – Aurora MySQL uses a new lock release algorithm that improves performance, particularly when there is hot page contention (that is, many transactions contending for the rows on the same page). In tests with the TPC-C benchmark, this can result in up to 16x throughput improvement in transactions per minute relative to MySQL 5.7. This feature is disabled by default and can be activated by enabling Aurora lab mode. For information, see [Amazon Aurora MySQL Lab Mode \(p. 762\)](#).

Improvements

- Full-text search index cache replication speed has been improved by updating the cache only after a read request to an Aurora Replica. This approach avoids any reads from disk by the replication thread.
- Fixed an issue where dictionary cache invalidation does not work on an Aurora Replica for tables that have a special character in the database name or table name.
- Fixed a `STUCK IO` issue during data migration for distributed storage nodes when storage heat management is enabled.
- Fixed an issue in the lock manager where an assertion check fails for the transaction lock wait thread when preparing to rollback or commit a transaction.
- Fixed an issue when opening a corrupted dictionary table by correctly updating the reference count to the dictionary table entries.
- Fixed a bug where the DB cluster minimum read point can be held by slow Aurora Replicas.
- Fixed a potential memory leak in the query cache.
- Fixed a bug where an Aurora Replica places a row-level lock on a table when a query is used in an `IF` statement of a stored procedure.

Integration of MySQL Bug Fixes

- UNION of derived tables returns wrong results with '1=0/false'-clauses. (Bug #69471)
- Server crashes in `ITEM_FUNC_GROUP_CONCAT::FIX_FIELDS` on 2nd execution of stored procedure. (Bug #20755389)
- Avoid MySQL queries from stalling for too long during FTS cache sync to disk by offloading the cache sync task to a separate thread, as soon as the cache size crosses 10% of the total size. (Bug #22516559, #73816)

Aurora MySQL Database Engine Updates: 2016-11-10

Version: 1.9.0, 1.9.1

New Features

- **Improved index build** – The implementation for building secondary indexes now operates by building the index in a bottom-up fashion, which eliminates unnecessary page splits. This can reduce the time needed to create an index or rebuild a table by up to 75% (based on an db.r3.8xlarge DB instance class). This feature was in lab mode in Aurora MySQL version 1.7 and is enabled by default in Aurora version 1.9 and later. For information, see [Amazon Aurora MySQL Lab Mode \(p. 762\)](#).
- **Lock compression (lab mode)** – This implementation significantly reduces the amount of memory that lock manager consumes by up to 66%. Lock manager can acquire more row locks without encountering an out-of-memory exception. This feature is disabled by default and can be activated by enabling Aurora lab mode. For information, see [Amazon Aurora MySQL Lab Mode \(p. 762\)](#).
- **Performance schema** – Aurora MySQL now includes support for performance schema with minimal impact on performance. In our testing using SysBench, enabling performance schema could degrade MySQL performance by up to 60%.

SysBench testing of an Aurora DB cluster showed an impact on performance that is 4x less than MySQL. Running the db.r3.8xlarge DB instance class resulted in 100K SQL writes/sec and over 550K SQL reads/sec, even with performance schema enabled.

- **Hot row contention improvement** – This feature reduces CPU utilization and increases throughput when a small number of hot rows are accessed by a large number of connections. This feature also eliminates `error 188` when there is hot row contention.
- **Improved out-of-memory handling** – When non-essential, locking SQL statements are executed and the reserved memory pool is breached, Aurora forces rollback of those SQL statements. This feature frees memory and prevents engine crashes due to out-of-memory exceptions.
- **Smart read selector** – This implementation improves read latency by choosing the optimal storage segment among different segments for every read, resulting in improved read throughput. SysBench testing has shown up to a 27% performance increase for write workloads .

Improvements

- Fixed an issue where an Aurora Replica encounters a shared lock during engine start up.
- Fixed a potential crash on an Aurora Replica when the read view pointer in the purge system is NULL.

Aurora MySQL Database Engine Updates: 2016-10-26

Version: 1.8.1

Improvements

- Fixed an issue where bulk inserts that use triggers that invoke AWS Lambda procedures fail.
- Fixed an issue where catalog migration fails when autocommit is off globally.
- Resolved a connection failure to Aurora when using SSL and improved Diffie-Hellman group to deal with LogJam attacks.

Integration of MySQL Bug Fixes

- OpenSSL changed the Diffie-Hellman key length parameters due to the LogJam issue. (Bug #18367167)

Aurora MySQL Database Engine Updates: 2016-10-18

Version: 1.8

New Features

- **AWS Lambda integration** – You can now asynchronously invoke an AWS Lambda function from an Aurora DB cluster using the `mysql.lambda_async` procedure. For more information, see [Invoking a Lambda Function from an Amazon Aurora MySQL DB Cluster \(p. 752\)](#).
- **Load data from Amazon S3** – You can now load text or XML files from an Amazon S3 bucket into your Aurora DB cluster using the `LOAD DATA FROM S3` or `LOAD XML FROM S3` commands. For more information, see [Loading Data into an Amazon Aurora MySQL DB Cluster from Text Files in an Amazon S3 Bucket \(p. 739\)](#).
- **Catalog migration** – Aurora now persists catalog metadata in the cluster volume to support versioning. This enables seamless catalog migration across versions and restores.
- **Cluster-level maintenance and patching** – Aurora now manages maintenance updates for an entire DB cluster. For more information, see [Maintaining an Amazon Aurora DB Cluster \(p. 408\)](#).

Improvements

- Fixed an issue where an Aurora Replica crashes when not granting a metadata lock to an inflight DDL table.
- Allowed Aurora Replicas to modify non-InnoDB tables to facilitate rotation of the slow and general log CSV files where `log_output=TABLE`.
- Fixed a lag when updating statistics from the primary instance to an Aurora Replica. Without this fix, the statistics of the Aurora Replica can get out of sync with the statistics of the primary instance and result in a different (and possibly under-performing) query plan on an Aurora Replica.
- Fixed a race condition that ensures that an Aurora Replica does not acquire locks.
- Fixed a rare scenario where an Aurora Replica that registers or de-registers with the primary instance could fail.
- Fixed a race condition that could lead to a deadlock on `db.r3.large` instances when opening or closing a volume.
- Fixed an out-of-memory issue that can occur due to a combination of a large write workload and failures in the Aurora Distributed Storage service.
- Fixed an issue with high CPU consumption because of the purge thread spinning in the presence of a long-running transaction.
- Fixed an issue when running information schema queries to get information about locks under heavy load.
- Fixed an issue with a diagnostics process that could in rare cases cause Aurora writes to storage nodes to stall and restart/fail-over.
- Fixed a condition where a successfully created table might be deleted during crash recovery if the crash occurred while a `CREATE TABLE [if not exists]` statement was being handled.
- Fixed a case where the log rotation procedure is broken when the general log and slow log are not stored on disk using catalog mitigation.
- Fixed a crash when a user creates a temporary table within a user defined function, and then uses the user defined function in the select list of the query.
- Fixed a crash that occurred when replaying GTID events. GTID is not supported by Aurora MySQL.

Integration of MySQL Bug Fixes:

- When dropping all indexes on a column with multiple indexes, InnoDB failed to block a `DROP INDEX` operation when a foreign key constraint requires an index. (Bug #16896810)
- Solve add foreign key constraint crash. (Bug #16413976)
- Fixed a crash when fetching a cursor in a stored procedure, and analyzing or flushing the table at the same time. (Bug # 18158639)

- Fixed an auto-increment bug when a user alters a table to change the AUTO_INCREMENT value to less than the maximum auto-increment column value. (Bug # 16310273)

Aurora MySQL Database Engine Updates: 2016-09-20

Version: 1.7.1

Improvements

- Fixes an issue where an Aurora Replica crashes if the InnoDB full-text search cache is full.
- Fixes an issue where the database engine crashes if a worker thread in the thread pool waits for itself.
- Fixes an issue where an Aurora Replica crashes if a metadata lock on a table causes a deadlock.
- Fixes an issue where the database engine crashes due to a race condition between two worker threads in the thread pool.
- Fixes an issue where an unnecessary failover occurs under heavy load if the monitoring agent doesn't detect the advancement of write operations to the distributed storage subsystem.

Aurora MySQL Database Engine Updates: 2016-08-30

Version: 1.7.0

New Features

- **NUMA aware scheduler** – The task scheduler for the Aurora MySQL engine is now Non-Uniform Memory Access (NUMA) aware. This minimizes cross-CPU socket contention resulting in improved performance throughput for the db.r3.8xlarge DB instance class.
- **Parallel read-ahead operates asynchronously in the background** – Parallel read-ahead has been revised to improve performance by using a dedicated thread to reduce thread contention.
- **Improved index build (lab mode)** – The implementation for building secondary indexes now operates by building the index in a bottom-up fashion, which eliminates unnecessary page splits. This can reduce the time needed to create an index or rebuild a table. This feature is disabled by default and can be activated by enabling Aurora lab mode. For information, see [Amazon Aurora MySQL Lab Mode \(p. 762\)](#).

Improvements

- Fixed an issue where establishing a connection was taking a long time if there was a surge in the number of connections requested for an instance.
- Fixed an issue where a crash occurred if ALTER TABLE was run on a partitioned table that did not use InnoDB.
- Fixed an issue where heavy write workload can cause a failover.
- Fixed an erroneous assertion that caused a failure if RENAME TABLE was run on a partitioned table.
- Improved stability when rolling back a transaction during insert-heavy workload.
- Fixed an issue where full-text search indexes were not viable on an Aurora Replica.

Integration of MySQL Bug Fixes

- Improve scalability by partitioning LOCK_grant lock. (Port WL #8355)
- Opening cursor on SELECT in stored procedure causes segfault. (Port Bug #16499751)

- MySQL gives the wrong result with some special usage. (Bug #11751794)
- Crash in GET_SEL_ARG_FOR_KEYPART – caused by patch for bug #11751794. (Bug #16208709)
- Wrong results for a simple query with GROUP BY. (Bug #17909656)
- Extra rows on semijoin query with range predicates. (Bug #16221623)
- Adding an ORDER BY clause following an IN subquery could cause duplicate rows to be returned. (Bug #16308085)
- Crash with explain for a query with loose scan for GROUP BY, MyISAM. (Bug #16222245)
- Loose index scan with quoted int predicate returns random data. (Bug #16394084)
- If the optimizer was using a loose index scan, the server could exit while attempting to create a temporary table. (Bug #16436567)
- COUNT(DISTINCT) should not count NULL values, but they were counted when the optimizer used loose index scan. (Bug #17222452)
- If a query had both MIN() MAX() and aggregate_function(DISTINCT) (for example, SUM(DISTINCT)) and was executed using loose index scan, the result values of MIN() MAX() were set improperly. (Bug #17217128)

Aurora MySQL Database Engine Updates: 2016-06-01

Version: 1.6.5

New Features

- **Efficient storage of Binary Logs** – Efficient storage of binary logs is now enabled by default for all Aurora MySQL DB clusters, and is not configurable. Efficient storage of binary logs was introduced in the April 2016 update. For more information, see [Aurora MySQL Database Engine Updates: 2016-04-06 \(p. 862\)](#).

Improvements

- Improved stability for Aurora Replicas when the primary instance is encountering a heavy workload.
- Improved stability for Aurora Replicas when running queries on partitioned tables and tables with special characters in the table name.
- Fixed connection issues when using secure connections.

Integration of MySQL Bug Fixes

- SLAVE CAN'T CONTINUE REPLICATION AFTER MASTER'S CRASH RECOVERY (Port Bug #17632285)

Aurora MySQL Database Engine Updates: 2016-04-06

Version: 1.6

This update includes the following improvements:

New Features

- **Parallel read-ahead** – Parallel read-ahead is now enabled by default for all Aurora MySQL DB clusters, and is not configurable. Parallel read-ahead was introduced in the December 2015 update. For more information, see [Aurora MySQL Database Engine Updates: 2015-12-03 \(p. 864\)](#).

In addition to enabling parallel read-ahead by default, this release includes the following improvements to parallel read-ahead:

- Improved logic so that parallel read-ahead is less aggressive, which is beneficial when your DB cluster encounters many parallel workloads.
- Improved stability on smaller tables.
- **Efficient storage of Binary Logs (lab mode)** – MySQL binary log files are now stored more efficiently in Aurora MySQL. The new storage implementation enables binary log files to be deleted much earlier and improves system performance for an instance in an Aurora MySQL DB cluster that is a binary log replication master.

To enable efficient storage of binary logs, set the `aurora_lab_mode` parameter to 1 in the parameter group for your primary instance or Aurora Replica. The `aurora_lab_mode` parameter is an instance-level parameter that is in the `default.aurora5.6` parameter group by default. For information on modifying a DB parameter group, see [Modifying Parameters in a DB Parameter Group \(p. 291\)](#). For information on parameter groups and Aurora MySQL, see [Aurora MySQL Parameters \(p. 773\)](#).

Only turn on efficient storage of binary logs for instances in an Aurora MySQL DB cluster that are MySQL binary log replication master instances.

- **AURORA_VERSION system variable** – You can now get the Aurora version of your Aurora MySQL DB cluster by querying for the `AURORA_VERSION` system variable.

To get the Aurora version, use one of the following queries:

```
select AURORA_VERSION();
select @@aurora_version;
show variables like '%version';
```

You can also see the Aurora version in the AWS Management Console when you modify a DB cluster, or by calling the [describe-db-engine-versions](#) AWS CLI command or the [DescribeDBEngineVersions](#) API operation.

- **Lock manager memory usage metric** – Information about lock manager memory usage is now available as a metric.

To get the lock manager memory usage metric, use one of the following queries:

```
show global status where variable_name in ('aurora_lockmgr_memory_used');
select * from INFORMATION_SCHEMA.GLOBAL_STATUS where variable_name in
('aurora_lockmgr_memory_used');
```

Improvements

- Improved stability during binlog and XA transaction recovery.
- Fixed a memory issue resulting from a large number of connections.
- Improved accuracy in the following metrics: Read Throughput, Read IOPS, Read Latency, Write Throughput, Write IOPS, Write Latency, and Disk Queue Depth.
- Fixed a stability issue causing slow startup for large instances after a crash.
- Improved concurrency in the data dictionary regarding synchronization mechanisms and cache eviction.
- Stability and performance improvements for Aurora Replicas:
 - Fixed a stability issue for Aurora Replicas during heavy or burst write workloads for the primary instance.

- Improved replica lag for db.r3.4xlarge and db.r3.8xlarge instances.
- Improved performance by reducing contention between application of log records and concurrent reads on an Aurora Replica.
- Fixed an issue for refreshing statistics on Aurora Replicas for newly created or updated statistics.
- Improved stability for Aurora Replicas when there are many transactions on the primary instance and concurrent reads on the Aurora Replicas across the same data.
- Improved stability for Aurora Replicas when running UPDATE and DELETE statements with JOIN statements.
- Improved stability for Aurora Replicas when running INSERT ... SELECT statements.

Integration of MySQL Bug Fixes

- BACKPORT Bug #18694052 FIX FOR ASSERTION `!M_ORDERED_REC_BUFFER' FAILED TO 5.6 (Port Bug #18305270)
- SEGV IN MEMCPY(), HA_PARTITION::POSITION (Port Bug # 18383840)
- WRONG RESULTS WITH PARTITIONING,INDEX_MERGE AND NO PK (Port Bug # 18167648)
- FLUSH TABLES FOR EXPORT: ASSERTION IN HA_PARTITION::EXTRA (Port Bug # 16943907)
- SERVER CRASH IN VIRTUAL HA_ROWS HANDLER::MULTI_RANGE_READ_INFO_CONST (Port Bug # 16164031)
- RANGE OPTIMIZER CRASHES IN SEL_ARG::RB_INSERT() (Port Bug # 16241773)

Aurora MySQL Database Engine Updates: 2016-01-11

Version: 1.5

This update includes the following improvements:

Improvements

- Fixed a 10 second pause of write operations for idle instances during Aurora storage deployments.
- Logical read-ahead now works when innodb_file_per_table is set to No. For more information on logical read-ahead, see [Aurora MySQL Database Engine Updates: 2015-12-03 \(p. 864\)](#).
- Fixed issues with Aurora Replicas reconnecting with the primary instance. This improvement also fixes an issue when you specify a large value for the quantity parameter when testing Aurora Replica failures using fault-injection queries. For more information, see [Testing an Aurora Replica Failure \(p. 640\)](#).
- Improved monitoring of Aurora Replicas falling behind and restarting.
- Fixed an issue that caused an Aurora Replica to lag, become deregistered, and then restart.
- Fixed an issue when you run the show innodb status command during a deadlock.
- Fixed an issue with failovers for larger instances during high write throughput.

Integration of MySQL Bug Fixes

- Addressed incomplete fix in MySQL full text search affecting tables where the database name begins with a digit. (Port Bug #17607956)

Aurora MySQL Database Engine Updates: 2015-12-03

Version: 1.4

This update includes the following improvements:

New Features

- **Fast Insert** – Accelerates parallel inserts sorted by primary key. For more information, see [Amazon Aurora MySQL Performance Enhancements \(p. 578\)](#).
- **Large dataset read performance** – Aurora MySQL automatically detects an IO heavy workload and launches more threads in order to boost the performance of the DB cluster. The Aurora scheduler looks into IO activity and decides to dynamically adjust the optimal number of threads in the system, quickly adjusting between IO heavy and CPU heavy workloads with low overhead.
- **Parallel read-ahead** – Improves the performance of B-Tree scans that are too large for the memory available on your primary instance or Aurora Replica (including range queries). Parallel read-ahead automatically detects page read patterns and pre-fetches pages into the buffer cache in advance of when they are needed. Parallel read-ahead works multiple tables at the same time within the same transaction.

Improvements:

- Fixed brief Aurora database availability issues during Aurora storage deployments.
- Correctly enforce the `max_connection` limit.
- Improve binlog purging where Aurora is the binlog master and the database is restarting after a heavy data load.
- Fixed memory management issues with the table cache.
- Add support for huge pages in shared memory buffer cache for faster recovery.
- Fixed an issue with thread-local storage not being initialized.
- Allow 16K connections by default.
- Dynamic thread pool for IO heavy workloads.
- Fixed an issue with properly invalidating views involving UNION in the query cache.
- Fixed a stability issue with the dictionary stats thread.
- Fixed a memory leak in the dictionary subsystem related to cache eviction.
- Fixed high read latency issue on Aurora Replicas when there is very low write load on the master.
- Fixed stability issues on Aurora Replicas when performing operations on DDL partitioned tables such as `ALTER TABLE ... REORGANIZE PARTITION` on the master.
- Fixed stability issues on Aurora Replicas during volume growth.
- Fixed performance issue for scans on non-clustered indexes in Aurora Replicas.
- Fix stability issue that makes Aurora Replicas lag and eventually get deregistered and re-started.

Integration of MySQL Bug Fixes

- SEGV in FTSPARSE(). (Bug #16446108)
- InnoDB data dictionary is not updated while renaming the column. (Bug #19465984)
- FTS crash after renaming table to different database. (Bug #16834860)
- Failed preparing of trigger on truncated tables cause error 1054. (Bug #18596756)
- Metadata changes might cause problems with trigger execution. (Bug #18684393)
- Materialization is not chosen for long UTF8 VARCHAR field. (Bug #17566396)
- Poor execution plan when ORDER BY with limit X. (Bug #16697792)
- Backport bug #11765744 TO 5.1, 5.5 AND 5.6. (Bug #17083851)

- Mutex issue in SQL/SQL_SHOW.CC resulting in SIG6. Source likely FILL_VARIABLES. (Bug #20788853)
- Backport bug #18008907 to 5.5+ versions. (Bug #18903155)
- Adapt fix for a stack overflow error in MySQL 5.7. (Bug #19678930)

Aurora MySQL Database Engine Updates: 2015-10-16

Versions: 1.2, 1.3

This update includes the following improvements:

Fixes

- Resolved out-of-memory issue in the new lock manager with long-running transactions
- Resolved security vulnerability when replicating with non-RDS MySQL databases
- Updated to ensure that quorum writes retry correctly with storage failures
- Updated to report replica lag more accurately
- Improved performance by reducing contention when many concurrent transactions are trying to modify the same row
- Resolved query cache invalidation for views that are created by joining two tables
- Disabled query cache for transactions with UNCOMMITTED_READ isolation

Improvements

- Better performance for slow catalog queries on warm caches
- Improved concurrency in dictionary statistics
- Better stability for the new query cache resource manager, extent management, files stored in Amazon Aurora smart storage, and batch writes of log records

Integration of MySQL Bug Fixes

- Killing a query inside innodb causes it to eventually crash with an assertion. (Bug #1608883)
- For failure to create a new thread for the event scheduler, event execution, or new connection, no message was written to the error log. (Bug #16865959)
- If one connection changed its default database and simultaneously another connection executed SHOW PROCESSLIST, the second connection could access invalid memory when attempting to display the first connection's default database memory. (Bug #11765252)
- PURGE BINARY LOGS by design does not remove binary log files that are in use or active, but did not provide any notice when this occurred. (Bug #13727933)
- For some statements, memory leaks could result when the optimizer removed unneeded subquery clauses. (Bug #15875919)
- During shutdown, the server could attempt to lock an uninitialized mutex. (Bug #16016493)
- A prepared statement that used GROUP_CONCAT() and an ORDER BY clause that named multiple columns could cause the server to exit. (Bug #16075310)
- Performance Schema instrumentation was missing for slave worker threads. (Bug #16083949)
- STOP SLAVE could cause a deadlock when issued concurrently with a statement such as SHOW STATUS that retrieved the values for one or more of the status variables Slave_retried_transactions, Slave_heartbeat_period, Slave_received_heartbeats, Slave_last_heartbeat, or Slave_running. (Bug #16088188)

- A full-text query using Boolean mode could return zero results in some cases where the search term was a quoted phrase. (Bug #16206253)
- The optimizer's attempt to remove redundant subquery clauses raised an assertion when executing a prepared statement with a subquery in the ON clause of a join in a subquery. (Bug #16318585)
- GROUP_CONCAT unstable, crash in ITEM_SUM::CLEAN_UP_AFTER_REMOVAL. (Bug #16347450)
- Attempting to replace the default InnoDB full-text search (FTS) stopword list by creating an InnoDB table with the same structure as INFORMATION_SCHEMA.INNODB_FT_DEFAULT_STOPWORD would result in an error. (Bug #16373868)
- After the client thread on a slave performed a FLUSH TABLES WITH READ LOCK and was followed by some updates on the master, the slave hung when executing SHOW SLAVE STATUS. (Bug #16387720)
- When parsing a delimited search string such as "abc-def" in a full-text search, InnoDB now uses the same word delimiters as MyISAM. (Bug #16419661)
- Crash in FTS_AST_TERM_SET_WILDCARD. (Bug #16429306)
- SEGFAULT in FTS_AST_VISIT() for FTS RQG test. (Bug # 16435855)
- For debug builds, when the optimizer removed an Item_ref pointing to a subquery, it caused a server exit. (Bug #16509874)
- Full-text search on InnoDB tables failed on searches for literal phrases combined with + or - operators. (Bug #16516193)
- START SLAVE failed when the server was started with the options --master-info-repository=TABLE relay-log-info-repository=TABLE and with autocommit set to 0, together with --skip-slave-start. (Bug #16533802)
- Very large InnoDB full-text search (FTS) results could consume an excessive amount of memory. (Bug #16625973)
- In debug builds, an assertion could occur in OPT_CHECK_ORDER_BY when using binary directly in a search string, as binary might include NULL bytes and other non-meaningful characters. (Bug #16766016)
- For some statements, memory leaks could result when the optimizer removed unneeded subquery clauses. (Bug #16807641)
- It was possible to cause a deadlock after issuing FLUSH TABLES WITH READ LOCK by issuing STOP SLAVE in a new connection to the slave, then issuing SHOW SLAVE STATUS using the original connection. (Bug #16856735)
- GROUP_CONCAT() with an invalid separator could cause a server exit. (Bug #16870783)
- The server did excessive locking on the LOCK_active_mi and active_mi->rli->data_lock mutexes for any SHOW STATUS LIKE 'pattern' statement, even when the pattern did not match status variables that use those mutexes (Slave_heartbeat_period, Slave_last_heartbeat, Slave_received_heartbeats, Slave_retried_transactions, Slave_running). (Bug #16904035)
- A full-text search using the IN BOOLEAN MODE modifier would result in an assertion failure. (Bug #16927092)
- Full-text search on InnoDB tables failed on searches that used the + boolean operator. (Bug #17280122)
- 4-way deadlock: zombies, purging binlogs, show processlist, show binlogs. (Bug #17283409)
- When an SQL thread which was waiting for a commit lock was killed and restarted it caused a transaction to be skipped on slave. (Bug #17450876)
- An InnoDB full-text search failure would occur due to an "unended" token. The string and string length should be passed for string comparison. (Bug #17659310)
- Large numbers of partitioned InnoDB tables could consume much more memory when used in MySQL 5.6 or 5.7 than the memory used by the same tables used in previous releases of the MySQL Server. (Bug #17780517)
- For full-text queries, a failure to check that num_token is less than max_proximity_item could result in an assertion. (Bug #18233051)

- Certain queries for the INFORMATION_SCHEMA TABLES and COLUMNS tables could lead to excessive memory use when there were large numbers of empty InnoDB tables. (Bug #18592390)
- When committing a transaction, a flag is now used to check whether a thread has been created, rather than checking the thread itself, which uses more resources, particularly when running the server with master_info_repository=TABLE. (Bug #18684222)
- If a client thread on a slave executed FLUSH TABLES WITH READ LOCK while the master executed a DML, executing SHOW SLAVE STATUS in the same client became blocked, causing a deadlock. (Bug #19843808)
- Ordering by a GROUP_CONCAT() result could cause a server exit. (Bug #19880368)

Aurora MySQL Database Engine Updates: 2015-08-24

Version: 1.1

This update includes the following improvements:

- Replication stability improvements when replicating with a MySQL database (binlog replication). For information on Aurora MySQL replication with MySQL, see [Replication with Amazon Aurora \(p. 47\)](#).
- A 1 gigabyte (GB) limit on the size of the relay logs accumulated for an Aurora MySQL DB cluster that is a replication slave. This improves the file management for the Aurora DB clusters.
- Stability improvements in the areas of read ahead, recursive foreign-key relationships, and Aurora replication.
- Integration of MySQL bug fixes.
 - InnoDB databases with names beginning with a digit cause a full-text search (FTS) parser error. (Bug #17607956)
 - InnoDB full-text searches fail in databases whose names began with a digit. (Bug #17161372)
 - For InnoDB databases on Windows, the full-text search (FTS) object ID is not in the expected hexadecimal format. (Bug #16559254)
 - A code regression introduced in MySQL 5.6 negatively impacted DROP TABLE and ALTER TABLE performance. This could cause a performance drop between MySQL Server 5.5.x and 5.6.x. (Bug #16864741)
 - Simplified logging to reduce the size of log files and the amount of storage that they require.

MySQL Bugs Fixed by Aurora MySQL Database Engine Updates

The following sections identify MySQL bugs that have been fixed by Aurora MySQL database engine updates.

Topics

- [MySQL Bugs Fixed by Aurora MySQL 2.x Database Engine Updates \(p. 868\)](#)
- [MySQL Bugs Fixed by Aurora MySQL 1.x Database Engine Updates \(p. 869\)](#)

MySQL Bugs Fixed by Aurora MySQL 2.x Database Engine Updates

MySQL 5.7-compatible version Aurora contains all MySQL bug fixes through MySQL 5.7.12. The following table identifies additional MySQL bugs that have been fixed by Aurora MySQL database engine updates, and which update they were fixed in.

Database engine update	Version	MySQL bugs fixed
Aurora MySQL Database Engine Updates 2019-11-25 (Version 2.07.0) (p. 798)	2.07.0	<ul style="list-style-type: none"> Bug #26251621: INCORRECT BEHAVIOR WITH TRIGGER AND GCOL Bug #22574695: ASSERTION `!TABLE (!TABLE->READ_SET BITMAP_IS_SET(TABLE->READ_SET, FIELD))` Bug #25966845: INSERT ON DUPLICATE KEY GENERATE A DEADLOCK Bug #23070734: CONCURRENT TRUNCATE TABLES CAUSE STALL Bug #26191879: FOREIGN KEY CASCADES USE EXCESSIVE MEMORY Bug #20989615: INNODB AUTO_INCREMENT PRODUCES SAME VALUE TWICE
Aurora MySQL Database Engine Updates 2019-11-11 (Version 2.05.0) (p. 803)	2.05.0	<ul style="list-style-type: none"> Bug#23054591: PURGE BINARY LOGS TO is reading the whole binlog file and causing MySql to Stall
Aurora MySQL Database Engine Updates 2019-09-19 (Version 2.04.6) (p. 808)	2.04.6	<ul style="list-style-type: none"> Bug#23054591: PURGE BINARY LOGS TO is reading the whole binlog file and causing MySql to Stall
Aurora MySQL Database Engine Updates 2018-10-11 (p. 819)	2.03	<ul style="list-style-type: none"> REVERSE SCAN ON A PARTITIONED TABLE DOES ICP - ORDER BY DESC (Bug #24929748). JSON_OBJECT CREATES INVALID JSON CODE (Bug#26867509). INSERTING LARGE JSON DATA TAKES AN INORDINATE AMOUNT OF TIME (Bug #22843444). PARTITIONED TABLES USE MORE MEMORY IN 5.7 THAN 5.6 (Bug #25080442).
Aurora MySQL Database Engine Updates 2018-05-03 (p. 825)	2.02	Left join returns incorrect results on the outer side (Bug #22833364)
Aurora MySQL Database Engine Updates 2019-05-02 (Version 2.04.2) (p. 814)	2.04.2	Bug #24829050 - INDEX_MERGE_INTERSECTION OPTIMIZATION CAUSES WRONG QUERY RESULTS

MySQL Bugs Fixed by Aurora MySQL 1.x Database Engine Updates

MySQL 5.6-compatible version Aurora contains all MySQL bug fixes through MySQL 5.6.10. The following table identifies additional MySQL bugs that have been fixed by Aurora MySQL database engine updates, and which update they were fixed in.

Database engine update	Version	MySQL bugs fixed
Aurora MySQL Database Engine Updates: 2015-08-24 (p. 868)	1.1	<ul style="list-style-type: none"> InnoDB databases with names beginning with a digit cause a full-text search (FTS) parser error. (Bug #17607956) InnoDB full-text searches fail in databases whose names began with a digit. (Bug #17161372) For InnoDB databases on Windows, the full-text search (FTS) object ID is not in the expected hexadecimal format. (Bug #16559254) A code regression introduced in MySQL 5.6 negatively impacted DROP TABLE and ALTER TABLE performance. This could cause a performance drop between MySQL Server 5.5.x and 5.6.x. (Bug #16864741)
Aurora MySQL Database Engine Updates: 2015-10-16 (p. 866)	1.2, 1.3	<ul style="list-style-type: none"> Killing a query inside innodb causes it to eventually crash with an assertion. (Bug #1608883) For failure to create a new thread for the event scheduler, event execution, or new connection, no message was written to the error log. (Bug #16865959) If one connection changed its default database and simultaneously another connection executed SHOW PROCESSLIST, the second connection could access invalid memory when attempting to display the first connection's default database memory. (Bug #11765252) PURGE BINARY LOGS by design does not remove binary log files that are in use or active, but did not provide any notice when this occurred. (Bug #13727933) For some statements, memory leaks could result when the optimizer removed unneeded subquery clauses. (Bug #15875919) During shutdown, the server could attempt to lock an uninitialized mutex. (Bug #16016493) A prepared statement that used GROUP_CONCAT() and an ORDER BY clause that named multiple columns could cause the server to exit. (Bug #16075310) Performance Schema instrumentation was missing for slave worker threads. (Bug #16083949) STOP SLAVE could cause a deadlock when issued concurrently with a statement such as SHOW STATUS that retrieved the values for one or more of the status variables Slave_retried_transactions, Slave_heartbeat_period, Slave_received_heartbeats, Slave_last_heartbeat, or Slave_running. (Bug #16088188) A full-text query using Boolean mode could return zero results in some cases where the search term was a quoted phrase. (Bug #16206253) The optimizer's attempt to remove redundant subquery clauses raised an assertion when executing a prepared statement with a subquery in the ON clause of a join in a subquery. (Bug #16318585) GROUP_CONCAT unstable, crash in ITEM_SUM::CLEAN_UP_AFTER_REMOVAL. (Bug #16347450)

Database engine update	Version	MySQL bugs fixed
		<ul style="list-style-type: none"> • Attempting to replace the default InnoDB full-text search (FTS) stopword list by creating an InnoDB table with the same structure as INFORMATION_SCHEMA.INNODB_FT_DEFAULT_STOPWORD would result in an error. (Bug #16373868) • After the client thread on a slave performed a FLUSH TABLES WITH READ LOCK and was followed by some updates on the master, the slave hung when executing SHOW SLAVE STATUS. (Bug #16387720) • When parsing a delimited search string such as "abc-def" in a full-text search, InnoDB now uses the same word delimiters as MyISAM. (Bug #16419661) • Crash in FTS_AST_TERM_SET_WILDCARD. (Bug #16429306) • SEGFAULT in FTS_AST_VISIT() for FTS RQG test. (Bug # 16435855) • For debug builds, when the optimizer removed an Item_ref pointing to a subquery, it caused a server exit. (Bug #16509874) • Full-text search on InnoDB tables failed on searches for literal phrases combined with + or - operators. (Bug #16516193) • START SLAVE failed when the server was started with the options --master-info-repository=TABLE relay-log-info-repository=TABLE and with autocommit set to 0, together with --skip-slave-start. (Bug #16533802) • Very large InnoDB full-text search (FTS) results could consume an excessive amount of memory. (Bug #16625973) • In debug builds, an assertion could occur in OPT_CHECK_ORDER_BY when using binary directly in a search string, as binary might include NULL bytes and other non-meaningful characters. (Bug #16766016) • For some statements, memory leaks could result when the optimizer removed unneeded subquery clauses. (Bug #16807641) • It was possible to cause a deadlock after issuing FLUSH TABLES WITH READ LOCK by issuing STOP SLAVE in a new connection to the slave, then issuing SHOW SLAVE STATUS using the original connection. (Bug #16856735) • GROUP_CONCAT() with an invalid separator could cause a server exit. (Bug #16870783) • The server did excessive locking on the LOCK_active_mi and active_mi->rli->data_lock mutexes for any SHOW STATUS LIKE 'pattern' statement, even when the pattern did not match status variables that use those mutexes (Slave_heartbeat_period, Slave_last_heartbeat, Slave_received_heartbeats, Slave_retried_transactions, Slave_running). (Bug #16904035) • A full-text search using the IN BOOLEAN MODE modifier would result in an assertion failure. (Bug #16927092) • Full-text search on InnoDB tables failed on searches that used the + boolean operator. (Bug #17280122) • 4-way deadlock: zombies, purging binlogs, show processlist, show binlogs. (Bug #17283409)

Database engine update	Version	MySQL bugs fixed
		<ul style="list-style-type: none"> When an SQL thread which was waiting for a commit lock was killed and restarted it caused a transaction to be skipped on slave. (Bug #17450876) An InnoDB full-text search failure would occur due to an "unended" token. The string and string length should be passed for string comparison. (Bug #17659310) Large numbers of partitioned InnoDB tables could consume much more memory when used in MySQL 5.6 or 5.7 than the memory used by the same tables used in previous releases of the MySQL Server. (Bug #17780517) For full-text queries, a failure to check that num_token is less than max_proximity_item could result in an assertion. (Bug #18233051) Certain queries for the INFORMATION_SCHEMA TABLES and COLUMNS tables could lead to excessive memory use when there were large numbers of empty InnoDB tables. (Bug #18592390) When committing a transaction, a flag is now used to check whether a thread has been created, rather than checking the thread itself, which uses more resources, particularly when running the server with master_info_repository=TABLE. (Bug #18684222) If a client thread on a slave executed FLUSH TABLES WITH READ LOCK while the master executed a DML, executing SHOW SLAVE STATUS in the same client became blocked, causing a deadlock. (Bug #19843808) Ordering by a GROUP_CONCAT() result could cause a server exit. (Bug #19880368)
Aurora MySQL Database Engine Updates: 2015-12-03 (p. 864)	1.4	<ul style="list-style-type: none"> SEGV in FTSPARSE(). (Bug #16446108) InnoDB data dictionary is not updated while renaming the column. (Bug #19465984) FTS crash after renaming table to different database. (Bug #16834860) Failed preparing of trigger on truncated tables cause error 1054. (Bug #18596756) Metadata changes might cause problems with trigger execution. (Bug #18684393) Materialization is not chosen for long UTF8 VARCHAR field. (Bug #17566396) Poor execution plan when ORDER BY with limit X. (Bug #16697792) Backport bug #11765744 TO 5.1, 5.5 AND 5.6. (Bug #17083851) Mutex issue in SQL/SQL_SHOW.CC resulting in SIG6. Source likely FILL_VARIABLES. (Bug #20788853) Backport bug #18008907 to 5.5+ versions. (Bug #18903155) Adapt fix for a stack overflow error in MySQL 5.7. (Bug #19678930)

Database engine update	Version	MySQL bugs fixed
Aurora MySQL Database Engine Updates: 2016-01-11 (p. 864)	1.5	<ul style="list-style-type: none"> Addressed incomplete fix in MySQL full text search affecting tables where the database name begins with a digit. (Port Bug #17607956)
Aurora MySQL Database Engine Updates: 2016-04-06 (p. 862)	1.6	<ul style="list-style-type: none"> BACKPORT Bug #18694052 FIX FOR ASSERTION `!M_ORDERED_REC_BUFFER' FAILED TO 5.6 (Port Bug #18305270) SEGV IN MEMCPY(), HA_PARTITION::POSITION (Port Bug # 18383840) WRONG RESULTS WITH PARTITIONING,INDEX_MERGE AND NO PK (Port Bug # 18167648) FLUSH TABLES FOR EXPORT: ASSERTION IN HA_PARTITION::EXTRA (Port Bug # 16943907) SERVER CRASH IN VIRTUAL HA_ROWS HANDLER::MULTI_RANGE_READ_INFO_CONST (Port Bug # 16164031) RANGE OPTIMIZER CRASHES IN SEL_ARG::RB_INSERT() (Port Bug # 16241773)
Aurora MySQL Database Engine Updates: 2016-06-01 (p. 862)	1.6.5	<ul style="list-style-type: none"> SLAVE CAN'T CONTINUE REPLICATION AFTER MASTER'S CRASH RECOVERY (Port Bug #17632285)

Database engine update	Version	MySQL bugs fixed
Aurora MySQL Database Engine Updates: 2016-08-30 (p. 861)	1.7	<ul style="list-style-type: none"> • Improve scalability by partitioning LOCK_grant lock. (Port WL #8355) • Opening cursor on SELECT in stored procedure causes segfault. (Port Bug #16499751) • MySQL gives the wrong result with some special usage. (Bug #11751794) • Crash in GET_SEL_ARG_FOR_KEYPART – caused by patch for bug #11751794. (Bug #16208709) • Wrong results for a simple query with GROUP BY. (Bug #17909656) • Extra rows on semijoin query with range predicates. (Bug #16221623) • Adding an ORDER BY clause following an IN subquery could cause duplicate rows to be returned. (Bug #16308085) • Crash with explain for a query with loose scan for GROUP BY, MyISAM. (Bug #16222245) • Loose index scan with quoted int predicate returns random data. (Bug #16394084) • If the optimizer was using a loose index scan, the server could exit while attempting to create a temporary table. (Bug #16436567) • COUNT(DISTINCT) should not count NULL values, but they were counted when the optimizer used loose index scan. (Bug #17222452) • If a query had both MIN() MAX() and aggregate_function(DISTINCT) (for example, SUM(DISTINCT)) and was executed using loose index scan, the result values of MIN() MAX() were set improperly. (Bug #17217128)
Aurora MySQL Database Engine Updates: 2016-10-18 (p. 859)	1.8	<ul style="list-style-type: none"> • When dropping all indexes on a column with multiple indexes, InnoDB failed to block a DROP INDEX operation when a foreign key constraint requires an index. (Bug #16896810) • Solve add foreign key constraint crash. (Bug #16413976) • Fixed a crash when fetching a cursor in a stored procedure, and analyzing or flushing the table at the same time. (Bug # 18158639) • Fixed an auto-increment bug when a user alters a table to change the AUTO_INCREMENT value to less than the maximum auto-increment column value. (Bug # 16310273)
Aurora MySQL Database Engine Updates: 2016-10-26 (p. 859)	1.8.1	<ul style="list-style-type: none"> • OpenSSL changed the Diffie-Hellman key length parameters due to the LogJam issue. (Bug #18367167)

Database engine update	Version	MySQL bugs fixed
Aurora MySQL Database Engine Updates: 2016-12-14 (p. 857)	1.10	<ul style="list-style-type: none"> UNION of derived tables returns wrong results with '1=0/false'-clauses. (Bug #69471) Server crashes in ITEM_FUNC_GROUP_CONCAT::FIX_FIELDS on 2nd execution of stored procedure. (Bug #20755389) Avoid MySQL queries from stalling for too long during FTS cache sync to disk by offloading the cache sync task to a separate thread, as soon as the cache size crosses 10% of the total size. (Bugs #22516559, #73816)
Aurora MySQL Database Engine Updates: 2017-02-23 (p. 855)	1.11	<ul style="list-style-type: none"> Running ALTER table DROP foreign key simultaneously with another DROP operation causes the table to disappear. (Bug #16095573) Some INFORMATION_SCHEMA queries that used ORDER BY did not use a filesort optimization as they did previously. (Bug #16423536) FOUND_ROWS () returns the wrong count of rows on a table. (Bug #68458) The server fails instead of giving an error when too many temp tables are open. (Bug #18948649)
Aurora MySQL Database Engine Updates: 2017-04-05 (p. 853)	1.12	<ul style="list-style-type: none"> Reloading a table that was evicted while empty caused an AUTO_INCREMENT value to be reset. (Bug #21454472, Bug #77743) An index record was not found on rollback due to inconsistencies in the purge_node_t structure. The inconsistency resulted in warnings and error messages such as "error in sec index entry update", "unable to purge a record", and "tried to purge sec index entry not marked for deletion". (Bug #19138298, Bug #70214, Bug #21126772, Bug #21065746) Wrong stack size calculation for qsort operation leads to stack overflow. (Bug #73979) Record not found in an index upon rollback. (Bug #70214, Bug #72419) ALTER TABLE add column TIMESTAMP on update CURRENT_TIMESTAMP inserts ZERO-datas (Bug #17392)
Aurora MySQL Database Engine Updates: 2017-05-15 (p. 852)	1.13	<ul style="list-style-type: none"> Reloading a table that was evicted while empty caused an AUTO_INCREMENT value to be reset. (Bug #21454472, Bug #77743) An index record was not found on rollback due to inconsistencies in the purge_node_t structure. The inconsistency resulted in warnings and error messages such as "error in sec index entry update", "unable to purge a record", and "tried to purge sec index entry not marked for deletion". (Bug #19138298, Bug #70214, Bug #21126772, Bug #21065746) Wrong stack size calculation for qsort operation leads to stack overflow. (Bug #73979) Record not found in an index upon rollback. (Bug #70214, Bug #72419) ALTER TABLE add column TIMESTAMP on update CURRENT_TIMESTAMP inserts ZERO-datas (Bug #17392)

Database engine update	Version	MySQL bugs fixed
Aurora MySQL Database Engine Updates: 2017-08-07 (p. 851)	1.14	A full-text search combined with derived tables (subqueries in the <code>FROM</code> clause) caused a server exit. Now, if a full-text operation depends on a derived table, the server produces an error indicating that a full-text search cannot be done on a materialized table. (Bug #68751, Bug #16539903)
Aurora MySQL Database Engine Updates: 2018-03-13 (p. 850)	1.14.4	<ul style="list-style-type: none"> Ignorable events don't work and are not tested (Bug #74683) <code>NEW->OLD ASSERT FAILURE 'GTID_MODE > 0'</code> (Bug #20436436)
Aurora MySQL Database Engine Updates 2017-10-24 (p. 848)	1.15	<ul style="list-style-type: none"> <code>CREATE USER</code> accepts plugin and password hash, but ignores the password hash (Bug #78033) The partitioning engine adds fields to the read bit set to be able to return entries sorted from a partitioned index. This leads to the join buffer will try to read unneeded fields. Fixed by not adding all partitioning fields to the <code>read_set</code>, but instead only sort on the already set prefix fields in the <code>read_set</code>. Added a <code>DBUG_ASSERT</code> that if doing <code>key_cmp</code>, at least the first field must be read (Bug #16367691). MySQL instance stalling "doing SYNC index" (Bug #73816) Assert <code>RBT_EMPTY(INDEX_CACHE->WORDS)</code> in <code>ALTER TABLE CHANGE COLUMN</code> (Bug #17536995) InnoDB Fulltext search doesn't find records when savepoints are involved (Bug #70333)
Aurora MySQL Database Engine Updates 2017-11-20 (p. 847)	1.15.1	<ul style="list-style-type: none"> Reverted — MySQL instance stalling "doing SYNC index" (Bug #73816) Reverted — Assert <code>RBT_EMPTY(INDEX_CACHE->WORDS)</code> in <code>ALTER TABLE CHANGE COLUMN</code> (Bug #17536995) Reverted — InnoDB Fulltext search doesn't find records when savepoints are involved (Bug #70333)
Aurora MySQL Database Engine Updates 2018-03-13 (p. 845)	1.17	<ul style="list-style-type: none"> <code>LAST_INSERT_ID</code> is replicated incorrectly if replication filters are used (Bug #69861) Query returns different results depending on whether <code>INDEX_MERGE</code> setting (Bug #16862316) Query proc re-execute of stored routine, inefficient query plan (Bug #16346367) InnoDB FTS : Assert in <code>FTS_CACHE_APPEND_DELETED_DOC_IDS</code> (Bug #18079671) Assert <code>RBT_EMPTY(INDEX_CACHE->WORDS)</code> in <code>ALTER TABLE CHANGE COLUMN</code> (Bug #17536995) InnoDB fulltext search doesn't find records when savepoints are involved (Bug #70333, Bug #17458835)
Aurora MySQL Database Engine Updates 2018-09-06 (p. 842)	1.17.6	<ul style="list-style-type: none"> For an <code>ALTER TABLE</code> statement that renamed or changed the default value of a <code>BINARY</code> column, the alteration was done using a table copy and not in place. (Bug #67141, Bug #14735373, Bug #69580, Bug #17024290) An outer join between a regular table and a derived table that is implicitly grouped could cause a server exit. (Bug #16177639)

Database engine update	Version	MySQL bugs fixed
Aurora MySQL Database Engine Updates 2018-10-08 (p. 841)	1.17.7	<ul style="list-style-type: none"> Drop index on a foreign key column leads to missing table. (Bug #16208542) Memory leak in add_derived_key(). (Bug #76349) For partitioned tables, queries could return different results depending on whether Index Merge was used. (Bug #16862316) Queries using the index_merge optimization (see Index Merge Optimization) could return invalid results when run against tables that were partitioned by HASH. (Bug #17588348)
Aurora MySQL Database Engine Updates 2019-01-17 (p. 840)	1.17.8	<ul style="list-style-type: none"> BUG #13418638: CREATE TABLE IF NOT EXISTS METADATA LOCK IS TOO RESTRICTIVE
Aurora MySQL Database Engine Updates 2019-02-07 (Version 1.19.0) (p. 838)	1.19.0	<ul style="list-style-type: none"> BUG #32917: DETECT ORPHAN TEMP-POOL FILES, AND HANDLE GRACEFULLY BUG #63144 CREATE TABLE IF NOT EXISTS METADATA LOCK IS TOO RESTRICTIVE
Aurora MySQL Database Engine Updates 2019-09-19 (Version 1.19.5) (p. 836)	1.19.5	<ul style="list-style-type: none"> CVE-2018-2696 CVE-2015-4737 Bug #19929406: HANDLE_FATAL_SIGNAL (SIG=11) IN __MEMMOVE_SSSE3_BACK FROM STRING::COPY Bug #17059925: For UNION statements, the rows-examined value was calculated incorrectly. This was manifest as too-large values for the ROWS_EXAMINED column of Performance Schema statement tables (such as events_statements_current). Bug #11827369: Some queries with SELECT ... FROM DUAL nested subqueries raised an assertion. Bug #16311231: Incorrect results were returned if a query contained a subquery in an IN clause which contained an XOR operation in the WHERE clause.
Aurora MySQL Database Engine Updates 2019-11-11 (Version 1.20.0) (p. 835)	1.20.0	<ul style="list-style-type: none"> Bug #19929406: HANDLE_FATAL_SIGNAL (SIG=11) IN __MEMMOVE_SSSE3_BACK FROM STRING::COPY Bug #17059925: For UNION statements, the rows-examined value was calculated incorrectly. This was manifested as too-large values for the ROWS_EXAMINED column of Performance Schema statement tables (such as events_statements_current). Bug #11827369: Some queries with SELECT ... FROM DUAL nested subqueries raised an assertion. Bug #16311231: Incorrect results were returned if a query contained a subquery in an IN clause that contained an XOR operation in the WHERE clause.

Database engine update	Version	MySQL bugs fixed
Aurora MySQL Database Engine Updates 2019-11-25 (Version 1.21.0) (p. 833)	1.21.0	<ul style="list-style-type: none"> Bug #19929406: HANDLE_FATAL_SIGNAL (SIG=11) IN __MEMMOVE_SSSE3_BACK FROM STRING::COPY Bug #17059925: For UNION statements, the rows-examined value was calculated incorrectly. This was manifested as too-large values for the ROWS_EXAMINED column of Performance Schema statement tables (such as events_statements_current). Bug #11827369: Some queries with SELECT ... FROM DUAL nested subqueries raised an assertion. Bug #16311231: Incorrect results were returned if a query contained a subquery in an IN clause that contained an XOR operation in the WHERE clause.
Aurora MySQL Database Engine Updates 2019-11-25 (Version 1.22.0) (p. 831)	1.22.0	<ul style="list-style-type: none"> Bug#16346241 - SERVER CRASH IN ITEM_PARAM::QUERY_VAL_STR Bug#17733850 - NAME_CONST() CRASH IN ITEM_NAME_CONST::ITEM_NAME_CONST() Bug #20989615 - INNODB AUTO_INCREMENT PRODUCES SAME VALUE TWICE Bug #20181776 - ACCESS CONTROL DOESN'T MATCH MOST SPECIFIC HOST WHEN IT CONTAINS WILDCARD Bug #27326796 - MYSQL CRASH WITH INNODB ASSERTION FAILURE IN FILE PARSONS.CC Bug #20590013 - IF YOU HAVE A FULLTEXT INDEX AND DROP IT YOU CAN NO LONGER PERFORM ONLINE DDL

Working with Amazon Aurora PostgreSQL

Amazon Aurora PostgreSQL is a fully managed, PostgreSQL-compatible, and ACID-compliant relational database engine that combines the speed and reliability of high-end commercial databases with the simplicity and cost-effectiveness of open-source databases. Aurora PostgreSQL is a drop-in replacement for PostgreSQL and makes it simple and cost-effective to set up, operate, and scale your new and existing PostgreSQL deployments, thus freeing you to focus on your business and applications. Amazon RDS provides administration for Aurora by handling routine database tasks such as provisioning, patching, backup, recovery, failure detection, and repair. Amazon RDS also provides push-button migration tools to convert your existing Amazon RDS for PostgreSQL applications to Aurora PostgreSQL.

Aurora PostgreSQL can work with many industry standards. For example, you can use Aurora PostgreSQL databases to build HIPAA-compliant applications and to store healthcare related information, including protected health information (PHI), under an executed 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 \(p. 879\)](#)
- [Updating Applications to Connect to Aurora PostgreSQL DB Clusters Using New SSL/TLS Certificates \(p. 883\)](#)
- [Migrating Data to Amazon Aurora with PostgreSQL Compatibility \(p. 887\)](#)
- [Managing Amazon Aurora PostgreSQL \(p. 910\)](#)
- [Replication with Amazon Aurora PostgreSQL \(p. 911\)](#)
- [Integrating Amazon Aurora PostgreSQL with Other AWS Services \(p. 916\)](#)
- [Managing Query Execution Plans for Aurora PostgreSQL \(p. 916\)](#)
- [Publishing Aurora PostgreSQL Logs to Amazon CloudWatch Logs \(p. 943\)](#)
- [Fast Recovery After Failover with Cluster Cache Management for Aurora PostgreSQL \(p. 946\)](#)
- [Upgrading an Aurora PostgreSQL DB Cluster Engine Version \(p. 950\)](#)
- [Best Practices with Amazon Aurora PostgreSQL \(p. 953\)](#)
- [Amazon Aurora PostgreSQL Reference \(p. 960\)](#)
- [Database Engine Updates for Amazon Aurora PostgreSQL \(p. 970\)](#)

Security with Amazon Aurora PostgreSQL

Security for Amazon Aurora PostgreSQL is managed at three levels:

- To control who can perform Amazon RDS management actions on Aurora DB clusters and DB instances, you use AWS Identity and Access Management (IAM). When you connect to AWS using IAM credentials, your IAM account must have IAM policies that grant the permissions required to perform Amazon RDS management operations. For more information, see [Identity and Access Management in Amazon Aurora \(p. 191\)](#).

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

- Aurora DB clusters must be created in an Amazon Virtual Private Cloud (VPC). To control which devices and Amazon EC2 instances can open connections to the endpoint and port of the DB instance for Aurora DB clusters in a VPC, you use a VPC security group. These endpoint and port connections can be made using Secure Sockets Layer (SSL). In addition, firewall rules at your company can control whether devices running at your company can open connections to a DB instance. For more information on VPCs, see [Amazon Virtual Private Cloud VPCs and Amazon Aurora \(p. 239\)](#).

Aurora PostgreSQL supports db.r4 and db.t3 instance classes with default VPC only. With default VPC tenancy, the VPC runs on shared hardware. With dedicated VPC tenancy, the VPC runs on a dedicated hardware instance. For more information about instance classes, see [Choosing the DB Instance Class \(p. 76\)](#). For more information about default and dedicated VPC tenancy, see [Dedicated Instances in the Amazon EC2 User Guide for Linux Instances](#).

- To authenticate login and permissions for an Amazon Aurora DB cluster, you can take the same approach as with a stand-alone instance of PostgreSQL.

Commands such as `CREATE ROLE`, `ALTER ROLE`, `GRANT`, and `REVOKE` work just as they do in on-premises databases, as does directly modifying database schema tables. For more information, see [Client Authentication](#) in the PostgreSQL documentation.

Note

The Salted Challenge Response Authentication Mechanism (SCRAM) is not supported with Aurora PostgreSQL.

When you create an Amazon Aurora PostgreSQL DB instance, the master user has the following default privileges:

- `LOGIN`
- `NOSUPERUSER`
- `INHERIT`
- `CREATEDB`
- `CREATEROLE`
- `NOREPLICATION`
- `VALID UNTIL 'infinity'`

To provide management services for each DB cluster, the `rdsadmin` user is created when the DB cluster is created. Attempting to drop, rename, change the password, or change privileges for the `rdsadmin` account will result in an error.

Restricting Password Management

You can restrict who can manage database user passwords to a special role. By doing this, you can have more control over password management on the client side.

You enable restricted password management with the static parameter `rds.restrict_password_commands` and use a role called `rds_password`. When the parameter `rds.restrict_password_commands` is set to 1, only users that are members of the `rds_password` role can run certain SQL commands. The restricted SQL commands are commands that modify database user passwords and password expiration time.

To use restricted password management, your DB cluster must be running Amazon Aurora for PostgreSQL 10.6 or higher. Because the `rds.restrict_password_commands` parameter is static, changing this parameter requires a database restart.

When a database has restricted password management enabled, if you try to run restricted SQL commands you get the following error: `ERROR: must be a member of rds_password to alter passwords`.

Following are some examples of SQL commands that are restricted when restricted password management is enabled.

```
postgres=> CREATE ROLE myrole WITH PASSWORD 'mypassword';
postgres=> CREATE ROLE myrole WITH PASSWORD 'mypassword' VALID UNTIL '2020-01-01';
postgres=> ALTER ROLE myrole WITH PASSWORD 'mypassword' VALID UNTIL '2020-01-01';
postgres=> ALTER ROLE myrole WITH PASSWORD 'mypassword';
postgres=> ALTER ROLE myrole VALID UNTIL '2020-01-01';
postgres=> ALTER ROLE myrole RENAME TO myrole2;
```

Some `ALTER ROLE` commands that include `RENAME TO` might also be restricted. They might be restricted because renaming a PostgreSQL role that has an MD5 password clears the password.

The `rds_superuser` role has membership for the `rds_password` role by default, and you can't change this. You can give other roles membership for the `rds_password` role by using the `GRANT` SQL command. We recommend that you give membership to `rds_password` to only a few roles that you use solely for password management. These roles require the `CREATEROLE` attribute to modify other roles.

Make sure that you verify password requirements such as expiration and needed complexity on the client side. We recommend that you restrict password-related changes by using your own client-side utility. This utility should have a role that is a member of `rds_password` and has the `CREATEROLE` role attribute.

Securing Aurora PostgreSQL Data with SSL

Amazon RDS supports Secure Socket Layer (SSL) encryption for Aurora PostgreSQL DB clusters. Using SSL, 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.

For general information about SSL support and PostgreSQL databases, see [SSL Support](#) in the PostgreSQL documentation. For information about using an SSL connection over JDBC, see [Configuring the Client](#) in the PostgreSQL documentation.

Topics

- [Requiring an SSL Connection to an Aurora PostgreSQL DB cluster \(p. 882\)](#)
- [Determining the SSL Connection Status \(p. 882\)](#)

SSL support is available in all AWS Regions for Aurora PostgreSQL. Amazon RDS creates an SSL certificate for your Aurora PostgreSQL DB cluster when the DB cluster is created. If you enable SSL certificate verification, then the SSL certificate includes the DB cluster endpoint as the Common Name (CN) for the SSL certificate to guard against spoofing attacks.

To connect to an Aurora PostgreSQL DB cluster over SSL

1. Download the certificate.

For information about downloading certificates, see [Using SSL/TLS to Encrypt a Connection to a DB Cluster \(p. 177\)](#).

2. Import the certificate into your operating system.
3. Connect to your Aurora PostgreSQL DB cluster over SSL.

When you connect using SSL, 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 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`.

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 Connection to an Aurora PostgreSQL DB cluster

You can require that connections to your Aurora PostgreSQL DB cluster use SSL 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 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 configuration.

You can set the `rds.force_ssl` parameter value by updating the DB cluster parameter group for your DB cluster. If the DB cluster parameter group isn't the default one, and the `ssl` parameter is already set to 1 when you set `rds.force_ssl` to 1, you don't need to reboot your DB cluster. Otherwise, you must reboot your DB cluster for the change to take effect. For more information on parameter groups, see [Working with DB Parameter Groups and DB Cluster Parameter Groups \(p. 286\)](#).

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 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 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 is being used. The function returns `t` if the connection is using SSL, 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 cipher:

```
postgres=> select ssl_cipher();
ssl_cipher
-----
DHE-RSA-AES256-SHA
(1 row)
```

If you enable `set rds.force_ssl` and restart your DB cluster, non-SSL connections are refused with the following message:

```
$ export PGSSLMODE=disable
$ psql postgres -h SOMEHOST.amazonaws.com -p 8192 -U someuser
psql: FATAL: no pg_hba.conf entry for host "host.ip", user "someuser", database "postgres",
      SSL off
$
```

For information about the `sslmode` option, see [Database Connection Control Functions](#) in the PostgreSQL documentation.

Updating Applications to Connect to Aurora PostgreSQL DB Clusters Using New SSL/TLS Certificates

As of September 19, 2019, Amazon RDS has published new Certificate Authority (CA) certificates for connecting to your Aurora DB clusters using Secure Socket Layer or Transport Layer Security (SSL/TLS). The previous CA certificates expire on March 5, 2020. Following, you can find information about updating your applications to use the new certificates. If your application connects to an Aurora DB cluster using SSL/TLS, you must take the following steps before **March 5, 2020**. Doing this means you can avoid interruption of connectivity between your applications and your Aurora DB clusters.

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 \(p. 179\)](#). For information about using SSL/TLS with PostgreSQL DB clusters, see [Securing Aurora PostgreSQL Data with SSL \(p. 881\)](#).

Topics

- [Determining Whether Applications Are Connecting to Aurora PostgreSQL DB Clusters Using SSL \(p. 884\)](#)
- [Determining Whether a Client Requires Certificate Verification in Order to Connect \(p. 884\)](#)
- [Updating Your Application Trust Store \(p. 885\)](#)
- [Using SSL/TLS Connections for Different Types of Applications \(p. 886\)](#)

Determining Whether Applications Are Connecting to Aurora PostgreSQL DB Clusters Using SSL

Check the DB cluster configuration for the value of the `rds.force_ssl` parameter. By default, the `rds.force_ssl` parameter is set to 0 (off). If the `rds.force_ssl` parameter is set to 1 (on), clients are required to use SSL/TLS for connections. For more information about parameter groups, see [Working with DB Parameter Groups and DB Cluster Parameter Groups \(p. 286\)](#).

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 masteruser -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=masteruser dbname=postgres"
```

For example, in the preceding case, if you use an invalid root certificate, you see an error similar to the following on your client.

```
psql: SSL error: certificate verify failed
```

Updating Your Application Trust Store

For information about updating the trust store for PostgreSQL applications, see [Secure TCP/IP Connections with SSL](#) in the PostgreSQL documentation.

Note

When you update the trust store, you can retain older certificates in addition to adding the new certificates.

Updating Your Application Trust Store for JDBC

You can update the trust store for applications that use JDBC for SSL/TLS connections.

To update the trust store for JDBC applications

1. Download the 2019 root certificate that works for all AWS Regions and put the file in your trust store directory.

For information about downloading the root certificate, see [Using SSL/TLS to Encrypt a Connection to a DB Cluster \(p. 177\)](#).

2. Convert the certificate to .der format using the following command.

```
openssl x509 -outform der -in rds-ca-2019-root.pem -out rds-ca-2019-root.der
```

Replace the file name with the one that you downloaded.

3. Import the certificate into the key store using the following command.

```
keytool -import -alias rds-root -keystore clientkeystore -file rds-ca-2019-root.der
```

4. Confirm that the key store was updated successfully.

```
keytool -list -v -keystore clientkeystore.jks
```

Enter the key store password when you are prompted for it.

Your output should contain the following.

```
rds-root,date, trustedCertEntry,  
Certificate fingerprint (SHA1):  
D4:0D:DB:29:E3:75:0D:FF:A6:71:C3:14:0B:BF:5F:47:8D:1C:80:96  
# This fingerprint should match the output from the below command  
openssl x509 -fingerprint -in rds-ca-2019-root.pem -noout
```

Using SSL/TLS Connections for Different Types of Applications

The following provides information about using SSL/TLS connections for different types of applications"

- **psql**

The client is invoked from the command line by specifying options either as a connection string or as environment variables. For SSL/TLS connections, the relevant options are `sslmode` (environment variable `PGSSLMODE`), `sslrootcert` (environment variable `PGSSLROOTCERT`).

For the complete list of options, see [Parameter Key Words](#) in the PostgreSQL documentation. For the complete list of environment variables, see [Environment Variables](#) in the PostgreSQL documentation.

- **pgAdmin**

This browser-based client is a more user-friendly interface for connecting to a PostgreSQL database.

For information about configuring connections, see the [pgAdmin documentation](#).

- **JDBC**

JDBC enables database connections with Java applications.

For general information about connecting to a PostgreSQL database with JDBC, see [Connecting to the Database](#) in the PostgreSQL documentation. For information about connecting with SSL/TLS, see [Configuring the Client](#) in the PostgreSQL documentation.

- **Python**

A popular Python library for connecting to PostgreSQL databases is `psycopg2`.

For information about using `psycopg2`, see the [psycopg2 documentation](#). For a short tutorial on how to connect to a PostgreSQL database, see [Psycopg2 Tutorial](#). You can find information about the options the `connect` command accepts in [The psycopg2 module content](#).

Important

After you have determined that your database connections use SSL/TLS and have updated your application trust store, you can update your database to use the rds-ca-2019 certificates. For instructions, see step 3 in [Updating Your CA Certificate by Modifying Your DB Instance \(p. 180\)](#).

Migrating Data to Amazon Aurora with PostgreSQL Compatibility

You have several options for migrating data from your existing database to an Amazon Aurora with PostgreSQL compatibility 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 from an RDS PostgreSQL DB instance

You can migrate data directly from an Amazon RDS PostgreSQL DB snapshot to an Aurora PostgreSQL DB cluster. For more information, see [Migrating an RDS PostgreSQL DB Snapshot to an Aurora PostgreSQL DB Cluster \(p. 887\)](#).

You can also migrate from an RDS PostgreSQL DB instance by creating an Aurora PostgreSQL Read Replica of a PostgreSQL DB instance. When the replica lag between the 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. For more information, see [Migrating Data from an RDS PostgreSQL DB Instance to an Aurora PostgreSQL DB Cluster by Using an Aurora Read Replica \(p. 890\)](#).

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?](#)

Importing Amazon S3 data

You can migrate by importing data from Amazon S3 into a table belonging to an Aurora PostgreSQL DB cluster for an RDS PostgreSQL DB instance. For more information, see [Importing Amazon S3 Data into an Aurora PostgreSQL DB Cluster \(p. 899\)](#).

For a list of AWS Regions where Aurora is available, see [Amazon Aurora](#) in the [AWS General Reference](#).

Migrating an RDS PostgreSQL DB Snapshot to an Aurora PostgreSQL DB Cluster

To create an Aurora PostgreSQL DB cluster, you can migrate a DB snapshot of an RDS PostgreSQL DB instance. The new Aurora PostgreSQL DB cluster is populated with the data from the original RDS PostgreSQL DB instance. The DB snapshot must be from an RDS DB instance running PostgreSQL 9.6.1 or 9.6.3. 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](#).

When you migrate the DB snapshot by using the 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 PostgreSQL DB cluster to be encrypted at rest by using an AWS Key Management Service (AWS KMS) encryption key. This option is available only for unencrypted DB snapshots.

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 snapshot that you want to migrate into an Aurora PostgreSQL DB cluster.
4. Choose **Migrate Database**.
5. Set the following values on the **Migrate Database** page:
 - **DB Instance Class:** Choose 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, up to a maximum size of 64 tebibytes (TiB). So you only need to choose a DB instance class that meets your current storage requirements. For more information, see [Overview of Aurora Storage \(p. 24\)](#).
 - **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.
- **VPC:** If you have an existing VPC, then you can use that VPC with your Aurora PostgreSQL DB cluster by choosing your VPC identifier, for example **vpc-a464d1c1**. For information on using an existing VPC, see [How to Create a VPC for Use with Amazon Aurora \(p. 239\)](#).

Otherwise, you can choose to have Amazon RDS create a VPC for you by choosing **Create a new VPC**.

- **Subnet Group:** If you have an existing subnet group, then you can use that subnet group with your Aurora PostgreSQL DB cluster by choosing your subnet group identifier, for example **gs-subnet-group1**.

Otherwise, you can choose to have Amazon RDS create a subnet group for you by choosing **Create a new subnet group**.

- **Publicly Accessible:** 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. 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:** 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 **Yes** for your new Aurora PostgreSQL DB cluster to be encrypted at rest. If you choose **Yes**, also choose an AWS KMS encryption key as the **Master 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 **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 PostgreSQL DB cluster, see [Connecting to an Amazon Aurora DB Cluster \(p. 166\)](#).

Migrating Data from an RDS PostgreSQL DB Instance to an Aurora PostgreSQL DB Cluster by Using an Aurora Read Replica

You can migrate from a PostgreSQL DB instance to an Aurora PostgreSQL DB cluster by using an Aurora Read Replica. When you need to migrate from an RDS PostgreSQL DB instance to an Aurora PostgreSQL DB cluster, we recommend using this approach.

In this case, Amazon RDS uses the PostgreSQL DB engine's streaming replication functionality to create a special type of DB cluster for the source PostgreSQL DB instance. This type of DB cluster is called an Aurora Read Replica. Updates made to the source PostgreSQL DB instance are asynchronously replicated to the Aurora Read Replica.

Topics

- [Overview of Migrating Data by Using an Aurora Read Replica \(p. 890\)](#)
- [Preparing to Migrate Data by Using an Aurora Read Replica \(p. 891\)](#)
- [Creating an Aurora Read Replica \(p. 891\)](#)
- [Promoting an Aurora Read Replica \(p. 897\)](#)

Overview of Migrating Data by Using an Aurora Read Replica

To migrate from an RDS PostgreSQL DB instance to an Aurora PostgreSQL DB cluster, we recommend creating an Aurora Read Replica of your source PostgreSQL DB instance. When the replica lag between the PostgreSQL DB instance and the Aurora PostgreSQL Read Replica is zero, you can stop replication. At this point, you can promote the Aurora Read Replica to be a standalone Aurora PostgreSQL DB cluster. This standalone DB cluster can then accept write loads.

Be prepared for migration to take a while, roughly several hours per tebibyte (TiB) of data. While the migration is in progress, your Amazon RDS PostgreSQL instance accumulates write ahead log (WAL) segments. Make sure that your Amazon RDS instance has sufficient storage capacity for these segments.

When you create an Aurora Read Replica of a PostgreSQL DB instance, Amazon RDS creates a DB snapshot of your source PostgreSQL DB instance. This snapshot is private to Amazon RDS and incurs no charges. Amazon RDS then migrates the data from the DB snapshot to the Aurora Read Replica. After the DB snapshot data is migrated to the new Aurora PostgreSQL DB cluster, RDS starts replication between your PostgreSQL DB instance and the Aurora PostgreSQL DB cluster.

You can only have one Aurora Read Replica for a PostgreSQL DB instance. If you try to create an Aurora Read Replica for your Amazon RDS PostgreSQL instance and you already have a Read Replica, the request is rejected.

Note

Replication issues can arise due to feature differences between Aurora PostgreSQL and the PostgreSQL engine version of your RDS PostgreSQL DB instance that is the replication master. You can replicate only from an Amazon RDS PostgreSQL instance that is compatible with the Aurora PostgreSQL version in question. For example, if the supported Aurora PostgreSQL version is 9.6.3, the Amazon RDS PostgreSQL DB instance must be running version 9.6.1 or greater. If you encounter an error, you can find help in the [Amazon RDS Community Forum](#) or by contacting AWS Support.

For more information on PostgreSQL Read Replicas, see [Working with Read Replicas](#) in the *Amazon RDS User Guide*.

Preparing to Migrate Data by Using an Aurora Read Replica

Before you migrate data from your RDS PostgreSQL instance to an Aurora PostgreSQL cluster, make sure that your instance has sufficient storage capacity. This storage capacity is for the write ahead log (WAL) segments that accumulate during the migration. There are several metrics to check for this, described following.

Metric	Description
FreeStorageSpace	The available storage space. Units: Bytes
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 a PostgreSQL DB instance by using the console or the AWS CLI.

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 PostgreSQL DB instance that you want to use as the source for your Aurora Read Replica, and choose **Create Aurora read replica** for **Actions**.

Databases			
	DB identifier	Role	Engine
<input type="radio"/>	<input checked="" type="checkbox"/> gs-db-cluster1	Regional	Aurora MySQL
<input type="radio"/>	js-cluster	Serverless	Aurora PostgreSQL
<input type="radio"/>	js-cluster-mysql	Serverless	Aurora MySQL
<input type="radio"/>	<input checked="" type="checkbox"/> js-cluster-mysql-1	Regional	Aurora MySQL
<input type="radio"/>	<input checked="" type="checkbox"/> js-db-cluster-1	Regional	Aurora PostgreSQL
<input checked="" type="radio"/>	jsdbinstance2	Instance	PostgreSQL

4. Choose the DB cluster specifications that you want to use for the Aurora Read Replica, as described in the following table.

Option	Description
DB instance class	Choose a DB instance class that defines the processing and memory requirements for the primary instance in the DB cluster. For more information about DB instance class options, see Choosing the DB Instance Class (p. 76) .
Multi-AZ deployment	Choose Create Replica in Different Zone to create the writer instance of the new DB cluster in another Availability Zone in the target AWS Region. For more information about multiple Availability Zones, see Choosing the Regions and Availability Zones (p. 80) .
DB instance identifier	Enter a name for the primary instance in your Aurora Read Replica DB cluster. This identifier is used in the endpoint address for the primary instance of the new DB cluster. The DB instance identifier has the following constraints: <ul style="list-style-type: none">• It must contain 1–63 alphanumeric characters or hyphens.• Its first character must be a letter.• It can't end with a hyphen or contain two consecutive hyphens.• It must be unique for all DB instances for each AWS account, for each AWS Region.

Option	Description
	The Aurora Read Replica DB cluster is created from a snapshot of the source DB instance. Thus, the master user name and master password for the Aurora Read Replica are the same as the master user name and master password for the source DB instance.
Virtual Private Cloud (VPC)	Choose the VPC to host the DB cluster. Choose Create new VPC to have Amazon RDS create a VPC for you. For more information, see DB Cluster Prerequisites (p. 100) .
Subnet group	Choose the DB subnet group to use for the DB cluster. Choose Create new DB Subnet Group to have Amazon RDS create a DB subnet group for you. For more information, see DB Cluster Prerequisites (p. 100) .
Public accessibility	Choose Yes to give the DB cluster a public IP address; otherwise, choose No . The instances in your DB cluster can be a mix of both public and private DB instances. For more information about hiding instances from public access, see Hiding a DB Instance in a VPC from the Internet (p. 251) .
Availability zone	Determine if you want to specify a particular Availability Zone. For more information about Availability Zones, see Choosing the Regions and Availability Zones (p. 80) .
VPC security groups	Choose one or more VPC security groups to secure network access to the DB cluster. Choose Create new VPC security group to have Amazon RDS create a VPC security group for you. For more information, see DB Cluster Prerequisites (p. 100) .
Database port	Specify the port for applications and utilities to use to access the database. Aurora PostgreSQL DB clusters default to the default PostgreSQL port, 5432. Firewalls at some companies block connections to this port. If your company firewall blocks the default port, choose another port for the new DB cluster.
DB parameter group	Choose a DB parameter group for the Aurora PostgreSQL DB cluster. Aurora has a default DB parameter group you can use, or you can create your own DB parameter group. For more information about DB parameter groups, see Working with DB Parameter Groups and DB Cluster Parameter Groups (p. 286) .
DB cluster parameter group	Choose a DB cluster parameter group for the Aurora PostgreSQL DB cluster. Aurora has a default DB cluster parameter group you can use, or you can create your own DB cluster parameter group. For more information about DB cluster parameter groups, see Working with DB Parameter Groups and DB Cluster Parameter Groups (p. 286) .

Option	Description
Encryption	Choose Enable encryption for your new Aurora DB cluster to be encrypted at rest. If you choose Enable encryption , also choose an AWS KMS encryption key as the Master key value.
Priority	Choose a failover priority for the DB cluster. If you don't choose a value, the default is tier-1 . This priority determines the order in which Aurora Replicas are promoted when recovering from a primary instance failure. For more information, see Fault Tolerance for an Aurora DB Cluster (p. 380) .
Backup retention period	Choose the length of time, 1–35 days, for Aurora to retain backup copies of the database. Backup copies can be used for point-in-time restores (PITR) of your database down to the second.
Enhanced monitoring	Choose Enable enhanced monitoring to enable gathering metrics in real time for the operating system that your DB cluster runs on. For more information, see Enhanced Monitoring (p. 469) .
Monitoring Role	Only available if you chose Enable enhanced monitoring . The AWS Identity and Access Management (IAM) role to use for Enhanced Monitoring. For more information, see Setting Up for and Enabling Enhanced Monitoring (p. 469) .
Granularity	Only available if you chose Enable enhanced monitoring . Set the interval, in seconds, between when metrics are collected for your DB cluster.
Auto minor version upgrade	Choose Yes 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.
Maintenance window	Choose the weekly time range during which system maintenance can occur.

5. Choose **Create read replica**.

AWS CLI

To create an Aurora Read Replica from a source PostgreSQL DB instance, use the [create-db-cluster](#) and [create-db-instance](#) AWS CLI commands to create a new Aurora PostgreSQL DB cluster. When you call the **create-db-cluster** command, include the **--replication-source-identifier** parameter to identify the Amazon Resource Name (ARN) for the source PostgreSQL DB instance. For more information about Amazon RDS ARNs, see [Amazon Relational Database Service \(Amazon RDS\)](#) in the *AWS General Reference*.

Don't specify the master user name, master password, or database name. The Aurora Read Replica uses the same master user name, master password, and database name as the source PostgreSQL DB instance.

For Linux, OS X, or Unix:

```
aws rds create-db-cluster --db-cluster-identifier sample-replica-cluster --engine aurora-postgresql \
    --db-subnet-group-name mysubnetgroup --vpc-security-group-ids sg-c7e5b0d2 \
    --replication-source-identifier arn:aws:rds:us-west-2:123456789012:db:master-postgresql-instance
```

For Windows:

```
aws rds create-db-cluster --db-cluster-identifier sample-replica-cluster --engine aurora-postgresql ^
    --db-subnet-group-name mysubnetgroup --vpc-security-group-ids sg-c7e5b0d2 ^
    --replication-source-identifier arn:aws:rds:us-west-2:123456789012:db:master-postgresql-instance
```

If you use the console to create an Aurora Read Replica, then RDS automatically creates the primary instance for your DB cluster Aurora Read Replica. If you use the CLI to create an Aurora Read Replica, you must explicitly create the primary instance for your DB cluster. The primary instance is the first instance that is created in a DB cluster.

You can create a primary instance for your DB cluster by using the [create-db-instance](#) CLI command with the following parameters:

- **--db-cluster-identifier**
The name of your DB cluster.
- **--db-instance-class**
The name of the DB instance class to use for your primary instance.
- **--db-instance-identifier**
The name of your primary instance.
- **--engine aurora-postgresql**
The database engine to use.

In the following example, you create a primary instance named **myreadreplicainstance** for the DB cluster named **myreadreplicacluster**. You do this using the DB instance class specified in **myinstanceclass**.

Example

For Linux, OS X, or Unix:

```
aws rds create-db-instance \
    --db-cluster-identifier myreadreplicacluster \
    --db-instance-class myinstanceclass
    --db-instance-identifier myreadreplicainstance \
    --engine aurora-postgresql
```

For Windows:

```
aws rds create-db-instance \
--db-cluster-identifier myreadreplicacluster \
--db-instance-class myinstanceclass \
--db-instance-identifier myreadreplicainstance \
--engine aurora-postgresql
```

RDS API

To create an Aurora Read Replica from a source PostgreSQL DB instance, use the RDS API operations [CreateDBCluster](#) and [CreateDBInstance](#) to create a new Aurora DB cluster and primary instance. Don't specify the master user name, master password, or database name. The Aurora Read Replica uses the same master user name, master password, and database name as the source PostgreSQL DB instance.

You can create a new Aurora DB cluster for an Aurora Read Replica from a source PostgreSQL DB instance. To do so, use the RDS API operation [CreateDBCluster](#) with the following parameters:

- **DBClusterIdentifier**
The name of the DB cluster to create.
- **DBSubnetGroupName**
The name of the DB subnet group to associate with this DB cluster.
- **Engine=aurora-postgresql**
The name of the engine to use.
- **ReplicationSourceIdentifier**
The Amazon Resource Name (ARN) for the source PostgreSQL DB instance. For more information about Amazon RDS ARNs, see [Amazon Relational Database Service \(Amazon RDS\)](#) in the *Amazon Web Services General Reference*.
- **VpcSecurityGroupIds**
The list of Amazon EC2 VPC security groups to associate with this DB cluster.

In the following example, you create a DB cluster named *myreadreplicacluster* from a source PostgreSQL DB instance. This cluster has an ARN set to *mysqlmasterARN*. The cluster is 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-postgresql
&ReplicationSourceIdentifier=mysqlmasterARN
&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 Amazon RDS automatically creates the primary instance for your DB cluster Aurora Read Replica. If you use the CLI to create an Aurora Read Replica, you must explicitly create the primary instance for your DB cluster. The primary instance is the first instance that is created in a DB cluster.

You can create a primary instance for your DB cluster by using the RDS API operation [CreateDBInstance](#) with the following parameters:

- **DBClusterIdentifier**
The name of your DB cluster.
- **DBInstanceClass**
The name of the DB instance class to use for your primary instance.
- **DBInstanceIdentifier**
The name of your primary instance.
- **Engine=aurora-postgresql**
The name of the engine to use.

In this example, you create a primary instance named *myreadreplicainstance* for the DB cluster named *myreadreplicacluster*. You do this using the DB instance class specified in *myinstanceclass*.

Example

```
https://rds.us-east-1.amazonaws.com/
?Action=CreateDBInstance
&DBClusterIdentifier=myreadreplicacluster
&DBInstanceClass=myinstanceclass
&DBInstanceIdentifier=myreadreplicainstance
&Engine=aurora-postgresql
&SignatureMethod=HmacSHA256
&SignatureVersion=4
&Version=2014-09-01
&X-Amz-Algorithm=AWS4-HMAC-SHA256
&X-Amz-Credential=AKIADQKE4SARGYLE/20140424/us-east-1/rds/aws4_request
&X-Amz-Date=20140424T194844Z
&X-Amz-SignedHeaders=content-type;host;user-agent;x-amz-content-sha256;x-amz-date
&X-Amz-Signature=bee4aabc750bf7dad0cd9e22b952bd6089d91e2a16592c2293e532eeaab8bc77
```

Promoting an Aurora Read Replica

After migration completes, you can promote the Aurora Read Replica to a standalone DB cluster. You then direct your client applications to the endpoint for the Aurora Read Replica. For more information on the Aurora endpoints, see [Amazon Aurora Connection Management \(p. 4\)](#). Promotion should complete fairly quickly. You can't delete the master PostgreSQL DB instance or unlink the DB instance and the Aurora Read Replica until the promotion is complete.

Before you promote your Aurora Read Replica, stop any transactions from being written to the source PostgreSQL DB instance. Then wait for the replica lag on the Aurora Read Replica to reach zero.

After you promote your Read Replica, confirm that the promotion has completed. To do this, choose **Instances** in the navigation pane and confirm that there is a **Promoted Read Replica cluster to stand-alone database cluster** event for your Read Replica. After promotion is complete, the master PostgreSQL DB Instance and the Aurora Read Replica are unlinked. At this point, you can safely delete the DB instance if you want to.

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 **Instances**.
3. Choose the DB instance for the Aurora Read Replica and choose **Promote Read Replica** for **Actions**.
4. Choose **Promote Read Replica**.

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, OS X, 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
```

Importing Amazon S3 Data into an Aurora PostgreSQL DB Cluster

You can import data from Amazon S3 into a table belonging to an Aurora PostgreSQL DB cluster. To do this, you use the `aws_s3` PostgreSQL extension that Aurora PostgreSQL provides.

Note

To import from Amazon S3 into Aurora PostgreSQL, your database must be running PostgreSQL version 10.7 or later.

For more information on storing data with Amazon S3, see [Create a Bucket](#) in the *Amazon Simple Storage Service Getting Started Guide*. For instructions on how to upload a file to an Amazon S3 bucket, see [Add an Object to a Bucket](#) in the *Amazon Simple Storage Service Getting Started Guide*.

Topics

- [Overview of Importing Amazon S3 Data \(p. 899\)](#)
- [Setting Up Access to an Amazon S3 Bucket \(p. 900\)](#)
- [Using the `aws_s3.table_import_from_s3` Function to Import Amazon S3 Data \(p. 904\)](#)
- [Function Reference \(p. 906\)](#)

Overview of Importing Amazon S3 Data

To import data stored in an Amazon S3 bucket to a PostgreSQL database table, follow these steps.

To import S3 data into Aurora PostgreSQL

1. Install the required PostgreSQL extensions. These include the `aws_s3` and `aws_commons` extensions. To do so, start `psql` and use the following command.

```
psql=> CREATE EXTENSION aws_s3 CASCADE;
NOTICE: installing required extension "aws_commons"
```

The `aws_s3` extension provides the [aws_s3.table_import_from_s3 \(p. 907\)](#) function that you use to import Amazon S3 data. The `aws_commons` extension provides additional helper functions.

2. Identify the database table and Amazon S3 file to use.

The [aws_s3.table_import_from_s3 \(p. 907\)](#) function requires the name of the PostgreSQL database table that you want to import data into. The function also requires that you identify the Amazon S3 file to import. To provide this information, take the following steps.

- a. Identify the PostgreSQL database table to put the data in. For example, the following is a sample `t1` database table used in the examples for this topic.

```
psql=> CREATE TABLE t1 (col1 varchar(80), col2 varchar(80), col3 varchar(80));
```

- b. Get the following information to identify the Amazon S3 file that you want to import:

- Bucket name – A *bucket* is a container for Amazon S3 objects or files.
- File path – The file path locates the file in the Amazon S3 bucket.
- AWS Region – The AWS Region is the location of the Amazon S3 bucket. For example, if the S3 bucket is in the US East (N. Virginia) Region, use `us-east-1`. For a listing of AWS Region names and associated values, see [Choosing the Regions and Availability Zones \(p. 80\)](#).

To find how to get this information, see [View an Object in the Amazon Simple Storage Service Getting Started Guide](#). You can confirm the information by using the AWS CLI command `aws s3 cp`. If the information is correct, this command downloads a copy of the Amazon S3 file.

```
aws s3 cp s3://sample_s3_bucket/sample_file_path ./
```

- c. Use the [aws_commons.create_s3_uri \(p. 909\)](#) function to create an `aws_commons._s3_uri_1` structure to hold the Amazon S3 file information. You provide this `aws_commons._s3_uri_1` structure as a parameter in the call to the [aws_s3.table_import_from_s3 \(p. 907\)](#) function.

For a psql example, see the following.

```
psql=> SELECT aws_commons.create_s3_uri(  
      'sample_s3_bucket',  
      'sample.csv',  
      'us-east-1'  
) AS s3_uri \gset
```

3. Provide permission to access the Amazon S3 file.

To import data from an Amazon S3 file, you need to give the Aurora PostgreSQL DB cluster permission to access the Amazon S3 bucket the file is in. To do this, you use either an AWS Identity and Access Management (IAM) role or security credentials. For more information, see [Setting Up Access to an Amazon S3 Bucket \(p. 900\)](#).

4. Import the Amazon S3 data by calling the `aws_s3.table_import_from_s3` function.

After you complete the previous preparation tasks, use the [aws_s3.table_import_from_s3 \(p. 907\)](#) function to import the Amazon S3 data. For more information, see [Using the aws_s3.table_import_from_s3 Function to Import Amazon S3 Data \(p. 904\)](#).

Setting Up Access to an Amazon S3 Bucket

To import data from an Amazon S3 file, you need to give the Aurora PostgreSQL DB cluster permission to access the Amazon S3 bucket the file is in. 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 \(p. 900\)](#)
- [Using Security Credentials to Access an Amazon S3 Bucket \(p. 904\)](#)
- [Troubleshooting Access to Amazon S3 \(p. 904\)](#)

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 \(p. 907\)](#) function call.

To do this, create an IAM policy that provides access to the Amazon S3 bucket. Create an IAM role and attach the policy to the role. Then assign the IAM role to your DB cluster.

To give an Aurora PostgreSQL DB cluster access to Amazon S3 through an IAM role

1. Create an IAM policy. This policy provides the bucket and object permissions that allow your Aurora PostgreSQL DB cluster to access Amazon S3.

Include in the policy the following required actions to allow the transfer of files from an Amazon S3 bucket to Aurora PostgreSQL:

- s3:GetObject
- s3>ListBucket

Include in the policy the following resources to identify the Amazon S3 bucket and objects in the bucket. This shows the Amazon Resource Name (ARN) format for accessing Amazon S3.

- arn:aws:s3:::*your-s3-bucket*
- arn:aws:s3:::*your-s3-bucket*/*

For more information on creating an IAM policy for Aurora PostgreSQL, see [Creating and Using an IAM Policy for IAM Database Access \(p. 209\)](#). 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

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.

Example

For Linux, OS X, or Unix:

```
aws iam create-policy \
--policy-name rds-s3-import-policy \
--policy-document '{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "s3import",
            "Action": [
                "s3:GetObject",
                "s3>ListBucket"
            ],
            "Effect": "Allow",
            "Resource": [
                "arn:aws:s3:::your-s3-bucket",
                "arn:aws:s3:::your-s3-bucket/*"
            ]
        }
    ]
}'
```

For Windows:

```
aws iam create-policy ^
--policy-name rds-s3-import-policy ^
--policy-document '{
    "Version": "2012-10-17",
    "Statement": [
```

```
{
    "Sid": "s3import",
    "Action": [
        "s3:GetObject",
        "s3>ListBucket"
    ],
    "Effect": "Allow",
    "Resource": [
        "arn:aws:s3:::your-s3-bucket",
        "arn:aws:s3:::your-s3-bucket/*"
    ]
}
}'
```

2. Create an IAM role. You do this so Aurora PostgreSQL can assume this IAM role 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-import-role`.

Example

For Linux, OS X, 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"
        }
    ]
}'
```

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"
        }
    ]
}'
```

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-import-role`. Replace `your-policy-arn` with the policy ARN that you noted in an earlier step.

Example

For Linux, OS X, 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 instance**.
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, OS X, 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
```

Using Security Credentials to Access an Amazon S3 Bucket

If you prefer, you can use security credentials to provide access to an Amazon S3 bucket instead of providing access with an IAM role. To do this, use the `credentials` parameter in the [aws_s3.table_import_from_s3 \(p. 907\)](#) 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 \(p. 909\)](#) function to set the access key and secret key in an `aws_commons._aws_credentials_1` structure, as shown following.

```
psql=> SELECT aws_commons.create_aws_credentials(
    'sample_access_key', 'sample_secret_key', '')
AS creds \gset
```

After creating the `aws_commons._aws_credentials_1` structure, use the [aws_s3.table_import_from_s3 \(p. 907\)](#) function with the `credentials` parameter to import the data, as shown following.

```
psql=> SELECT aws_s3.table_import_from_s3(
    't', '', '(format csv)',
    :'s3_uri',
    :'creds'
);
```

Or you can include the [aws_commons.create_aws_credentials \(p. 909\)](#) function call inline within the `aws_s3.table_import_from_s3` function call.

```
psql=> SELECT aws_s3.table_import_from_s3(
    't', '', '(format csv)',
    :'s3_uri',
    aws_commons.create_aws_credentials('sample_access_key', 'sample_secret_key', '')
);
```

Troubleshooting Access to Amazon S3

If you encounter connection problems when attempting to import Amazon S3 file data, see the following for recommendations:

- [Troubleshooting Amazon Aurora Identity and Access \(p. 223\)](#)
- [Troubleshooting Amazon S3](#)
- [Troubleshooting Amazon S3 and IAM](#)

Using the aws_s3.table_import_from_s3 Function to Import Amazon S3 Data

Import your Amazon S3 data by calling the [aws_s3.table_import_from_s3 \(p. 907\)](#) function.

Note

The following examples use the IAM role method for providing access to the Amazon S3 bucket. Thus, there are no credential parameters in the `aws_s3.table_import_from_s3` function calls.

The following shows a typical PostgreSQL example using `psql`.

```
psql=> SELECT aws_s3.table_import_from_s3(
    't1',
```

```
  '',
  '(format csv)',
  :'s3_uri'
);
```

The parameters are the following:

- `t1` – The name for the table in the PostgreSQL DB cluster to copy the data into.
- `''` – An optional list of columns in the database table. You can use this parameter to indicate which columns of the S3 data go in which table columns. If no columns are specified, all the columns are copied to the table. For an example of using a column list, see [Importing an Amazon S3 File That Uses a Custom Delimiter \(p. 905\)](#).
- `(format csv)` – PostgreSQL COPY arguments. The copy process uses the arguments and format of the [PostgreSQL COPY](#) command. In the preceding example, the COPY command uses the comma-separated value (CSV) file format to copy the data.
- `s3_uri` – A structure that contains the information identifying the Amazon S3 file. For an example of using the [aws_commons.create_s3_uri \(p. 909\)](#) function to create an `s3_uri` structure, see [Overview of Importing Amazon S3 Data \(p. 899\)](#).

For the full reference of this function, see [aws_s3.table_import_from_s3 \(p. 907\)](#).

The following examples show how to specify different kinds of files when importing Amazon S3 data.

Topics

- [Importing an Amazon S3 File That Uses a Custom Delimiter \(p. 905\)](#)
- [Importing an Amazon S3 Compressed \(gzip\) File \(p. 906\)](#)
- [Importing an Encoded Amazon S3 File \(p. 906\)](#)

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 \(p. 907\)](#) function.

For this example, assume that the following information is organized into pipe-delimited columns in the Amazon S3 file.

```
1|foo1|bar1|elephant1
2|foo2|bar2|elephant2
3|foo3|bar3|elephant3
4|foo4|bar4|elephant4
...
```

To import a file that uses a custom delimiter

1. Create a table in the database for the imported data.

```
psql=> CREATE TABLE test (a text, b text, c text, d text, e text);
CREATE TABLE
```

2. Use the following form of the [aws_s3.table_import_from_s3 \(p. 907\)](#) function to import data from the Amazon S3 file.

You can include the [aws_commons.create_s3_uri \(p. 909\)](#) function call inline within the `aws_s3.table_import_from_s3` function call to specify the file.

```
psql=> SELECT aws_s3.table_import_from_s3(
    'test',
    'a,b,d,e',
    'DELIMITER ''|''',
    aws_commons.create_s3_uri('sampleBucket', 'pipeDelimitedSampleFile', 'us-east-2')
);
```

The data is now in the table in the following columns.

```
psql=> SELECT * FROM test;
a | b | c | d | e
---+---+---+---+---+
1 | foo1 | | bar1 | elephant1
2 | foo2 | | bar2 | elephant2
3 | foo3 | | bar3 | elephant3
4 | foo4 | | bar4 | elephant4
```

Importing an Amazon S3 Compressed (gzip) File

The following example shows how to import a file from Amazon S3 that is compressed with gzip.

Ensure that the file contains the following Amazon S3 metadata:

- Key: Content-Encoding
- Value: gzip

For more about adding these values to Amazon S3 metadata, see [How Do I Add Metadata to an S3 Object?](#) in the *Amazon Simple Storage Service Console User Guide*.

Import the gzip file into your Aurora PostgreSQL DB cluster as shown following.

```
psql=> CREATE TABLE test_gzip(id int, a text, b text, c text, d text);
CREATE TABLE
psql=> SELECT aws_s3.table_import_from_s3(
    'test_gzip', '', '(format csv)',
    'myS3Bucket', 'test-data.gz', 'us-east-2'
);
```

Importing an Encoded Amazon S3 File

The following example shows how to import a file from Amazon S3 that has Windows-1252 encoding.

```
psql=> SELECT aws_s3.table_import_from_s3(
    'test_table', '', 'encoding ''WIN1252'''',
    aws_commons.create_s3_uri('sampleBucket', 'SampleFile', 'us-east-2')
);
```

Function Reference

Functions

- [aws_s3.table_import_from_s3 \(p. 907\)](#)
- [aws_commons.create_s3_uri \(p. 909\)](#)
- [aws_commons.create_aws_credentials \(p. 909\)](#)

[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 three 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 these 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
)
```

The `aws_s3.table_import_from_s3` parameters are described in the following table.

Parameter	Description
<code>table_name</code>	A required text string containing the name of the PostgreSQL database table to import the data into.
<code>column_list</code>	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. 905) .
<code>options</code>	A required text string containing arguments for the PostgreSQL <code>COPY</code> command. These arguments specify how the data is to be copied into the PostgreSQL table. For more details, see the PostgreSQL COPY documentation .
<code>s3_info</code>	An <code>aws_commons._s3_uri_1</code> composite type containing the following information about the S3 object: <ul style="list-style-type: none"> • <code>bucket</code> – The Amazon S3 bucket name containing the file. • <code>file_path</code> – The Amazon S3 path of the file. • <code>region</code> – The AWS Region that the file is in. For a listing of AWS Region names and associated region values, see Choosing the Regions and Availability Zones (p. 80). To create an <code>aws_commons._s3_uri_1</code> composite structure, see aws_commons.create_s3_uri (p. 909) .

Parameter	Description
credentials	An <code>aws_commons._aws_credentials_1</code> composite type containing the following credentials to use for the import operation: <ul style="list-style-type: none"> • Access key • Secret key • Session token To create an <code>aws_commons._aws_credentials_1</code> composite structure, see aws_commons.create_aws_credentials (p. 909) .

Alternate Parameters

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
)
```

Find descriptions for these alternate parameters in the following table.

Parameter	Description
<code>bucket</code>	A text string containing the name of the Amazon S3 bucket that contains the file.
<code>file_path</code>	A text string containing the Amazon S3 path of the file.
<code>region</code>	A text string containing the AWS Region that the file is in. For a listing of AWS Region names and associated values, see Choosing the Regions and Availability Zones (p. 80) .

Parameter	Description
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. You use the results of the `aws_commons.create_s3_uri` function in the `s3_info` parameter of the [aws_s3.table_import_from_s3 \(p. 907\)](#) function. The function syntax is as follows.

```
aws_commons.create_s3_uri(
    bucket text,
    file_path text,
    region text
)
```

The `aws_commons.create_s3_uri` function parameters are described in the following table.

Parameter	Description
bucket	A required text string containing the Amazon S3 bucket name for the file.
file_path	A required text string containing the Amazon S3 path of the file.
region	A required text string containing the AWS Region the file is in. For a listing of AWS Region names and associated region values, see Choosing the Regions and Availability Zones (p. 80) .

[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 \(p. 907\)](#) function. The function syntax is as follows.

```
aws_commons.create_aws_credentials(
    access_key text,
    secret_key text,
    session_token text
)
```

The `aws_commons.create_aws_credentials` function parameters are described in the following table.

Parameter	Description
access_key	A required text string containing the access key to use for importing an Amazon S3 file. The default is NULL.

Parameter	Description
<code>secret_key</code>	A required text string containing the secret key to use for importing an Amazon S3 file. The default is NULL.
<code>session_token</code>	An optional text string containing the session token to use for importing an Amazon S3 file. The default is NULL. Note, if you provide an optional <code>session_token</code> , you can use temporary credentials.

Managing Amazon Aurora PostgreSQL

The following sections discuss managing performance and scaling for an Amazon Aurora PostgreSQL DB cluster.

Topics

- [Scaling Aurora PostgreSQL DB Instances \(p. 910\)](#)
- [Maximum Connections to an Aurora PostgreSQL DB Instance \(p. 910\)](#)

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 \(p. 284\)](#).

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. For detailed specifications of the DB instance classes supported by Aurora PostgreSQL, see [Hardware Specifications for All Available DB Instance Classes for Aurora \(p. 77\)](#).

Maximum Connections to an Aurora PostgreSQL DB Instance

The maximum number of connections allowed to an Aurora PostgreSQL DB instance is determined by the `max_connections` parameter in the instance-level parameter group for the DB instance. By default, this value is set to the following equation:

`LEAST({DBInstanceClassMemory} / 9531392 , 5000)`.

Setting the `max_connections` parameter to this equation makes sure that the number of allowed connection scales well with the size of the instance. For example, suppose your DB instance class is `db.r4.large`, which has 15.25 gibabytes (GiB) of memory. Then the maximum connections allowed is 1660, as shown in the following equation:

```
LEAST( (15.25 * 1000000000) / 9531392 , 5000 ) = 1600
```

The following table lists the resulting default value of `max_connections` for each DB instance class available to Aurora PostgreSQL. You can increase the maximum number of connections to your Aurora PostgreSQL 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, up to 262,143.

Instance Class	max_connections Default Value		
<code>db.r4.large</code>	1600		

Instance Class	max_connections Default Value		
db.r4.xlarge	3200		
db.r4.2xlarge	5000		
db.r4.4xlarge	5000		
db.r4.8xlarge	5000		
db.r4.16xlarge	5000		
db.r5.large	1600		
db.r5.xlarge	3300		
db.r5.2xlarge	5000		
db.r5.4xlarge	5000		
db.r5.12xlarge	5000		
db.r5.24xlarge	5000		

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 \(p. 911\)](#)
- [Monitoring Aurora PostgreSQL Replication \(p. 912\)](#)
- [Using PostgreSQL Logical Replication with Aurora \(p. 912\)](#)

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. 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 \(p. 47\)](#).

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 additional work is required to replicate a copy of the data for each Aurora Replica. In contrast, PostgreSQL Read Replicas must apply, on a single thread, all write operations from the master DB instance to their local data store. This limitation can affect the ability of PostgreSQL Read Replicas to support large volumes of write traffic.

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 an Amazon Aurora DB Cluster \(p. 433\)](#).

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 \(p. 912\)](#)
- [Example of Logical Replication of a Database Table \(p. 913\)](#)
- [Logical Replication Using the AWS Database Migration Service \(p. 914\)](#)

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 databases.

Logical replication uses a publish and subscribe model. *Publishers* and *subscribers* are the nodes. A *publication* is defined on a publisher and is a set of changes generated from one or more database tables. A *subscription* defines the connection to another database and one or more publications to which it subscribes. A *subscription* is defined on a subscriber. The publication and subscription make the connection between the publisher and subscriber databases.

Note

To perform logical replication for a PostgreSQL database, your AWS user account needs the `rds_superuser` role.

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 \(p. 290\)](#). Use the following settings:
 - For **Parameter group family**, choose **aurora-postgres10** or later.
 - For **Type**, choose **DB Cluster Parameter Group**.
2. Modify the cluster parameter group, as described in [Modifying Parameters in a DB Cluster Parameter Group \(p. 295\)](#). Set the `rds.logical_replication` static parameter to 1.

Enabling the `rds.logical_replication` parameter affects the DB cluster's performance.

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 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 \(p. 264\)](#).
 2. Restart the DB cluster for static parameter changes to take effect. The 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 \(p. 101\)](#).
 1. Choose the **Amazon Aurora** engine and choose the **PostgreSQL-compatible** edition.
 2. For **DB 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 an RDS for PostgreSQL database as the subscriber. 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 \(p. 912\)](#).
2. Set up the publisher database. 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 a database to be your subscriber.

For this example, create an Amazon RDS for PostgreSQL database. Be sure to use the PostgreSQL DB engine version 10.4 or later, which supports logical replication. For details on creating an RDS PostgreSQL DB instance, see [Creating a DB Instance Running the PostgreSQL Database Engine](#) 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 for information 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. In this example, you migrate data from a database table to an RDS PostgreSQL database as the subscriber. This example uses the same publisher and subscriber that were created in [Example of Logical Replication of a Database Table \(p. 913\)](#).

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>. 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 the same subnet group as for the writer DB instance.
 - 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**.
 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 the task is created, AWS DMS begins migrating data from the publisher to the subscriber.

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 [Using Amazon RDS Performance Insights \(p. 476\)](#).

- Automatically add or remove Aurora Replicas with Aurora Auto Scaling. For more information, see [Using Amazon Aurora Auto Scaling with Aurora Replicas \(p. 351\)](#).
- 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 \(p. 943\)](#).

Managing Query Execution Plans for Aurora PostgreSQL

With query plan management for Amazon Aurora with PostgreSQL compatibility, 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 worse. A number of changes can all cause the query optimizer to choose a different plan and lead to performance regression. These changes include changes in statistics, constraints, environment settings, query parameter bindings, and software upgrades. Regression is a major concern for high-performance applications.

With query plan management, you can control execution plans for a set of statements that you want to manage. You can do the following:

- Improve plan stability by forcing the optimizer to choose from a small number of known, good plans.
- Optimize plans centrally and then distribute the best plans globally.
- Identify indexes that aren't used and assess the impact of creating or dropping an index.
- Automatically detect a new minimum-cost plan discovered by the optimizer.
- Try new optimizer features with less risk, because you can choose to approve only the plan changes that improve performance.

Topics

- [Enabling Query Plan Management for Aurora PostgreSQL \(p. 917\)](#)
- [Upgrading Query Plan Management \(p. 918\)](#)
- [Basics of Query Plan Management \(p. 918\)](#)
- [Best Practices for Query Plan Management \(p. 921\)](#)
- [Examining Plans in the `apg_plan_mgmt.dba_plans` view \(p. 922\)](#)
- [Capturing Execution Plans \(p. 925\)](#)
- [Using Managed Plans \(p. 927\)](#)
- [Maintaining Execution Plans \(p. 930\)](#)
- [Parameter Reference for Query Plan Management \(p. 934\)](#)
- [Function Reference for Query Plan Management \(p. 938\)](#)

Enabling Query Plan Management for Aurora PostgreSQL

Query plan management is available with Amazon Aurora PostgreSQL version 2.1.0 and greater.

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. For more information, see [Creating a DB Parameter Group \(p. 289\)](#).
3. Create a new cluster-level parameter group to use for query plan management. For more information, see [Creating a DB Cluster Parameter Group \(p. 290\)](#).
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 \(p. 295\)](#).
5. Restart 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.

Note

To create the `apg_plan_mgmt` extension, you need the `rds_superuser` role. When you create the `apg_plan_mgmt` extension, the `apg_plan_mgmt` role is created. Users must be granted the `apg_plan_mgmt` role to administer the `apg_plan_mgmt` extension.

Upgrading Query Plan Management

If you installed query plan management version 1.0, we strongly recommend that you upgrade to version 1.0.1.

To upgrade, run the following commands at the cluster or DB instance level.

```
ALTER EXTENSION apg_plan_mgmt UPDATE TO '1.0.1';
SELECT apg_plan_mgmt.validate_plans('update_plan_hash');
SELECT apg_plan_mgmt.reload();
```

Basics of Query Plan Management

You can manage any SELECT, INSERT, UPDATE, or DELETE statement with query plan management, regardless of how complex the statement is. Prepared, dynamic, embedded, and immediate-mode SQL statements are all supported. All PostgreSQL language features can be used, including partitioned tables, inheritance, row-level security, and recursive common table expressions (CTEs).

Note

Currently, you can't capture plans for statements inside a PL/pgSQL function.

Topics

- [Performing a Manual Plan Capture \(p. 918\)](#)
- [Viewing Captured Plans \(p. 918\)](#)
- [Working with Managed Statements and the SQL Hash \(p. 919\)](#)
- [Working with Automatic Plan Capture \(p. 920\)](#)
- [Validating Plans \(p. 920\)](#)
- [Approving New Plans That Improve Performance \(p. 920\)](#)
- [Deleting Plans \(p. 921\)](#)

Performing a Manual Plan Capture

To capture plans for specific statements, use the manual capture mode as in the following example.

```
/* Turn on manual capture */
SET apg_plan_mgmt.capture_plan_baselines = manual;
EXPLAIN SELECT COUNT(*) from pg_class;           -- capture the plan baseline
SET apg_plan_mgmt.capture_plan_baselines = off;    -- turn off capture
SET apg_plan_mgmt.use_plan_baselines = true;       -- turn on plan usage
```

You can either execute SELECT, INSERT, UPDATE, or DELETE statements, or you can include the EXPLAIN statement as shown above. Use EXPLAIN to capture a plan without the overhead or potential side-effects of executing the statement. For more about manual capture, see [Manually Capturing Plans for Specific SQL Statements \(p. 925\)](#).

Viewing Captured Plans

When EXPLAIN SELECT runs in the previous example, the optimizer saves the plan. To do so, it inserts a row into the `apg_plan_mgmt.dba_plans` view and commits the plan in an autonomous transaction. You can see the contents of the `apg_plan_mgmt.dba_plans` view if you've been granted the `apg_plan_mgmt` role. The following query displays some important columns of the `dba_plans` view.

```
SELECT sql_hash, plan_hash, status, enabled, plan_outline, sql_text::varchar(40)
FROM apg_plan_mgmt.dba_plans
ORDER BY sql_text, plan_created;
```

Each row displayed represents a managed plan. The preceding example displays the following information.

- `sql_hash` – The ID of the managed statement that the plan is for.
- `plan_hash` – The ID of the managed plan.
- `status` – The status of the plan. The optimizer can run an approved plan.
- `enabled` – A value that indicates whether the plan is enabled for use or disabled and not for use.
- `plan_outline` – Details of the managed plan.

For more about the `apg_plan_mgmt.dba_plans` view, see [Examining Plans in the apg_plan_mgmt.dba_plans view \(p. 922\)](#).

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 whitespace 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, by default, the optimizer captures plans for statements that run at least two times. Do the following to use automatic plan capture.

1. Turn on automatic plan capture by setting `apg_plan_mgmt.capture_plan_baselines` to `automatic` in the parameter group for the DB instance. For more information, see [Modifying Parameters in a DB Parameter Group \(p. 291\)](#).
2. Restart your DB instance.

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. Then restart the database for the setting to take effect.

For more about plan capture, see [Capturing Execution Plans \(p. 925\)](#).

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 \(p. 931\)](#).

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 \(p. 930\)](#).

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 \(p. 933\)](#).

Best Practices for Query Plan Management

Consider using a plan management style that is either proactive or reactive. These plan management styles contrast in how and when new plans get approved for use.

Proactive Plan Management to Help Prevent Performance Regression

With proactive plan management, you manually approve new plans after you have verified that they are faster. Do this to prevent plan performance regressions. Follow these steps for proactive plan management:

1. In a development environment, identify the SQL statements that have the greatest impact on performance or system throughput. Then capture the plans for these statements as described in [Manually Capturing Plans for Specific SQL Statements \(p. 925\)](#) and [Automatically Capturing Plans \(p. 925\)](#).
2. Export the captured plans from the development environment and import them into the production environment. For more information, see [Exporting and Importing Plans \(p. 933\)](#).
3. In production, run your application and enforce the use of approved managed plans. For more information, see [Using Managed Plans \(p. 927\)](#). While the application runs, also add new plans as the optimizer discovers them. For more information, see [Automatically Capturing Plans \(p. 925\)](#).
4. Analyze the unapproved plans and approve those that perform well. For more information, see [Evaluating Plan Performance \(p. 930\)](#).
5. While your application continues to run, the optimizer begins to use the new plans as appropriate.

Reactive Plan Management to Detect and Repair Performance Regression

With reactive plan management, you monitor your application as it runs to detect plans that cause performance regressions. When you detect regressions, you manually reject or fix the bad plans. Follow these steps for reactive plan management:

1. While your application runs, enforce the use of managed plans and automatically add newly discovered plans as unapproved. For more information, see [Using Managed Plans \(p. 927\)](#) and [Automatically Capturing Plans \(p. 925\)](#).
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 \(p. 931\)](#).

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 \(p. 931\)](#).

Examining Plans in the apg_plan_mgmt.dba_plans view

Query plan management provides a new SQL view for database administrators (DBAs) to use called `apg_plan_mgmt.dba_plans`. Each database in a DB instance has its own `apg_plan_mgmt.dba_plans` view.

This view contains the plan history for all of your managed statements. Each managed plan is identified by the combination of a SQL hash value and a plan hash value. With these identifiers, you can use tools such as Amazon RDS Performance Insights to track individual plan performance. For more information on Performance Insights, see [Using Amazon RDS Performance Insights](#).

Note

Access to the `apg_plan_mgmt.dba_plans` view is restricted to users that hold the `apg_plan_mgmt` role.

Listing Managed Plans

To list the managed plans, use a `SELECT` statement on the `apg_plan_mgmt.dba_plans` view. The following example displays some columns in the `dba_plans` view such as the `status`, which identifies the approved and unapproved plans.

```
SELECT sql_hash, plan_hash, status, enabled, stmt_name
FROM apg_plan_mgmt.dba_plans;

sql_hash | plan_hash | status | enabled | stmt_name
-----+-----+-----+-----+-----
1984047223 | 512153379 | approved | t | rangequery
1984047223 | 512284451 | unapproved | t | rangequery
(2 rows)
```

Reference for the apg_plan_mgmt.dba_plans View

The columns of plan information in the `apg_plan_mgmt.dba_plans` view include the following.

dba_plans Column	Description
<code>cardinality_error</code>	A measure of the error between the estimated cardinality versus the actual cardinality. <i>Cardinality</i> is the number of table rows that the plan is to process. If the cardinality error is large, then it increases the

dba_plans Column	Description
	likelihood that the plan isn't optimal. This column is populated by the apg_plan_mgmt.evolve_plan_baselines (p. 939) function.
compatibility_level	The feature level of the Aurora PostgreSQL optimizer.
created_by	The authenticated user (<code>session_user</code>) who created the plan.
enabled	An indicator of whether the plan is enabled or disabled. All plans are enabled by default. You can disable plans to prevent them from being used by the optimizer. To modify this value, use the apg_plan_mgmt.set_plan_enabled (p. 941) 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 apg_plan_mgmt.evolve_plan_baselines (p. 939) function.
execution_time_ms	The estimated time in milliseconds that the plan would run. This column is populated by the apg_plan_mgmt.evolve_plan_baselines (p. 939) 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 apg_plan_mgmt.plan_retention_period (p. 937) parameter.
last_validated	The most recent date and time when it was verified that the plan could be recreated by either the apg_plan_mgmt.validate_plans (p. 942) function or the apg_plan_mgmt.evolve_plan_baselines (p. 939) 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 apg_plan_mgmt.evolve_plan_baselines (p. 939) function.
origin	How the plan was captured with the apg_plan_mgmt.capture_plan_baselines (p. 935) 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.

dba_plans Column	Description
plan_hash	The plan identifier. The combination of <code>plan_hash</code> and <code>sql_hash</code> uniquely identifies a specific plan.
plan_outline	A representation of the plan that is used to recreate the actual execution plan, and that is database-independent. Operators in the tree correspond to operators that appear in the EXPLAIN output.
planning_time_ms	The actual time to run the planner, in milliseconds. This column is populated by the apg_plan_mgmt.evolve_plan_baselines (p. 939) function.
queryId	A statement hash, as calculated by the <code>pg_stat_statements</code> extension. This isn't a stable or database-independent identifier because it depends on object identifiers (OIDs).
sql_hash	A hash value of the SQL statement text, normalized with literals removed.
sql_text	The full text of the SQL statement.
status	<p>A plan's status, which determines how the optimizer uses a plan. Valid values include the following. Case does not matter.</p> <ul style="list-style-type: none"> • approved – A usable plan that the optimizer can choose to run. The optimizer runs the least-cost plan from a managed statement's set of approved plans (baseline). To reset a plan to approved, use the apg_plan_mgmt.evolve_plan_baselines (p. 939) function. • unapproved – A captured plan that you have not verified for use. For more information, see Evaluating Plan Performance (p. 930). • rejected – A plan that the optimizer won't use. For more information, see Rejecting or Disabling Slower Plans (p. 931). • preferred – A plan that you have determined is a preferred plan to use for a managed statement. <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 apg_plan_mgmt.set_plan_status (p. 941) 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 apg_plan_mgmt.evolve_plan_baselines (p. 939) function.</p>

Capturing Execution Plans

You can capture execution plans for specific SQL statements by using manual plan capture. Alternatively, you can capture all (or the slowest, or the most volatile) 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 restart for a new value to take effect. For more information, see the [apg_plan_mgmt.max_plans \(p. 935\)](#) parameter.

Topics

- [Manually Capturing Plans for Specific SQL Statements \(p. 925\)](#)
- [Automatically Capturing Plans \(p. 925\)](#)
- [Using Statistics to Identify SQL Statements \(p. 926\)](#)

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 \(p. 931\)](#). To compare the performance of the unapproved and approved plans and approve, reject, or delete them, see [Evaluating Plan Performance \(p. 930\)](#).

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 cannot be expressed in PostgreSQL SQL.

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

You can modify some parameters to capture plans for statements that have the greatest impact on throughput, run the longest, or have the most volatile performance. However, this extended mode of automatic plan capture has a noticeable performance overhead.

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 \(p. 930\)](#).

You can also identify which SQL statements to automatically capture based on statistical properties of the statements. For more information, see [Using Statistics to Identify SQL Statements \(p. 926\)](#).

Using Statistics to Identify SQL Statements

With automatic plan capture, you can identify the SQL statements that you want to manage by their statistical properties. To access SQL statements' statistics, install the `pg_stat_statements` extension.

Query plan management provides parameters that access these statistics. You set these parameters to supply criteria to the optimizer so it can identify which SQL statements to manage. During automatic plan capture, the optimizer captures plans for statements that meet the statistical criteria.

Use the following parameters to define statistical criteria for the SQL statements that you want to manage.

Parameter	Description
<code>apg_plan_mgmt.pgss_min_calls</code>	How many times the statement ran. For more information, see the apg_plan_mgmt.pgss_min_calls (p. 936) parameter.
<code>apg_plan_mgmt.pgss_min_total_time_ms</code>	The statement's total execution time. For more information, see the apg_plan_mgmt.pgss_min_total_time_ms (p. 936) parameter.
<code>apg_plan_mgmt.pgss_min_mean_time_ms</code>	The statement's average execution time. For more information, see the apg_plan_mgmt.pgss_min_mean_time_ms (p. 936) parameter.
<code>apg_plan_mgmt.pgss_min_stddev_time_ms</code>	The statement's standard deviation of execution time. For more information, see the apg_plan_mgmt.pgss_min_stddev_time_ms (p. 936) parameter.

Note

Your application's performance is affected if you use these statistics to identify which SQL statements to manage.

Before you can use statistics to identify which statements you want to manage, you need to install the `pg_stat_statements` extension. For installation and other details, see the [PostgreSQL pg_stat_statements documentation](#).

The following procedure shows how to identify SQL statements to manage based on statistical criteria and capture plans automatically for all matching statements.

To capture plans based on SQL statement statistics

Note

Set the following `apg_plan_mgmt` parameters in the parameter group for your DB instance, then restart your DB instance. For more about parameter groups, see [Modifying Parameters in a DB Parameter Group \(p. 291\)](#).

1. Turn on automatic plan capture by setting the `apg_plan_mgmt.capture_plan_baselines` parameter to `automatic`.
2. Set `apg_plan_mgmt.use_plan_baselines` to `true` to enforce the optimizer to use managed plans as more plans continue to be captured. Set a value of `false` if you only want to capture plans and not use them yet.
3. Set the statistical criteria to identify which SQL statements that you want to manage.

For example, setting `apg_plan_mgmt.pgss_min_calls` to 3 tells the optimizer to save plans only for statements that run at least 3 times. Setting `apg_plan_mgmt.pgss_min_total_time_ms` to 30000 tells the optimizer to save plans for statements that run for 30 seconds or more.

4. Restart your DB instance.
5. Enable the `pg_stat_statements` extension to make the statistics for the SQL statements available for this DB instance.

```
CREATE EXTENSION IF NOT EXISTS pg_stat_statements;
```

As the application runs, the optimizer captures plans for each matching statement.

Using Managed Plans

To get the optimizer to use captured plans for your managed statements, set the parameter `apg_plan_mgmt.use_plan_baselines` to `true`. The following is a local instance example.

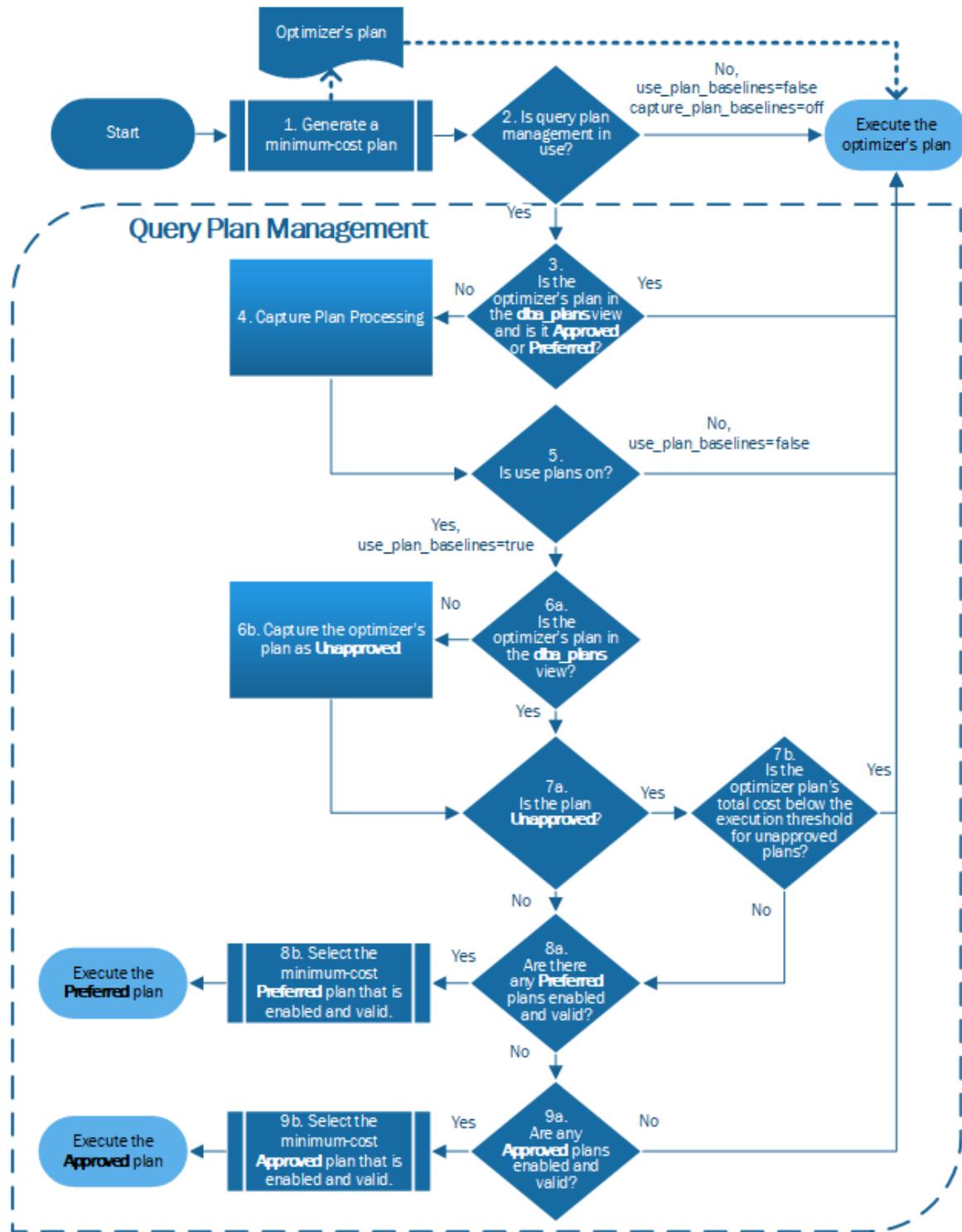
```
SET apg_plan_mgmt.use_plan_baselines = true;
```

While the application runs, this setting causes the optimizer to use the minimum-cost, preferred, or approved plan that is valid and enabled, for each managed statement.

How the Optimizer Chooses Which Plan to Run

The cost of an execution plan is an estimate that the optimizer makes to compare different plans. Optimizer cost is a function of several factors that include the CPU and I/O operations that the plan uses. For more information about PostgreSQL query planner costs, see the [PostgreSQL documentation on query planning](#).

The following flowchart shows how the query plan management optimizer chooses which plan to run.



The flow is as follows:

1. When the optimizer processes every SQL statement, it generates a minimum-cost plan.
2. Without query plan management, the optimizer simply runs its generated plan. The optimizer uses query plan management if you set one or both of the following parameter settings:

- `apg_plan_mgmt.capture_plan_baselines` to manual or automatic
 - `apg_plan_mgmt.use_plan_baselines` to true
3. The optimizer immediately runs the generated plan if the following are both true:
 - The optimizer's plan is already in the `apg_plan_mgmt.dba_plans` view for the SQL statement.
 - The plan's status is either approved or preferred.
 4. The optimizer goes through the capture plan processing if the parameter `apg_plan_mgmt.capture_plan_baselines` is manual or automatic.

For details on how the optimizer captures plans, see [Capturing Execution Plans \(p. 925\)](#).
 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 \(p. 937\)](#) 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 \(p. 925\)](#).

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 \(p. 930\)](#).

Maintaining Execution Plans

Query plan management provides techniques and functions to add, maintain, and improve execution plans.

Topics

- [Evaluating Plan Performance \(p. 930\)](#)
- [Validating Plans \(p. 931\)](#)
- [Fixing Plans Using pg_hint_plan \(p. 931\)](#)
- [Deleting Plans \(p. 933\)](#)
- [Exporting and Importing Plans \(p. 933\)](#)

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 \(p. 930\)](#)
- [Rejecting or Disabling Slower Plans \(p. 931\)](#)

Approving Better Plans

The following example demonstrates how to change the status of managed plans to approved using the `apg_plan_mgmt.evolve_plan_baselines` function.

```
SELECT apg_plan_mgmt.evolve_plan_baselines (
    sql_hash,
    plan_hash,
    min_speedup_factor := 1.0,
    action := 'approve'
)
FROM apg_plan_mgmt.dba_plans WHERE status = 'unapproved';

NOTICE:      rangequery (1,10000)
NOTICE:      Baseline   [ Planning time 0.761 ms, Execution time 13.261 ms]
NOTICE:      Baseline+1 [ Planning time 0.204 ms, Execution time 8.956 ms]
NOTICE:      Total time benefit: 4.862 ms, Execution time benefit: 4.305 ms
NOTICE:      Unapproved -> Approved
evolve_plan_baselines
-----
0
(1 row)
```

The output shows a performance report for the `rangequery` statement with parameter bindings of 1 and 10,000. The new unapproved plan (`Baseline+1`) is better than the best previously approved plan (`Baseline`). To confirm that the new plan is now approved, check the `apg_plan_mgmt.dba_plans` view.

```

SELECT sql_hash, plan_hash, status, enabled, stmt_name
FROM apg_plan_mgmt.dba_plans;

sql_hash | plan_hash | status | enabled | stmt_name
-----+-----+-----+-----+
1984047223 | 512153379 | Approved | t | rangequery
1984047223 | 512284451 | Approved | t | rangequery
(2 rows)

```

The managed plan now includes two approved plans that are the statement's plan baseline. You can also call the `apg_plan_mgmt.set_plan_status` function to directly set a plan's status field to '`approved`', '`rejected`', '`unapproved`', or '`preferred`'.

Rejecting or Disabling Slower Plans

To reject or disable plans, pass '`reject`' or '`disable`' as the action parameter to the `apg_plan_mgmt.evolve_plan_baselines` function. This example disables any captured unapproved plan that is slower by at least 10 percent than the best approved plan for the statement.

```

SELECT apg_plan_mgmt.evolve_plan_baselines(
    sql_hash, -- The managed statement ID
    plan_hash, -- The plan ID
    1.1, -- number of times faster the plan must be
    'disable' -- The action to take. This sets the enabled field to false.
)
FROM apg_plan_mgmt.dba_plans
WHERE status = 'Unapproved' AND -- plan is Unapproved
origin = 'Automatic'; -- plan was auto-captured

```

You can also directly set a plan to rejected or disabled. To directly set a plan's enabled field to `true` or `false`, call the `apg_plan_mgmt.set_plan_enabled` function. To directly set a plan's status field to '`approved`', '`rejected`', '`unapproved`', or '`preferred`', call the `apg_plan_mgmt.set_plan_status` function.

Validating Plans

Use the `apg_plan_mgmt.validate_plans` function to delete or disable plans that are invalid.

Plans can become invalid or stale when objects that they depend on are removed, such as an index or a table. However, a plan might be invalid only temporarily if the removed object gets recreated. If an invalid plan can become valid later, you might prefer to disable an invalid plan or do nothing rather than delete it.

To find and delete all plans that are invalid and haven't been used in the past week, use the `apg_plan_mgmt.validate_plans` function as follows.

```

SELECT apg_plan_mgmt.validate_plans(sql_hash, plan_hash, 'delete')
FROM apg_plan_mgmt.dba_plans
WHERE last_used < (current_date - interval '7 days');

```

To enable or disabled a plan directly, use the `apg_plan_mgmt.set_plan_enabled` function.

Fixing Plans Using pg_hint_plan

The query optimizer is well-designed to find an optimal plan for all statements, and in most cases the optimizer finds a good plan. However, occasionally you might know that a much better plan exists than

that generated by the optimizer. Two recommended ways to get the optimizer to generate a desired plan include using the `pg_hint_plan` extension or setting Grand Unified Configuration (GUC) variables in PostgreSQL:

- `pg_hint_plan` extension – Specify a “hint” to modify how the planner works by using PostgreSQL’s `pg_hint_plan` extension. To install and learn more about how to use the `pg_hint_plan` extension, see the [pg_hint_plan documentation](#).
- GUC variables – Override one or more cost model parameters or other optimizer parameters, such as the `fromCollapse_limit` or `GEOO_threshold`.

When you use one of these techniques to force the query optimizer to use a plan, you can also use query plan management to capture and enforce use of the new plan.

You can use the `pg_hint_plan` extension to change the join order, the join methods, or the access paths for a SQL statement. You use a SQL comment with special `pg_hint_plan` syntax to modify how the optimizer creates a plan. For example, assume the problem SQL statement has a two-way join.

```
SELECT *
FROM t1, t2
WHERE t1.id = t2.id;
```

Then suppose that the optimizer chooses the join order (`t1, t2`), but we know that the join order (`t2, t1`) is faster. The following hint forces the optimizer to use the faster join order, (`t2, t1`). Include EXPLAIN so that the optimizer generates a plan for the SQL statement but does not run the statement. (Output not shown.)

```
/*+ Leading ((t2 t1)) */ EXPLAIN SELECT *
FROM t1, t2
WHERE t1.id = t2.id;
```

The following steps show how to use `pg_hint_plan`.

To modify the optimizer’s generated plan and capture the plan using `pg_hint_plan`

1. Turn on the manual capture mode.

```
SET apg_plan_mgmt.capture_plan_baselines = manual;
```

2. Specify a hint for the SQL statement of interest.

```
/*+ Leading ((t2 t1)) */ EXPLAIN SELECT *
FROM t1, t2
WHERE t1.id = t2.id;
```

After this runs, the optimizer captures the plan in the `apg_plan_mgmt.dba_plans` view. The captured plan doesn’t include the special `pg_hint_plan` comment syntax because query plan management normalizes the statement by removing leading comments.

3. View the managed plans by using the `apg_plan_mgmt.dba_plans` view.

```
SELECT sql_hash, plan_hash, status, sql_text, plan_outline
FROM apg_plan_mgmt.dba_plans;
```

4. Set the status of the plan to `preferred`. Doing so makes sure that the optimizer chooses to run it, instead of selecting from the set of approved plans, when the minimum-cost plan isn’t already approved or `preferred`.

```
SELECT apg_plan_mgmt.set_plan_status(<sql-hash>, <plan-hash>, 'preferred' );
```

5. Turn off manual plan capture and enforce the use of managed plans.

```
SET apg_plan_mgmt.capture_plan_baselines = false;
SET apg_plan_mgmt.use_plan_baselines = true;
```

Now, when the original SQL statement runs, the optimizer chooses either an approved or preferred plan. If the minimum-cost plan isn't approved or preferred, then the optimizer chooses the preferred plan.

Deleting Plans

Delete plans that have not been used for a long time or that are no longer relevant. Each plan has a `last_used` date that the optimizer updates each time it executes a plan or picks it as the minimum-cost plan for a statement. Use the `last_used` date to determine if a plan has been used recently and is still relevant.

For example, you can use the `apg_plan_mgmt.delete_plan` function as follows. Doing this deletes all plans that haven't been chosen as the minimum-cost plan or haven't run in at least 31 days. However, this example doesn't delete plans that have been explicitly rejected.

```
SELECT SUM(apg_plan_mgmt.delete_plan(sql_hash, plan_hash))
FROM apg_plan_mgmt.dba_plans
WHERE last_used < (current_date - interval '31 days')
AND status <> 'rejected';
```

To delete any plan that is no longer valid and that you expect not to become valid again, use the `apg_plan_mgmt.validate_plans` function. For more information, see [Validating Plans \(p. 931\)](#).

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.

Important

If you don't clean up plans, you might eventually run out of shared memory that is set aside for query plan management. To control how much memory is available for managed plans, use the `apg_plan_mgmt.max_plans` parameter. Set this parameter in your DB instance-level parameter group and restart your DB instance for changes to take effect. For more information, see the [apg_plan_mgmt.max_plans \(p. 935\)](#) 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,
estimated_rows = EXCLUDED.estimated_rows,
actual_rows = EXCLUDED.actual_rows,
total_time_benefit_ms = EXCLUDED.total_time_benefit_ms,
execution_time_benefit_ms = EXCLUDED.execution_time_benefit_ms;
```

4. Reload the managed plans into shared memory and remove the temporary plans table.

```
SELECT apg_plan_mgmt.reload(); -- refresh shared memory
DROP TABLE plans_copy;
```

Parameter Reference for Query Plan Management

The apg_plan_mgmt extension provides the following parameters.

Parameters

- [apg_plan_mgmt.capture_plan_baselines \(p. 935\)](#)
- [apg_plan_mgmt.max_databases \(p. 935\)](#)
- [apg_plan_mgmt.max_plans \(p. 935\)](#)
- [apg_plan_mgmt.pgss_min_calls \(p. 936\)](#)
- [apg_plan_mgmt.pgss_min_mean_time_ms \(p. 936\)](#)
- [apg_plan_mgmt.pgss_min_stddev_time_ms \(p. 936\)](#)
- [apg_plan_mgmt.pgss_min_total_time_ms \(p. 936\)](#)
- [apg_plan_mgmt.plan_retention_period \(p. 937\)](#)

- [apg_plan_mgmt.unapproved_plan_execution_threshold \(p. 937\)](#)
- [apg_plan_mgmt.use_plan_baselines \(p. 937\)](#)

Set the query plan management parameters at the appropriate level:

- Set at the cluster-level to provide the same settings for all DB instances. For more information, see [Modifying Parameters in a DB Cluster Parameter Group \(p. 295\)](#).
- Set at the DB instance level to isolate the settings to an individual DB instance. For more information, see [Modifying Parameters in a DB Parameter Group \(p. 291\)](#).
- Set in a specific client session such as in psql, to isolate the values to only that session.

You must be set the parameters `apg_plan_mgmt.max_databases` and `apg_plan_mgmt.max_plans` at the cluster or DB instance level.

apg_plan_mgmt.capture_plan_baselines

Enable execution plan capture for SQL statements.

```
SET apg_plan_mgmt.capture_plan_baselines = [off | automatic |manual]
```

Value	Description
off	Disable plan capture. This is the default.
automatic	Enable plan capture for subsequent SQL statements that satisfy the eligibility criteria.
manual	Enable plan capture for subsequent SQL statements.

apg_plan_mgmt.max_databases

Sets the maximum number of database objects that might use query plan management. A database object is what gets created with the CREATE DATABASE SQL statement.

Important

Set `apg_plan_mgmt.max_databases` at the cluster or DB instance level. It requires a DB instance restart for a new value to take effect.

Value	Default	Description
Positive integer	10	A positive integer value.

apg_plan_mgmt.max_plans

Sets the maximum number of plans that might be captured in the `apg_plan_mgmt.dba_plans` view.

Important

Set `apg_plan_mgmt.max_plans` at the cluster or DB instance level. It requires a DB instance restart for a new value to take effect.

Value	Default	Description
integer	1000	A positive integer value greater or equal to 10.

apg_plan_mgmt.pgss_min_calls

Sets the minimum number of pg_stat_statements calls that are eligible for plan capture.

```
SET apg_plan_mgmt.pgss_min_calls = integer-value;
```

Value	Default	Description
Positive integer	2	A positive integer value greater or equal to 2.

Usage Notes

Requires installation of the pg_stat_statements extension. For more information, see the [PostgreSQL pg_stats_statements documentation](#).

apg_plan_mgmt.pgss_min_mean_time_ms

Minimum value of the pg_stat_statements mean_time to be eligible for plan capture.

```
SET apg_plan_mgmt.pgss_min_mean_time_ms = double-value;
```

Value	Default	Description
Positive number	0.0	A positive number value greater or equal to 0.0.

Usage Notes

Requires installation of the pg_stat_statements extension. For more information, see the [PostgreSQL pg_stats_statements documentation](#).

apg_plan_mgmt.pgss_min_stddev_time_ms

Minimum value of the pg_stat_statements stddev_time to be eligible for plan capture.

```
SET apg_plan_mgmt.pgss_min_stddev_time_ms = double-value;
```

Value	Default	Description
Positive number	0.0	A positive number value greater or equal to 0.0.

Usage Notes

Requires installation of the pg_stat_statements extension. For more information, see the [PostgreSQL pg_stats_statements documentation](#).

apg_plan_mgmt.pgss_min_total_time_ms

Minimum value of the pg_stat_statements total_time to be eligible for plan capture.

```
SET apg_plan_mgmt.pgss_min_total_time_ms = double-value;
```

Value	Default	Description
Positive number	0.0	A positive number value greater or equal to 0.0.

Usage Notes

Requires installation of the `pg_stat_statements` extension. For more information, see the [PostgreSQL pg_stats_statements documentation](#).

apg_plan_mgmt.plan_retention_period

The number of days plans are kept in the `apg_plan_mgmt.dba_plans` view before being automatically deleted. A plan is deleted when the current date is this many days since the plan's `last_used` date.

```
SET apg_plan_mgmt.plan_retention_period_ = integer-value;
```

Value	Default	Description
Positive integer	32	A positive integer value greater or equal to 32, representing days.

apg_plan_mgmt.unapproved_plan_execution_threshold

An estimated total plan cost threshold, below which the optimizer runs an unapproved plan. By default, the optimizer does not run unapproved plans. However, you can set an execution threshold for your fastest unapproved plans. With this setting, the optimizer bypasses the overhead of enforcing only approved plans.

```
SET apg_plan_mgmt.unapproved_plan_execution_threshold = integer-value;
```

Value	Default	Description
Positive integer	0	A positive integer value greater or equal to 0. A value of 0 means no unapproved plans run when <code>use_plan_baselines</code> is <code>true</code> .

With the following example, the optimizer runs an unapproved plan if the estimated cost is less than 550, even if `use_plan_baselines` is `true`.

```
SET apg_plan_mgmt.unapproved_plan_execution_threshold = 550;
```

apg_plan_mgmt.use_plan_baselines

Enforce the optimizer to use managed plans for managed statements.

```
SET apg_plan_mgmt.use_plan_baselines = [true | false];
```

Value	Description
<code>true</code>	<p>Enforce the use of managed plans. When a SQL statement runs and it is a managed statement in the <code>apg_plan_mgmt.dba_plans</code> view, the optimizer chooses a managed plan in the following order.</p> <ol style="list-style-type: none"> 1. The minimum-cost preferred plan that is valid and enabled. 2. The minimum cost approved plan that is valid and enabled. 3. The minimum cost unapproved plan that is valid, enabled, and that meets the threshold, if set with the <code>apg_plan_mgmt.unapproved_plan_execution_threshold</code> parameter. 4. The optimizer's generated minimum-cost plan.
<code>false</code>	(Default) Do not use managed plans. The optimizer uses its generated minimum-cost plan.

Usage Notes

When `use_plan_baselines` is `true`, then the optimizer makes the following execution decisions:

1. If the estimated cost of the optimizer's plan is below the `unapproved_plan_execution_threshold`, then execute it, else
2. If the plan is `approved` or `preferred`, then execute it, else
3. Execute a `minimum-cost preferred` plan, if possible, else
4. Execute a `minimum-cost approved` plan, if possible, else
5. Execute the optimizer's minimum-cost plan.

Function Reference for Query Plan Management

The `apg_plan_mgmt` extension provides the following functions.

Functions

- [apg_plan_mgmt.delete_plan \(p. 938\)](#)
- [apg_plan_mgmt.evolve_plan_baselines \(p. 939\)](#)
- [apg_plan_mgmt.plan_last_used \(p. 940\)](#)
- [apg_plan_mgmt.reload \(p. 941\)](#)
- [apg_plan_mgmt.set_plan_enabled \(p. 941\)](#)
- [apg_plan_mgmt.set_plan_status \(p. 941\)](#)
- [apg_plan_mgmt.validate_plans \(p. 942\)](#)

`apg_plan_mgmt.delete_plan`

Delete a managed plan.

Syntax

```
apg_plan_mgmt.delete_plan(
    sql_hash,
    plan_hash
)
```

Return Value

Returns 0 if the delete was successful or -1 if the delete failed.

Parameters

Parameter	Description
sql_hash	The <code>sql_hash</code> ID of the plan's managed SQL statement.
plan_hash	The managed plan's <code>plan_hash</code> ID.

apg_plan_mgmt.evolve_plan_baselines

Verifies whether an already approved plan is faster or whether a plan identified by the query optimizer as a minimum cost plan is faster.

Syntax

```
apg_plan_mgmt.evolve_plan_baselines(
    sql_hash,
    plan_hash,
    min_speedup_factor,
    action
)
```

Return Value

The number of plans that were not faster than the best approved plan.

Parameters

Parameter	Description
sql_hash	The <code>sql_hash</code> ID of the plan's managed SQL statement.
plan_hash	The managed plan's <code>plan_hash</code> ID. Use NULL to mean all plans that have the same <code>sql_hash</code> ID value.
min_speedup_factor	The <i>minimum speedup factor</i> can be the number of times faster that a plan must be than the best of the already approved plans to approve it. Alternatively, this factor can be the number of times slower that a plan must be to reject or disable it. This is a positive float value.
action	The action the function is to perform. Valid values include the following. Case does not matter. <ul style="list-style-type: none"> • 'disable' – Disable each matching plan that does not meet the minimum speedup factor. • 'approve' – Enable each matching plan that meets the minimum speedup factor and set its status to <code>approved</code>. • 'reject' – For each matching plan that does not meet the minimum speedup factor, set its status to <code>rejected</code>.

Parameter	Description
	<ul style="list-style-type: none"> • NULL – The function simply returns the number of plans that have no performance benefit because they do not meet the minimum speedup factor.

Usage Notes

Set specified plans to approved, rejected, or disabled based on whether the planning plus execution time is faster than the best approved plan by a factor that you can set. The action parameter might be set to 'approve' or 'reject' to automatically approve or reject a plan that meets the performance criteria. Alternatively, it might be set to "" (empty string) to do the performance experiment and produce a report, but take no action.

You can avoid pointlessly rerunning of the `apg_plan_mgmt.evolve_plan_baselines` function for a plan on which it was recently run. To do so, restrict the plans to just the recently created unapproved plans. Alternatively, you can avoid running the `apg_plan_mgmt.evolve_plan_baselines` function on any approved plan that has a recent `last_verified` timestamp.

Conduct a performance experiment to compare the planning plus execution time of each plan relative to the other plans in the baseline. In some cases, there is only one plan for a statement and the plan is approved. In such a case, compare the planning plus execution time of the plan to the planning plus execution time of using no plan.

The incremental benefit (or disadvantage) of each plan is recorded in the `apg_plan_mgmt.dba_plans` view in the `total_time_benefit_ms` column. When this value is positive, there is a measurable performance advantage to including this plan in the baseline.

In addition to collecting the planning and execution time of each candidate plan, the `last_verified` column of the `apg_plan_mgmt.dba_plans` view is updated with the `current_timestamp`. The `last_verified` timestamp might be used to avoid running this function again on a plan that recently had its performance verified.

apg_plan_mgmt.plan_last_used

Returns the `last_used` date of the specified plan from shared memory.

Syntax

```
apg_plan_mgmt.plan_last_used(
    sql_hash,
    plan_hash
)
```

Return Value

Returns the `last_used` date.

Parameters

Parameter	Description
<code>sql_hash</code>	The <code>sql_hash</code> ID of the plan's managed SQL statement.
<code>plan_hash</code>	The managed plan's <code>plan_hash</code> ID.

apg_plan_mgmt.reload

Reload plans into shared memory from the `apg_plan_mgmt.dba_plans` view.

Syntax

```
apg_plan_mgmt.reload()
```

Return Value

None.

Parameters

None.

Usage Notes

Call `reload` for the following situations:

- Use it to refresh the shared memory of a read-only replica immediately, rather than wait for new plans to propagate to the replica.
- Use it after importing managed plans.

apg_plan_mgmt.set_plan_enabled

Enable or disable a managed plan.

Syntax

```
apg_plan_mgmt.set_plan_enabled(
    sql_hash,
    plan_hash,
    [true | false]
)
```

Return Value

Returns 0 if the setting was successful or -1 if the setting failed.

Parameters

Parameter	Description
<code>sql_hash</code>	The <code>sql_hash</code> ID of the plan's managed SQL statement.
<code>plan_hash</code>	The managed plan's <code>plan_hash</code> ID.
<code>enabled</code>	Boolean value of true or false: <ul style="list-style-type: none">• A value of <code>true</code> enables the plan.• A value of <code>false</code> disables the plan.

apg_plan_mgmt.set_plan_status

Set a managed plan's status to approved, unapproved, rejected, or preferred.

Syntax

```
apg_plan_mgmt.set_plan_status(
    sql_hash,
    plan_hash,
    status
)
```

Return Value

Returns 0 if the setting was successful or -1 if the setting failed.

Parameters

Parameter	Description
sql_hash	The <code>sql_hash</code> ID of the plan's managed SQL statement.
plan_hash	The managed plan's <code>plan_hash</code> ID.
status	A string with one of the following values; case does not matter: <ul style="list-style-type: none"> • <code>approved</code> • <code>unapproved</code> • <code>rejected</code> • <code>preferred</code> For more information about these values, see status in Reference for the <code>apg_plan_mgmt.dba_plans</code> View (p. 922) .

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.

Parameter	Description
plan_hash	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.
action	<p>The action the function is to perform for invalid plans. Valid string values include the following. Case does not matter.</p> <ul style="list-style-type: none"> • '<code>disable</code>' – Each invalid plan is disabled. • '<code>delete</code>' – Each invalid plan is deleted. • <code>NULL</code> – The function simply returns the number of invalid plans. No other action is performed. • <code>"</code> – An empty string produces a message indicating the number of both valid and invalid plans. <p>Any other value is treated like the empty string.</p>

Usage Notes

Use the form `validate_plans(action)` to validate all the managed plans for all the managed statements in the entire `apg_plan_mgmt.dba_plans` view.

Use the form `validate_plans(sql_hash, plan_hash, action)` to validate a managed plan specified with `plan_hash`, for a managed statement specified with `sql_hash`.

Use the form `validate_plans(sql_hash, NULL, action)` to validate all the managed plans for the managed statement specified with `sql_hash`.

Publishing Aurora PostgreSQL Logs to Amazon CloudWatch Logs

You can configure your Aurora PostgreSQL DB cluster to publish log data to a log group in Amazon CloudWatch Logs. With CloudWatch Logs, you can perform real-time analysis of the log data, and use CloudWatch to create alarms and view metrics. You can use CloudWatch Logs to store your log records in highly durable storage.

Note

Be aware of the following:

- Aurora PostgreSQL supports publishing logs to CloudWatch Logs for versions 9.6.12 and above and versions 10.7 and above.
- 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 \(p. 728\)](#). 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 \(p. 235\)](#).

- 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.

Console

You can publish Aurora PostgreSQL logs to CloudWatch Logs with the console.

To publish Aurora PostgreSQL logs from the console

1. Open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. In the navigation pane, choose **Databases**.
3. Choose the Aurora PostgreSQL DB cluster that you want to publish the log data for.
4. Choose **Modify**.
5. In the **Log exports** section, choose the logs that you want to start publishing to CloudWatch Logs.
6. Choose **Continue**, and then choose **Modify cluster** on the summary page.

AWS CLI

You can publish Aurora PostgreSQL logs with the AWS CLI. You can run the `modify-db-cluster` AWS CLI command with the following options:

- `--db-cluster-identifier`—The DB cluster identifier.
- `--cloudwatch-logs-export-configuration`—The configuration setting for the log types to be enabled 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 modifies an existing Aurora PostgreSQL DB cluster to publish log files to CloudWatch Logs.

For Linux, OS X, or Unix:

```
aws rds modify-db-cluster \
--db-cluster-identifier my-db-cluster \
--cloudwatch-logs-export-configuration '{"EnableLogTypes":["postgresql", "upgrade"]}'
```

For Windows:

```
aws rds modify-db-cluster ^
--db-cluster-identifier my-db-cluster ^
--cloudwatch-logs-export-configuration '{"EnableLogTypes":["postgresql", "upgrade"]}'
```

Example

The following command creates an Aurora PostgreSQL DB cluster to publish log files to CloudWatch Logs.

For Linux, OS X, or Unix:

```
aws rds create-db-cluster \
--db-cluster-identifier my-db-cluster \
--engine aurora-postgresql \
--cloudwatch-logs-export-configuration '{"EnableLogTypes":["postgresql", "upgrade"]}'
```

For Windows:

```
aws rds create-db-cluster ^
--db-cluster-identifier my-db-cluster ^
--engine aurora-postgresql ^
--cloudwatch-logs-export-configuration '{"EnableLogTypes":["postgresql", "upgrade"]}'
```

RDS API

You can publish Aurora PostgreSQL logs with the RDS API. You can run the [ModifyDBCluster](#) action 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 actions:

- [CreateDBCluster](#)
- [RestoreDBClusterFromS3](#)
- [RestoreDBClusterFromSnapshot](#)
- [RestoreDBClusterToPointInTime](#)

Run the RDS API action with the following parameters:

- **DBClusterIdentifier**—The DB cluster identifier.
- **Engine**—The database engine.
- **EnableCloudwatchLogsExports**—The configuration setting for the log types to be enabled for export to CloudWatch Logs for the DB cluster.

Other parameters might be required depending on the AWS CLI command that you run.

Monitoring Log Events in Amazon CloudWatch

After enabling Aurora PostgreSQL log events, you can monitor the events in Amazon CloudWatch Logs. For more information about monitoring, see [View Log Data Sent to CloudWatch Logs](#).

A new log group is automatically created for the Aurora DB cluster under the following prefix, in 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 `postgresql` log for a DB cluster named `my-db-cluster`, PostgreSQL log data is stored in the `/aws/rds/cluster/my-db-cluster/postgresql` log group.

All of the events from all of the DB instances in a DB cluster are pushed to a log group using different log streams.

If a log group with the specified name exists, Aurora uses that log group to export log data for the Aurora DB cluster. You can use automated configuration, such as AWS CloudFormation, to create log groups with predefined log retention periods, metric filters, and customer access. Otherwise, a new log group is automatically created using the default log retention period, **Never Expire**, in CloudWatch Logs. You can use the CloudWatch Logs console, the AWS CLI, or the CloudWatch Logs API to change the log retention period. For more information about changing log retention periods in CloudWatch Logs, see [Change Log Data Retention in CloudWatch Logs](#).

You can use the CloudWatch Logs console, the AWS CLI, or the CloudWatch Logs API to search for information within the log events for a DB cluster. For more information about searching and filtering log data, see [Searching and Filtering Log Data](#).

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.

Topics

- [Configuring Cluster Cache Management \(p. 947\)](#)
- [Monitoring the Buffer Cache \(p. 949\)](#)

Configuring Cluster Cache Management

Note

Cluster cache management is supported for Aurora PostgreSQL DB clusters of versions 9.6.11 and above, and versions 10.5 and above.

To configure cluster cache management, take the following steps.

Configuration Steps

- [Enable Cluster Cache Management \(p. 947\)](#)
- [Set the Promotion Tier Priority for the Writer DB Instance \(p. 948\)](#)
- [Set the Promotion Tier Priority for a Reader DB Instance \(p. 949\)](#)

Note

Allow at least 1 minute after completing these steps for cluster cache management to be fully operational.

Enable Cluster Cache Management

To enable cluster cache management for a DB cluster, modify its parameter group by setting the `apg_ccm_enabled` parameter to 1 as 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, OS X, 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"
```

Set the Promotion Tier Priority for the Writer DB Instance

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 \(p. 380\)](#).

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 **Failover** panel, choose **tier-0** for the **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, OS X, 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
```

Set 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 as the writer DB instance. 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 **Failover** panel, choose **tier-0** for the **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, OS X, 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();
```

Upgrading an Aurora PostgreSQL DB Cluster Engine Version

Amazon Aurora provides newer versions of each supported database engine so you can keep your DB cluster up-to-date. Newer versions can include bug fixes, security enhancements, and other improvements for the database engine. When Amazon Aurora supports a new version of a database engine, you can choose how and when to upgrade your database DB clusters.

There are two kinds of upgrades: major version upgrades and minor version upgrades. In general, a *major engine version upgrade* can introduce changes that are not compatible with existing applications. In contrast, a *minor version upgrade* includes only changes that are backward-compatible with existing applications.

Note

Aurora PostgreSQL does not currently support in-place major version upgrades. To migrate a database from one major version to another you can use dump and load tools such as the PostgreSQL utilities `pg_dump` and `pg_restore`.

The version numbering sequence is specific to each database engine. For example, Aurora PostgreSQL 9.6 and 10.5 are major engine versions and upgrading from any 9.6 version to any 10.x version is a major version upgrade. Aurora PostgreSQL version 9.6.8 and 9.6.9 are minor versions and upgrading from 9.6.8 to 9.6.9 is a minor version upgrade. To determine the version of an Aurora DB cluster, follow the instructions in [Amazon Aurora Updates \(p. 432\)](#).

Note

A PostgreSQL engine upgrade doesn't upgrade any PostgreSQL extensions. For more information, see [Upgrading PostgreSQL Extensions \(p. 953\)](#).

Topics

- [Manually Upgrading the Minor Engine Version \(p. 951\)](#)
- [Automatically Upgrading the Minor Engine Version \(p. 952\)](#)
- [Upgrading PostgreSQL Extensions \(p. 953\)](#)

Manually Upgrading the Minor Engine Version

To perform a minor version upgrade of an Aurora PostgreSQL DB cluster, use the following instructions for the AWS Management Console, the AWS CLI, or the RDS API.

Console

To upgrade the engine version of a DB cluster by using the console

1. Sign in to the AWS Management Console and open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. In the navigation pane, choose **Databases**, and then choose the DB cluster that you want to upgrade.
3. Choose **Modify**. The **Modify DB cluster** page appears.
4. For **DB 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 \(p. 264\)](#).
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 upgrade the engine version of a DB cluster, use the CLI `modify-db-cluster` command. Specify the following parameters:

- `--db-cluster-identifier` – the name of the DB cluster.
- `--engine-version` – the version number of the database engine to upgrade to. For information about valid engine versions, use the AWS CLI `describe-db-engine-versions` command.
- `--no-apply-immediately` – apply changes during the next maintenance window. To apply changes immediately, use `--apply-immediately`.

Example

For Linux, OS X, 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](#) action. 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`.

Automatically Upgrading the Minor Engine Version

A *minor engine version* is an update to a DB engine version within a major engine version. For example, a major engine version might be 9.6 with the minor engine versions 9.6.11 and 9.6.12 within it.

If you want Amazon Aurora to upgrade the DB engine version of a database automatically, you can enable auto minor version upgrades for the database. When a minor engine version is designated as the preferred minor engine version, each database that meets both of the following conditions is upgraded to the minor engine version automatically:

- The database is running a minor version of the DB engine that is lower than the preferred minor engine version.
- The database has auto minor version upgrade enabled.

You can control whether auto minor version upgrade is enabled for a DB instance when you perform the following tasks:

- [Creating an Amazon Aurora DB cluster \(p. 100\)](#)
- [Modifying an Amazon Aurora DB cluster \(p. 264\)](#)

Note

Modify the writer DB instance to control whether auto minor version upgrade is enabled for the entire DB cluster.

- [Restoring a DB cluster from a snapshot \(p. 385\)](#)

Note

When you restore a DB cluster from a snapshot, you can control whether auto minor version upgrade is enabled for the DB cluster only when you use the console.

- [Restoring a DB cluster to a specific time \(p. 405\)](#)

Note

When you restore a DB cluster to a specific time, you can control whether auto minor version upgrade is enabled for the DB cluster only when you use the console.

When you perform these tasks, you can control whether auto minor version upgrade is enabled for the DB cluster in the following ways:

- Using the console, set the **Auto minor version upgrade** option.

- Using the AWS CLI, set the `--auto-minor-version-upgrade` | `--no-auto-minor-version-upgrade` option.
- Using the RDS API, set the `AutoMinorVersionUpgrade` parameter.

To determine whether a maintenance update, such as a DB engine version upgrade, is available for your DB cluster, you can use the console, AWS CLI, or RDS API. You can also upgrade the DB engine version manually and adjust the maintenance window. For more information, see [Maintaining an Amazon Aurora DB Cluster \(p. 408\)](#).

Upgrading PostgreSQL Extensions

A PostgreSQL engine upgrade doesn't upgrade any PostgreSQL extensions. To update an extension after an engine upgrade, use the `ALTER EXTENSION UPDATE` command.

Note

If you are running the `PostGIS` extension in your Amazon RDS PostgreSQL DB instance, make sure that you follow the [PostGIS upgrade instructions](#) in the PostGIS documentation before you upgrade the extension.

To upgrade an extension, use the following 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_extensions` view in the following command:

```
SELECT * FROM pg_available_extensions;
```

Best Practices with Amazon Aurora PostgreSQL

This topic includes information on best practices and options for using or migrating data to an Amazon Aurora PostgreSQL DB cluster.

Fast Failover with Amazon Aurora PostgreSQL

There are several things you can do to make a failover perform faster with Aurora PostgreSQL. This section discusses each of the following ways:

- Aggressively set TCP keepalives to ensure that longer running queries that are waiting for a server response will be killed before the read timeout expires in the event of a failure.
- Set the Java DNS caching timeouts aggressively to ensure the Aurora Read-Only Endpoint can properly cycle through read-only nodes on subsequent connection attempts.
- Set the timeout variables used in the JDBC connection string as low as possible. Use separate connection objects for short and long running queries.
- Use the provided read and write Aurora endpoints to establish a connection to the cluster.
- Use RDS APIs to test application response on server side failures and use a packet dropping tool to test application response for client-side failures.

Setting TCP Keepalives Parameters

The TCP keepalive process is simple: when you set up a TCP connection, you associate a set of timers. When the keepalive timer reaches zero, you send a keepalive probe packet. If you receive a reply to your keepalive probe, you can assume that the connection is still up and running.

Enabling TCP keepalive parameters and setting them aggressively ensures that if your client is no longer able to connect to the database, then any active connections are quickly closed. This action allows the application to react appropriately, such as by picking a new host to connect to.

The following TCP keepalive parameters need to be set:

- `tcp_keepalive_time` controls the time, in seconds, after which a keepalive packet is sent when no data has been sent by the socket (ACKs are not considered data). We recommend the following setting:

```
tcp_keepalive_time = 1
```

- `tcp_keepalive_intvl` controls the time, in seconds, between sending subsequent keepalive packets after the initial packet is sent (set using the `tcp_keepalive_time` parameter). We recommend the following setting:

```
tcp_keepalive_intvl = 1
```

- `tcp_keepalive_probes` is the number of unacknowledged keepalive probes that occur before the application is notified. We recommend the following setting:

```
tcp_keepalive_probes = 5
```

These settings should notify the application within five seconds when the database stops responding. A higher `tcp_keepalive_probes` value can be set if keepalive packets are often dropped within the application's network. This subsequently increases the time it takes to detect an actual failure, but allows for more buffer in less reliable networks.

Setting TCP keepalive parameters on Linux

1. When testing how to configure the TCP keepalive parameters, we recommend doing so via the command line with the following commands: This suggested configuration is system wide, meaning that it affects all other applications that create sockets with the SO_KEEPALIVE option on.

```
sudo sysctl net.ipv4.tcp_keepalive_time=1
sudo sysctl net.ipv4.tcp_keepalive_intvl=1
sudo sysctl net.ipv4.tcp_keepalive_probes=5
```

2. Once you've found a configuration that works for your application, these settings must be persisted by adding the following lines (including any changes you made) to `/etc/sysctl.conf`:

```
tcp_keepalive_time = 1
tcp_keepalive_intvl = 1
tcp_keepalive_probes = 5
```

For information on setting TCP keepalive parameters on Windows, see [Things You May Want to Know About TCP Keepalive](#).

Configuring Your Application for Fast Failover

This section discusses several Aurora PostgreSQL specific configuration changes you can make. Documentation for general setup and configuration of the JDBC driver is available from the [PostgreSQL JDBC site](#).

Reducing DNS Cache Timeouts

When your application tries to establish a connection after a failover, the new Aurora PostgreSQL writer will be a previous reader, which can be found using the Aurora **read only** endpoint before DNS updates have fully propagated. Setting the java DNS TTL to a low value helps cycle between reader nodes on subsequent connection attempts.

```
// Sets internal TTL to match the Aurora RO Endpoint TTL
java.security.Security.setProperty("networkaddress.cache.ttl" , "1");
// If the lookup fails, default to something like small to retry
java.security.Security.setProperty("networkaddress.cache.negative.ttl" , "3");
```

Setting an Aurora PostgreSQL Connection String for Fast Failover

To make use of Aurora PostgreSQL fast failover, your application's connection string should have a list of hosts (highlighted in bold in the following example) instead of just a single host. Here is an example connection string you could use to connect to an Aurora PostgreSQL cluster:

```
jdbc:postgresql://myauroracluster.cluster-c9bfei4hj1rd.us-east-1-beta.rds.amazonaws.com:5432,
myauroracluster.cluster-ro-c9bfei4hj1rd.us-east-1-beta.rds.amazonaws.com:5432
/postgres?user=<masteruser>&password=<masterpw>&loginTimeout=2
&connectTimeout=2&cancelSignalTimeout=2&socketTimeout=60
&tcpKeepAlive=true&targetServerType=master&loadBalanceHosts=true
```

For best availability and to avoid a dependency on the RDS API, the best option for connecting is to maintain a file with a host string that your application reads from when you establish a connection to the database. This host string would have all the Aurora endpoints available for the cluster. For more information about Aurora endpoints, see [Amazon Aurora Connection Management \(p. 4\)](#). For example, you could store the endpoints in a file locally like the following:

```
myauroracluster.cluster-c9bfei4hj1rd.us-east-1-beta.rds.amazonaws.com:5432,
myauroracluster.cluster-ro-c9bfei4hj1rd.us-east-1-beta.rds.amazonaws.com:5432
```

Your application would read from this file to populate the host section of the JDBC connection string. Renaming the DB Cluster causes these endpoints to change; ensure that your application handles that event should it occur.

Another option is to use a list of DB instance nodes:

```
my-node1.cksc6xlmwcyw.us-east-1-beta.rds.amazonaws.com:5432,
my-node2.cksc6xlmwcyw.us-east-1-beta.rds.amazonaws.com:5432,
my-node3.cksc6xlmwcyw.us-east-1-beta.rds.amazonaws.com:5432,
my-node4.cksc6xlmwcyw.us-east-1-beta.rds.amazonaws.com:5432
```

The benefit of this approach is that the PostgreSQL JDBC connection driver will loop through all nodes on this list to find a valid connection, whereas when using the Aurora endpoints only two nodes will be tried per connection attempt. The downside of using DB instance nodes is that if you add or remove nodes from your cluster and the list of instance endpoints becomes stale, the connection driver may never find the correct host to connect to.

Setting the following parameters aggressively helps ensure that your application doesn't wait too long to connect to any one host.

- `loginTimeout` - Controls how long your application waits to login to the database *after* a socket connection has been established.
- `connectTimeout` - Controls how long the socket waits to establish a connection to the database.

Other application parameters can be modified to speed up the connection process, depending on how aggressive you want your application to be.

- `cancelSignalTimeout` - In some applications, you may want to send a "best effort" cancel signal on a query that has timed out. If this cancel signal is in your failover path, you should consider setting it aggressively to avoid sending this signal to a dead host.
- `socketTimeout` - This parameter controls how long the socket waits for read operations. This parameter can be used as a global "query timeout" to ensure no query waits longer than this value. A good practice is to have one connection handler that runs short lived queries and sets this value lower, and to have another connection handler for long running queries with this value set much higher. Then, you can rely on TCP keepalive parameters to kill long running queries if the server goes down.
- `tcpKeepAlive` - Enable this parameter to ensure the TCP keepalive parameters that you set are respected.
- `targetServerType` - This parameter can be used to control whether the driver connects to a read (slave) or write (master) node. Possible values are: `any`, `master`, `slave` and `preferSlave`. The `preferSlave` value attempts to establish a connection to a reader first but falls back and connects to the writer if no reader connection can be established.
- `loadBalanceHosts` - When set to `true`, this parameter has the application connect to a random host chosen from a list of candidate hosts.

Other Options for Obtaining The Host String

You can get the host string from several sources, including the `aurora_replica_status` function and by using the Amazon RDS API.

Your application can connect to any DB instance in the DB Cluster and query the `aurora_replica_status` function to determine who the writer of the cluster is, or to find any other reader nodes in the cluster. You can use this function to reduce the amount of time it takes to find a host to connect to, though in certain scenarios the `aurora_replica_status` function may show out of date or incomplete information in certain network failure scenarios.

A good way to ensure your application can find a node to connect to is to attempt to connect to the **cluster writerendpoint** and then the **cluster readerendpoint** until you can establish a readable connection. These endpoints do not change unless you rename your DB Cluster, and thus can generally be left as static members of your application or stored in a resource file that your application reads from.

Once you establish a connection using one of these endpoints, you can call the `aurora_replica_status` function to get information about the rest of the cluster. For example, the following command retrieves information with the `aurora_replica_status` function.

```
postgres=> select server_id, session_id, highest_lsn_rcvd,
cur_replay_latency_in_usecs, now(), last_update_timestamp from
aurora_replica_status();
   server_id   |           session_id           | 
   vd1          | highest_lsn_rcvd | cur_replay_latency | 
   now          |           last_update_time | 
-----+-----+-----+-----+
-----+-----+-----+-----+
-----+-----+
      mynode-1 | 3e3c5044-02e2-11e7-b70d-95172646d6ca | 
594220999 | 594221001 | 201421 | 2017-03-07
19:50:24.695322+00 | 2017-03-07 19:50:23+00
      mynode-2 | 1efdd188-02e4-11e7-becd-f12d7c88a28a | 
594220999 | 594221001 | 201350 | 2017-03-07
19:50:24.695322+00 | 2017-03-07 19:50:23+00
      mynode-3 |           MASTER_SESSION_ID | 
594220999 |           | 2017-03-07
19:50:24.695322+00 | 2017-03-07 19:50:23+00
```

(3 rows)

So for example, the hosts section of your connection string could start with both the writer and reader cluster endpoints:

```
myauroracluster.cluster-c9bfei4hjlr.us-east-1-beta.rds.amazonaws.com:5432,  
myauroracluster.cluster-ro-c9bfei4hjlr.us-east-1-beta.rds.amazonaws.com:5432
```

In this scenario, your application would attempt to establish a connection to any node type, master or slave. Once connected, a good practice is to first examine the read-write status of the node by querying for the result of the command `SHOW transaction_read_only`.

If the return value of the query is OFF, then you've successfully connected to the master node. If the return value is ON, and your application requires a read-write connection, you can then call the `aurora_replica_status` function to determine the `server_id` that has `session_id='MASTER_SESSION_ID'`. This function gives you the name of the master node. You can use this in conjunction with the 'endpointPostfix' described below.

One thing to watch out for is when you connect to a replica that has data that has become stale. When this happens, the `aurora_replica_status` function may show out-of-date information. A threshold for staleness can be set at the application level and examined by looking at the difference between the server time and the `last_update_time`. In general, your application should be sure to avoid flip-flopping between two hosts due to conflicting information returned by the `aurora_replica_status` function. That is, your application should err on the side of trying all known hosts first instead of blindly following the data returned by the `aurora_replica_status` function.

RDS API

You can programmatically find the list of instances by using the [AWS Java SDK](#), specifically the [DescribeDbClusters](#) API. Here's a small example of how you might do this in java 8:

```
AmazonRDS client = AmazonRDSClientBuilder.defaultClient();  
DescribeDBClustersRequest request = new DescribeDBClustersRequest()  
    .withDBClusterIdentifier(clusterName);  
DescribeDBClustersResult result =  
rdsClient.describeDBClusters(request);  
  
DBCluster singleClusterResult = result.getDBClusters().get(0);  
  
String pgJDBCEndpointStr =  
singleClusterResult.getDBClusterMembers().stream()  
    .sorted(Comparator.comparing(DBClusterMember::getIsClusterWriter)  
    .reversed()) // This puts the writer at the front of the list  
    .map(m -> m.getDBInstanceIdentifier() + endpointPostfix + ":" +  
singleClusterResult.getPort())  
    .collect(Collectors.joining(", "));
```

`pgJDBCEndpointStr` will contain a formatted list of endpoints, e.g:

```
my-node1.cksc6xlmwcyw.us-east-1-beta.rds.amazonaws.com:5432,  
my-node2.cksc6xlmwcyw.us-east-1-beta.rds.amazonaws.com:5432
```

The variable 'endpointPostfix' can be a constant that your application sets, or can be obtained by querying the `DescribeDBInstances` API for a single instance in your cluster. This value remains constant within a region and for an individual customer, so it would save an API call to simply keep this constant in a resource file that your application reads from. In the example above, it would be set to:

```
.cksc6xlmwcyw.us-east-1-beta.rds.amazonaws.com
```

For availability purposes, a good practice would be to default to using the [Aurora Endpoints](#) of your DB Cluster if the API is not responding, or taking too long to respond. The endpoints are guaranteed to be up to date within the time it takes to update the DNS record (typically less than 30 seconds). This again can be stored in a resource file that your application consumes.

Testing Failover

In all cases you must have a DB Cluster with ≥ 2 DB instances in it.

From the server side, certain APIs can cause an outage that can be used to test how your applications responds:

- [FailoverDBCluster](#) - Will attempt to promote a new DB Instance in your DB Cluster to writer

```
public void causeFailover() {  
    /*  
     * See http://docs.aws.amazon.com/sdk-for-java/v1/developer-guide/basics.html for  
     * more details on setting up an RDS client  
     */  
    final AmazonRDS rdsClient = AmazonRDSClientBuilder.defaultClient();  
  
    FailoverDBClusterRequest request = new FailoverDBClusterRequest();  
    request.setDBClusterIdentifier("cluster-identifier");  
  
    rdsClient.failoverDBCluster(request);  
}
```

- [RebootDBInstance](#) - Failover is not guaranteed in this API. It will shutdown the database on the writer, though, and can be used to test how your application responds to connections dropping (note that the **ForceFailover** parameter is not applicable for Aurora engines and instead should use the FailoverDBCluster API)
- [ModifyDBCluster](#) - Modifying the **Port** will cause an outage when the nodes in the cluster begin listening on a new port. In general your application can respond to this failure by ensuring that only your application controls port changes and can appropriately update the endpoints it depends on, by having someone manually update the port when they make modifications at the API level, or by querying the RDS API in your application to determine if the port has changed.
- [ModifyDBInstance](#) - Modifying the **DBInstanceClass** will cause an outage
- [DeleteDBInstance](#) - Deleting the master/writer will cause a new DB Instance to be promoted to writer in your DB Cluster

From the application/client side, if using Linux, you can test how the application responds to sudden packet drops based on port, host, or if tcp keepalive packets are not sent or received by using iptables.

Fast Failover Example

The following code sample shows how an application might set up an Aurora PostgreSQL driver manager. The application would call `getConnection(...)` when it needed a connection. A call to that function can fail to find a valid host, such as when no writer is found but the `targetServerType` was set to "master"), and the calling application should simply retry. This can easily be wrapped into a connection pooler to avoid pushing the retry behavior onto the application. Most connection poolers allow you to specify a JDBC connection string, so your application could call into `getJdbcConnectionString(...)` and pass that to the connection pooler to make use of faster failover on Aurora PostgreSQL.

```
import java.sql.Connection;  
import java.sql.DriverManager;  
import java.sql.SQLException;  
import java.sql.Statement;  
import java.util.ArrayList;
```

```

import java.util.List;
import java.util.stream.Collectors;
import java.util.stream.IntStream;

import org.joda.time.Duration;

public class FastFailoverDriverManager {
    private static Duration LOGIN_TIMEOUT = Duration.standardSeconds(2);
    private static Duration CONNECT_TIMEOUT = Duration.standardSeconds(2);
    private static Duration CANCEL_SIGNAL_TIMEOUT = Duration.standardSeconds(1);
    private static Duration DEFAULT_SOCKET_TIMEOUT = Duration.standardSeconds(5);

    public FastFailoverDriverManager() {
        try {
            Class.forName("org.postgresql.Driver");
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        }

        /*
         * RO endpoint has a TTL of 1s, we should honor that here. Setting this
         aggressively makes sure that when
         * the PG JDBC driver creates a new connection, it will resolve a new different RO
         endpoint on subsequent attempts
         * (assuming there is > 1 read node in your cluster)
         */
        java.security.Security.setProperty("networkaddress.cache.ttl" , "1");
        // If the lookup fails, default to something like small to retry
        java.security.Security.setProperty("networkaddress.cache.negative.ttl" , "3");
    }

    public Connection getConnection(String targetServerType) throws SQLException {
        return getConnection(targetServerType, DEFAULT_SOCKET_TIMEOUT);
    }

    public Connection getConnection(String targetServerType, Duration queryTimeout) throws
    SQLException {
        Connection conn =
DriverManager.getConnection(getJdbcConnectionString(targetServerType, queryTimeout));

        /*
         * A good practice is to set socket and statement timeout to be the same thing
         since both
         * the client AND server will kill the query at the same time, leaving no running
         queries
         * on the backend
         */
        Statement st = conn.createStatement();
        st.execute("set statement_timeout to " + queryTimeout.getMillis());
        st.close();

        return conn;
    }

    private static String urlFormat = "jdbc:postgresql://%" +
        + "/postgres"
        + "?user=%s"
        + "&password=%s"
        + "&loginTimeout=%d"
        + "&connectTimeout=%d"
        + "&cancelSignalTimeout=%d"
        + "&socketTimeout=%d"
        + "&targetServerType=%s"
        + "&tcpKeepAlive=true"
        + "&ssl=true"
        + "&loadBalanceHosts=true";
}

```

```
public String getJdbcConnectionString(String targetServerType, Duration queryTimeout) {
    return String.format(urlFormat,
        getFormattedEndpointList(getLocalEndpointList()),
        CredentialManager.getUsername(),
        CredentialManager.getPassword(),
        LOGIN_TIMEOUT.getStandardSeconds(),
        CONNECT_TIMEOUT.getStandardSeconds(),
        CANCEL_SIGNAL_TIMEOUT.getStandardSeconds(),
        queryTimeout.getStandardSeconds(),
        targetServerType
    );
}

private List<String> getLocalEndpointList() {
    /*
     * As mentioned in the best practices doc, a good idea is to read a local resource
     * file and parse the cluster endpoints.
     * For illustration purposes, the endpoint list is hardcoded here
     */
    List<String> newEndpointList = new ArrayList<>();
    newEndpointList.add("myauroracluster.cluster-c9bfei4hjlr.us-east-1-
beta.rds.amazonaws.com:5432");
    newEndpointList.add("myauroracluster.cluster-ro-c9bfei4hjlr.us-east-1-
beta.rds.amazonaws.com:5432");

    return newEndpointList;
}

private static String getFormattedEndpointList(List<String> endpoints) {
    return IntStream.range(0, endpoints.size())
        .mapToObj(i -> endpoints.get(i).toString())
        .collect(Collectors.joining(","));
}
}
```

Troubleshooting storage issues

If the amount of memory required by a sort or index creation operation exceeds the amount of memory available, Aurora PostgreSQL writes the excess data to storage. When it writes the data it uses the same storage space it uses for storing error and message logs. If your sorts or index creation functions exceed memory available, you could develop a local storage shortage. If you experience issues with Aurora PostgreSQL running out of storage space, you can either reconfigure your data sorts to use more memory, or reduce the data retention period for your PostgreSQL log files. For more information about changing the log retention period see, [PostgreSQL Database Log Files \(p. 572\)](#).

Amazon Aurora PostgreSQL Reference

Amazon Aurora PostgreSQL Parameters

You manage your Amazon Aurora 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 Amazon Aurora DB cluster apply to the entire cluster, while other parameters apply only to a particular DB instance in the DB cluster.

Cluster-level parameters are managed in DB cluster parameter groups. Instance-level parameters are managed in DB parameter groups. Although each DB instance in an Aurora PostgreSQL DB cluster is compatible with the PostgreSQL database engine, some of the PostgreSQL database engine parameters

must be applied at the cluster level, and are managed using DB cluster parameter groups. Cluster-level parameters are not found in the DB parameter group for a DB instance in an Aurora PostgreSQL DB cluster and are listed later in this topic.

You can manage both cluster-level and instance-level 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. For example, you can use the [modify-db-cluster-parameter-group](#) AWS CLI command to manage cluster-level parameters in a DB cluster parameter group and use the [modify-db-parameter-group](#) AWS 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 Amazon RDS console, or by using the AWS CLI or Amazon 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 and use the [describe-db-parameters](#) AWS CLI command to view instance-level parameters in a DB parameter group for a DB instance in a DB cluster.

For more information about parameter groups, see [Working with DB Parameter Groups and DB Cluster Parameter Groups \(p. 286\)](#).

Cluster-level Parameters

The following table shows all of the parameters that apply to the entire Aurora PostgreSQL DB cluster.

Parameter name	Modifiable
ansi_constraint_trigger_ordering	Yes
ansi_force_foreign_key_checks	Yes
ansi_qualified_update_set_target	Yes
apg_ccm_enabled	Yes
archive_command	No
archive_timeout	No
array_nulls	Yes
autovacuum	Yes
autovacuum_analyze_scale_factor	Yes
autovacuum_analyze_threshold	Yes
autovacuum_freeze_max_age	Yes
autovacuum_max_workers	Yes
autovacuum_multixact_freeze_max_age	Yes
autovacuum_naptime	Yes
autovacuum_vacuum_cost_delay	Yes
autovacuum_vacuum_cost_limit	Yes
autovacuum_vacuum_scale_factor	Yes
autovacuum_vacuum_threshold	Yes

Parameter name	Modifiable
<code>autovacuum_work_mem</code>	Yes
<code>backslash_quote</code>	Yes
<code>client_encoding</code>	Yes
<code>data_directory</code>	No
<code>datestyle</code>	Yes
<code>default_tablespace</code>	Yes
<code>default_with_oids</code>	Yes
<code>extra_float_digits</code>	Yes
<code>huge_pages</code>	No
<code>intervalstyle</code>	Yes
<code>lc_monetary</code>	Yes
<code>lc_numeric</code>	Yes
<code>lc_time</code>	Yes
<code>log_autovacuum_min_duration</code>	Yes
<code>max_prepared_transactions</code>	Yes
<code>password_encryption</code>	No
<code>port</code>	No
<code>rds.enable_plan_management</code>	Yes
<code>rds.extensions</code>	No
<code>rds.force_autovacuum_logging_level</code>	Yes
<code>rds.force_ssl</code>	Yes
<code>rds.logical_replication</code>	Yes
<code>rds.restrict_password_commands</code>	Yes
<code>server_encoding</code>	No
<code>ssl</code>	Yes
<code>synchronous_commit</code>	Yes
<code>timezone</code>	Yes
<code>track_commit_timestamp</code>	Yes
<code>vacuum_cost_delay</code>	Yes
<code>vacuum_cost_limit</code>	Yes
<code>vacuum_cost_page_hit</code>	Yes

Parameter name	Modifiable
vacuum_cost_page_miss	Yes
vacuum_defer_cleanup_age	Yes
vacuum_freeze_min_age	Yes
vacuum_freeze_table_age	Yes
vacuum_multixact_freeze_min_age	Yes
vacuum_multixact_freeze_table_age	Yes
wal_buffers	Yes

Instance-level Parameters

The following table shows all of the parameters that apply to a specific DB instance in an Aurora PostgreSQL DB cluster.

Parameter name	Modifiable
apg_plan_mgmt.capture_plan_baselines	Yes
apg_plan_mgmt.max_databases	Yes
apg_plan_mgmt.max_plans	Yes
apg_plan_mgmt.pgss_min_calls	Yes
apg_plan_mgmt.pgss_min_mean_time_ms	Yes
apg_plan_mgmt.pgss_min_stddev_time_ms	Yes
apg_plan_mgmt.pgss_min_total_time_ms	Yes
apg_plan_mgmt.plan_retention_period	Yes
apg_plan_mgmt.unapproved_plan_execution_threshold	Yes
apg_plan_mgmt.use_plan_baselines	Yes
application_name	Yes
authentication_timeout	Yes
auto_explain.log_analyze	Yes
auto_explain.log_buffers	Yes
auto_explain.log_format	Yes
auto_explain.log_min_duration	Yes
auto_explain.log_nested_statements	Yes
auto_explain.log_timing	Yes
auto_explain.log_triggers	Yes

Parameter name	Modifiable
auto_explain.log_verbose	Yes
auto_explain.sample_rate	Yes
backend_flush_after	Yes
bgwriter_flush_after	Yes
bytea_output	Yes
check_function_bodies	Yes
checkpoint_flush_after	Yes
checkpoint_timeout	No
client_min_messages	Yes
config_file	No
constraint_exclusion	Yes
cpu_index_tuple_cost	Yes
cpu_operator_cost	Yes
cpu_tuple_cost	Yes
cursor_tuple_fraction	Yes
db_user_namespace	No
deadlock_timeout	Yes
debug_pretty_print	Yes
debug_print_parse	Yes
debug_print_plan	Yes
debug_print_rewritten	Yes
default_statistics_target	Yes
default_transaction_deferrable	Yes
default_transaction_isolation	Yes
default_transaction_read_only	Yes
effective_cache_size	Yes
effective_io_concurrency	Yes
enable_bitmapscan	Yes
enable_hashagg	Yes
enable_hashjoin	Yes
enable_indexonlyscan	Yes

Parameter name	Modifiable
enable_indexscan	Yes
enable_material	Yes
enable_mergejoin	Yes
enable_nestloop	Yes
enable_seqscan	Yes
enable_sort	Yes
enable_tidscan	Yes
escape_string_warning	Yes
exit_on_error	No
force_parallel_mode	Yes
fromCollapse_limit	Yes
geqo	Yes
geqo_effort	Yes
geqo_generations	Yes
geqo_pool_size	Yes
geqo_seed	Yes
geqo_selection_bias	Yes
geqo_threshold	Yes
gin_fuzzy_search_limit	Yes
gin_pending_list_limit	Yes
hba_file	No
hot_standby_feedback	No
ident_file	No
idle_in_transaction_session_timeout	Yes
joinCollapse_limit	Yes
lc_messages	Yes
listen_addresses	No
lo_compat_privileges	No
log_connections	Yes
log_destination	Yes
log_directory	No

Parameter name	Modifiable
log_disconnections	Yes
log_duration	Yes
log_error_verbosity	Yes
log_executor_stats	Yes
log_file_mode	No
log_filename	Yes
log_hostname	Yes
log_line_prefix	No
log_lock_waits	Yes
log_min_duration_statement	Yes
log_min_error_statement	Yes
log_min_messages	Yes
log_parser_stats	Yes
log_planner_stats	Yes
log_replication_commands	Yes
log_rotation_age	Yes
log_rotation_size	Yes
log_statement	Yes
log_statement_stats	Yes
log_temp_files	Yes
log_timezone	No
log_truncate_on_rotation	No
logging_collector	No
maintenance_work_mem	Yes
max_connections	Yes
max_files_per_process	Yes
max_locks_per_transaction	Yes
max_replication_slots	Yes
max_stack_depth	Yes
max_standby_archive_delay	No
max_standby_streaming_delay	No

Parameter name	Modifiable
max_wal_senders	Yes
max_worker_processes	Yes
min_parallel_relation_size	Yes
old_snapshot_threshold	Yes
operator_precedence_warning	Yes
parallel_setup_cost	Yes
parallel_tuple_cost	Yes
pg_hint_plan.debug_print	Yes
pg_hint_plan.enable_hint	Yes
pg_hint_plan.enable_hint_table	Yes
pg_hint_plan.message_level	Yes
pg_hint_plan.parse_messages	Yes
pg_stat_statements.max	Yes
pg_stat_statements.save	Yes
pg_stat_statements.track	Yes
pg_stat_statements.track_utility	Yes
pgaudit.log	Yes
pgaudit.log_catalog	Yes
pgaudit.log_level	Yes
pgaudit.log_parameter	Yes
pgaudit.log_relation	Yes
pgaudit.log_statement_once	Yes
pgaudit.role	Yes
postgis.gdal_enabled_drivers	Yes
quote_all_identifiers	Yes
random_page_cost	Yes
rds.force_admin_logging_level	Yes
rds.log_retention_period	Yes
rds.rds_superuser_reserved_connections	Yes
rds.superuser_variables	No
replacement_sort_tuples	Yes

Parameter name	Modifiable
<code>restart_after_crash</code>	No
<code>row_security</code>	Yes
<code>search_path</code>	Yes
<code>seq_page_cost</code>	Yes
<code>session_replication_role</code>	Yes
<code>shared_buffers</code>	Yes
<code>shared_preload_libraries</code>	Yes
<code>sql_inheritance</code>	Yes
<code>ssl_ca_file</code>	No
<code>ssl_cert_file</code>	No
<code>ssl_ciphers</code>	No
<code>ssl_key_file</code>	No
<code>standard_conforming_strings</code>	Yes
<code>statement_timeout</code>	Yes
<code>stats_temp_directory</code>	No
<code>superuser_reserved_connections</code>	No
<code>synchronize_seqscans</code>	Yes
<code>syslog_facility</code>	No
<code>tcp_keepalives_count</code>	Yes
<code>tcp_keepalives_idle</code>	Yes
<code>tcp_keepalives_interval</code>	Yes
<code>temp_buffers</code>	Yes
<code>temp_tablespaces</code>	Yes
<code>track_activities</code>	Yes
<code>track_activity_query_size</code>	Yes
<code>track_counts</code>	Yes
<code>track_functions</code>	Yes
<code>track_io_timing</code>	Yes
<code>transaction_deferrable</code>	Yes
<code>transaction_read_only</code>	Yes
<code>transform_null_equals</code>	Yes

Parameter name	Modifiable
unix_socket_directories	No
unix_socket_group	No
unix_socket_permissions	No
update_process_title	Yes
wal_receiver_status_interval	Yes
wal_receiver_timeout	Yes
wal_sender_timeout	Yes
work_mem	Yes
xmlbinary	Yes
xmloption	Yes

Amazon Aurora PostgreSQL Events

The following are some common wait events for Aurora PostgreSQL.

BufferPin:BufferPin

In this wait event, a session is waiting to access a data buffer during a period when no other session can examine that buffer. Buffer pin waits can be protracted if another process holds an open cursor which last read data from the buffer in question.

Client:ClientRead

In this wait event, a session is receiving data from an application client. This wait might be prevalent during bulk data loads using the COPY statement, or for applications that pass data to Aurora using many round trips between the client and the database. A high number of client read waits per transaction may indicate excessive round trips, such as parameter passing. You should compare this against the expected number of statements per transaction.

IO:DataFilePrefetch

In this wait event, a session is waiting for an asynchronous prefetch from Aurora Storage.

IO:DataFileRead

In this wait event, a session is reading data from Aurora Storage. This may be a typical wait event for I/O intensive workloads. SQL statements showing a comparatively large proportion of this wait event compared to other SQL statements may be using an inefficient query plan that requires reading large amounts of data.

IO:XactSync

In this wait event, a session is issuing a COMMIT or ROLLBACK, requiring the current transaction's changes to be persisted. Aurora is waiting for Aurora storage to acknowledge persistence.

This wait most often arises when there is a very high rate of commit activity on the system. You can sometimes alleviate this wait by modifying applications to commit transactions in batches. You might see this wait at the same time as CPU waits in a case where the DB load exceeds the number of virtual CPUs (vCPUs) for the DB instance. In this case, the storage persistence might be competing for CPU with CPU-intensive database workloads. To alleviate this scenario, you can try reducing those workloads, or scaling up to a DB instance with more vCPUs.

Lock:transactionid

In this wait event, a session is trying to modify data that has been modified by another session, and is waiting for the other session's transaction to be committed or rolled back. You can investigate blocking and waiting sessions in the pg_locks view.

LWLock:buffer_content

In this wait event, a session is waiting to read or write a data page in memory while another session has that page locked for writing. Heavy write contention for a single page (hot page), due to frequent updates of the same piece of data by many sessions, could lead to prevalence of this wait event. Excessive use of foreign key constraints could increase lock duration, leading to increased contention. You should investigate workloads experiencing high buffer_content waits for usage of foreign key constraints to determine if the constraints are necessary. Alternatively, decreasing the fillfactor on the parent table will spread the keys across more of the block and can reduce contention on the page.

LWLock:SubtransControlLock

In this wait event, a session is looking up or manipulating the parent/child relationship between a transaction and a subtransaction. The two most common causes of subtransaction use are savepoints and PL/pgSQL exception blocks. The wait event might occur if there is heavy concurrent querying of data that's simultaneously being updated from within subtransactions. You should investigate whether it is possible to reduce the use of savepoints and exception blocks, or to decrease concurrent queries on the rows being updated.

For a complete list of PostgreSQL wait events, see [PostgreSQL wait-event table](#).

Database Engine Updates for Amazon Aurora PostgreSQL

In the following topic, you can find version and update information specific to Amazon Aurora with PostgreSQL compatibility. For more information about updates that apply generally to Aurora, see [Amazon Aurora Updates \(p. 432\)](#).

Topics

- [Identifying Your Version of Amazon Aurora PostgreSQL \(p. 970\)](#)
- [Upgrading the Amazon Aurora PostgreSQL Engine Version \(p. 971\)](#)
- [Database Engine Versions for Amazon Aurora PostgreSQL \(p. 971\)](#)

Identifying Your Version 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 has two version numbers; the Aurora version number and the database engine version number.

- To get the Aurora version number, see [Identifying Your Amazon Aurora Version \(p. 432\)](#).

- You can get the database engine version number for an Aurora PostgreSQL DB instance by querying for the SERVER_VERSION runtime parameter. To get the database engine version number, use the following query.

```
SHOW SERVER_VERSION;
```

Upgrading the Amazon Aurora PostgreSQL Engine Version

For more information about upgrading the Amazon Aurora PostgreSQL engine version, see [Upgrading an Aurora PostgreSQL DB Cluster Engine Version \(p. 950\)](#).

Database Engine Versions for Amazon Aurora PostgreSQL

Following, you can find information about supported versions of the Aurora with PostgreSQL compatibility database engine. An Aurora database has two version numbers; the Aurora version number and the database engine version number. To determine the version numbers of your Aurora PostgreSQL database, see [Identifying Your Version of Amazon Aurora PostgreSQL \(p. 970\)](#).

The following table shows the version of each Aurora PostgreSQL release and the PostgreSQL version that it is compatible with.

Aurora with PostgreSQL Compatibility	Compatible PostgreSQL Release
Version 3.0 (p. 971)	11.4
Version 2.3 (p. 973)	10.7
Version 2.2 (p. 974)	10.6
Version 2.1 (p. 975)	10.5
Version 2.0 (p. 977)	10.4
Version 1.5 (p. 978)	9.6.12
Version 1.4 (p. 979)	9.6.11
Version 1.3 (p. 981)	9.6.9
Version 1.2 (p. 982)	9.6.8
Version 1.1 (p. 984)	9.6.6 deprecated
Version 1.0 (p. 985)	9.6.3 deprecated

The following Aurora PostgreSQL versions are supported.

Version 3.0

This version of Aurora PostgreSQL is compatible with PostgreSQL 11.4. For more information about the improvements in release 11.4, see [PostgreSQL Release 11.4](#).

Note

For the initial release, the supported AWS Regions are us-east-1, us-east-2, us-west-2, eu-west-1, ap-northeast-1, and ap-northeast-2. For the complete list of AWS Regions, see [Aurora PostgreSQL Region Availability \(p. 82\)](#).

You can find the following improvements in this engine update.

Improvements

1. This release contains all fixes, features, and improvements present in [Version 2.3.5 \(p. 973\)](#).
2. Partitioning – Partitioning improvements include support for hash partitioning, enabling creation of a default partition, and dynamic row movement to another partition based on the key column update.
3. Performance – Performance improvements include parallelism while creating indexes, materialized views, hash joins, and sequential scans to make the operations perform better.
4. Stored procedures – SQL stored procedures now added support for embedded transactions.
5. Support for Just-In-Time (JIT) capability – RDS PostgreSQL 11 instances are created with JIT capability, speeding evaluation of expressions. To enable this feature, set `jit` to ON.
6. Autovacuum improvements – To provide valuable logging, the parameter `rds.force_autovacuum_logging` is ON by default in conjunction with the `log_autovacuum_min_duration` parameter set to 10 seconds. To increase autovacuum effectiveness, the values for the `autovacuum_max_workers` and `autovacuum_vacuum_cost_limit` parameters are computed based on host memory capacity to provide larger default values.
7. Improved transaction timeout – The parameter `idle_in_transaction_session_timeout` is set to 12 hours. Any session that has been idle more than 12 hours is terminated.
8. The `tsearch2` module is no longer supported – If your application uses `tsearch2` functions, update it to use the equivalent functions provided by the core PostgreSQL engine. For more information about the `tsearch2` module, see [PostgreSQL tsearch2](#).
9. The `chkpass` module is no longer supported – For more information about the `chkpass` module, see [PostgreSQL chkpass](#).

10 Updated the following extensions:

- `address_standardizer` to version 2.5.1
- `address_standardizer_data_us` to version 2.5.1
- `btree_gin` to version 1.3
- `citext` to version 1.5
- `cube` to version 1.4
- `hstore` to version 1.5
- `ip4r` to version 2.2
- `isn` to version 1.2
- `orafce` to version 3.7
- `pg_hint_plan` to version 1.3.4
- `pg_prewarm` to version 1.2
- `pg_repack` to version 1.4.4
- `pg_trgm` to version 1.4
- `pgaudit` to version 1.3
- `pgrouting` to version 2.6.1
- `pgtap` to version 1.0.0
- `plcoffee` to version 2.3.8
- `plls` to version 2.3.8
- `plv8` to version 2.3.8

- `postgis` to version 2.5.1
- `postgis_tiger_geocoder` to version 2.5.1
- `postgis_topology` to version 2.5.1
- `rds_activity_stream` to version 1.3

Version 2.3

This version of Aurora PostgreSQL is compatible with PostgreSQL 10.7. For more information about the improvements in release 10.7, see [PostgreSQL Release 10.7](#).

Patch Versions

- [Version 2.3.5 \(p. 973\)](#)
- [Version 2.3.3 \(p. 973\)](#)
- [Version 2.3.1 \(p. 974\)](#)
- [Version 2.3.0 \(p. 974\)](#)

Version 2.3.5

You can find the following improvements in this engine update.

Improvements

1. Fixed a bug that could cause DB instance restarts.
2. Fixed a bug that could cause a crash when a backend exits while using logical replication.
3. Fixed a bug that could cause a restart when reads occurred during failovers.
4. Fixed a bug with the `wal2json` plugin for logical replication.
5. Fixed a bug that could result in inconsistent metadata.

Version 2.3.3

You can find the following improvements in this engine update.

Improvements

1. Provided a backport fix for the PostgreSQL community security issue CVE-2019-10130.
2. Provided a backport fix for the PostgreSQL community security issue CVE-2019-10164.
3. Fixed a bug whereby data activity streaming could consume excessive CPU time.
4. Fixed a bug whereby parallel threads scanning a B-tree index could hang following a disk read.
5. Fixed a bug where use of the `not in` predicate against a common table expression (CTE) could return the following error: "ERROR: bad levelsup for CTE".
6. Fixed a bug whereby the read node replay process could hang while applying a modification to a generalized search tree (GiST) index.
7. Fixed a bug whereby visibility map pages could contain incorrect freeze bits following a failover to a read node.
8. Optimized log traffic between the write node and read nodes during index maintenance.
9. Fixed a bug whereby queries on read nodes may crash while performing a B-tree index scan.
10. Fixed a bug whereby a query that has been optimized for redundant inner join removal could crash.
11. The function `aurora_stat_memctx_usage` now reports the number of instances of a given context name.

- 12Fixed a bug whereby the function `aurora_stat_memctx_usage` reported incorrect results.
- 13Fixed a bug whereby the read node replay process could wait to kill conflicting queries beyond the configured `max_standby_streaming_delay` value.
- 14Additional information is now logged on read nodes when active connections conflict with the relay process.
- 15Provided a backport fix for the PostgreSQL community bug #15677, where a crash could occur while deleting from a partitioned table.

Version 2.3.1

You can find the following improvements in this engine update.

Improvements

1. Fixed multiple bugs related to I/O prefetching that caused engine crashes.

Version 2.3.0

You can find the following improvements in this engine update.

New Features

1. Aurora PostgreSQL now performs I/O prefetching while scanning B-tree indexes. This results in significantly improved performance for B-tree scans over uncached data.

Improvements

1. Fixed a bug whereby read nodes may fail with the error "too many LWLocks taken".
2. Addressed numerous issues that caused read nodes to fail to startup while the cluster is under heavy write workload.
3. Fixed a bug whereby usage of the `aurora_stat_memctx_usage()` function could lead to a crash.
4. Improved the cache replacement strategy used by table scans to minimize thrashing of the buffer cache.

Version 2.2

This version of Aurora PostgreSQL is compatible with PostgreSQL 10.6. For more information about the improvements in release 10.6, see [PostgreSQL Release 10.6](#).

Patch Versions

- [Version 2.2.1 \(p. 974\)](#)
- [Version 2.2.0 \(p. 975\)](#)

Version 2.2.1

You can find the following improvements in this engine update.

Improvements

1. Improved stability of logical replication.
2. Fixed a bug which could cause an error running queries. The message reported would be of the form "CLOG segment 123 does not exist: No such file or directory".

3. Increased the supported size of IAM passwords to 8KB.
4. Improved consistency of performance under high throughput write workloads.
5. Fixed a bug which could cause a read replica to crash during a restart.
6. Fixed a bug which could cause an error running queries. The message reported would be of the form "SQL ERROR: Attempting to read past EOF of relation".
7. Fixed a bug which could cause an increase in memory usage after a restart.
8. Fixed a bug which could cause a transaction with a large number of subtransactions to fail.
9. Merged a patch from community PostgreSQL which addresses potential failures when using GIN indexes. For more information see <https://git.postgresql.org/gitweb/?p=postgresql.git;a=commit;h=f9e66f2fb9a493045c8d8086a9b15d95b8f18>.
10. Fixed a bug which could cause a snapshot import from RDS for PostgreSQL to fail.

Version 2.2.0

You can find the following improvements in this engine update.

New features

1. Added the restricted password management feature. Restricted password management enables you to restrict who can manage user passwords and password expiration changes by using the parameter `rds.restrict_password_commands` and the role `rds_password`. For more information, see [Restricting Password Management \(p. 880\)](#).

Version 2.1

This version of Aurora PostgreSQL is compatible with PostgreSQL 10.5. For more information about the improvements in release 10.5, see [PostgreSQL Release 10.5](#).

Patch Versions

- [Version 2.1.1 \(p. 975\)](#)
- [Version 2.1.0 \(p. 976\)](#)

Version 2.1.1

You can find the following improvements in this engine update.

Improvements

1. Fixed a bug which could cause an error running queries. The message reported would be of the form "CLOG segment 123 does not exist: No such file or directory".
2. Increased the supported size of IAM passwords to 8KB.
3. Improved consistency of performance under high throughput write workloads.
4. Fixed a bug which could cause a read replica to crash during a restart.
5. Fixed a bug which could cause an error running queries. The message reported would be of the form "SQL ERROR: Attempting to read past EOF of relation".
6. Fixed a bug which could cause an increase in memory usage after a restart.
7. Fixed a bug which could cause a transaction with a large number of subtransactions to fail.
8. Merged a patch from community PostgreSQL which addresses potential failures when using GIN indexes. For more information see <https://git.postgresql.org/gitweb/?p=postgresql.git;a=commit;h=f9e66f2fb9a493045c8d8086a9b15d95b8f18>.
9. Fixed a bug which could cause a snapshot import from RDS for PostgreSQL to fail.

Version 2.1.0

You can find the following improvements in this engine update.

New features

1. General availability of Aurora Query Plan Management, which enables customers to track and manage any or all query plans used by their applications, to control query optimizer plan selection, and to ensure high and stable application performance. For more information, see [Managing Query Execution Plans for Aurora PostgreSQL \(p. 916\)](#).
2. Updated the `libprotobuf` extension to version 1.3.0. This is used by the PostGIS extension.
3. Updated the `pg_similarity` extension to version 1.0.
4. Updated the `log_fdw` extension to version 1.1.
5. Updated the `pg_hint_plan` extension to version 1.3.1.

Improvements

1. Network traffic between the writer and reader nodes is now compressed to reduce network utilization. This reduces the chance of read node unavailability due to network saturation.
2. Implemented a high performance, scalable subsystem for PostgreSQL subtransactions. This improves the performance of applications which make extensive use of savepoints and PL/pgSQL exception handlers.
3. The `rds_superuser` role can now set the following parameters on a per-session, database, or role level:
 - `log_duration`
 - `log_error_verbosity`
 - `log_executor_stats`
 - `log_lock_waits`
 - `log_min_duration_statement`
 - `log_min_error_statement`
 - `log_min_messages`
 - `log_parser_stats`
 - `log_planner_stats`
 - `log_replication_commands`
 - `log_statement_stats`
 - `log_temp_files`
4. Fixed a bug whereby the SQL command "ALTER FUNCTION ... OWNER TO ..." might fail with error "improper qualified name (too many dotted names)".
5. Fixed a bug whereby a crash could occur while committing a transaction with more than two million subtransactions.
6. Fixed a bug in community PostgreSQL code related to GIN indexes which can cause the Aurora Storage volume to become unavailable.
7. Fixed a bug whereby an Aurora PostgreSQL replica of an RDS for PostgreSQL instance might fail to start, reporting error: "PANIC: could not locate a valid checkpoint record".
8. Fixed a bug whereby passing an invalid parameter to the `aurora_stat_backend_waits` function could cause a crash.

Known issues

1. The `pageinspect` extension is not supported in Aurora PostgreSQL.

Version 2.0

This version of Aurora PostgreSQL is compatible with PostgreSQL 10.4. For more information about the improvements in release 10.4, see [PostgreSQL Release 10.4](#).

Patch Versions

- [Version 2.0.1 \(p. 977\)](#)
- [Version 2.0.0 \(p. 977\)](#)

Version 2.0.1

You can find the following improvements in this engine update.

Improvements

1. Fixed a bug which could cause an error running queries. The message reported would be of the form "CLOG segment 123 does not exist: No such file or directory".
2. Increased the supported size of IAM passwords to 8KB.
3. Improved consistency of performance under high throughput write workloads.
4. Fixed a bug which could cause a read replica to crash during a restart.
5. Fixed a bug which could cause an error running queries. The message reported would be of the form "SQL ERROR: Attempting to read past EOF of relation".
6. Fixed a bug which could cause an increase in memory usage after a restart.
7. Fixed a bug which could cause a transaction with a large number of subtransactions to fail.
8. Merged a patch from community PostgreSQL which addresses potential failures when using GIN indexes. For more information see <https://git.postgresql.org/gitweb/?p=postgresql.git;a=commit;h=f9e66f2fbbb49a493045c8d8086a9b15d95b8f18>.
9. Fixed a bug which could cause a snapshot import from RDS for PostgreSQL to fail.

Version 2.0.0

You can find the following improvements in this engine update.

Improvements

1. This release contains all fixes, features, and improvements present in [Version 1.3 \(p. 981\)](#).
2. The temporary file size limitation is user-configurable. You require the **rds_superuser** role to modify the **temp_file_limit** parameter.
3. Updated the **GDAL** library, which is used by the **PostGIS** extension.
4. Updated the **ip4r** extension to version 2.1.1.
5. Updated the **pg_repack** extension to version 1.4.3.
6. Updated the **plv8** extension to version 2.1.2.
7. Parallel queries – When you create a new Aurora PostgreSQL version 2.0 instance, parallel queries are enabled for the **default.postgres10** parameter group. The parameter **max_parallel_workers_per_gather** is set to 2 by default, but you can modify it to support your specific workload requirements.
8. Fixed a bug whereby read nodes may crash following a specific type of free space change from the write node.

Version 1.5

This version of Aurora PostgreSQL is compatible with PostgreSQL 9.6.12. For more information about the improvements in release 9.6.12, see [PostgreSQL Release 9.6.12](#).

Patch Versions

- [Version 1.5.3 \(p. 978\)](#)
- [Version 1.5.2 \(p. 978\)](#)
- [Version 1.5.1 \(p. 978\)](#)
- [Version 1.5.0 \(p. 979\)](#)

Version 1.5.3

You can find the following improvements in this engine update.

Improvements

1. Fixed a bug that could cause DB instance restarts.
2. Fixed a bug that could cause a restart when reads occurred during failovers.
3. Fixed a bug that could result in inconsistent metadata.

Version 1.5.2

You can find the following improvements in this engine update.

Improvements

1. Provided a backport fix for the PostgreSQL community security issue CVE-2019-10130.
 2. Fixed a bug whereby the read node replay process may hang while applying a modification to a generalized search tree (GiST) index.
 3. Fixed a bug whereby visibility map pages may contain incorrect freeze bits following a failover to a read node.
 4. Fixed a bug whereby the error "relation `relation-name` does not exist" is incorrectly reported.
 5. Optimized log traffic between the write node and read nodes during index maintenance.
 6. Fixed a bug whereby queries on read nodes may crash while performing a B-tree index scan.
 7. The function `aurora_stat_memctx_usage` now reports the number of instances of a given context name.
 8. Fixed a bug whereby the function `aurora_stat_memctx_usage` reported incorrect results.
 9. Fixed a bug whereby the read node replay process may wait to kill conflicting queries beyond the configured `max_standby_streaming_delay`.
- 10Additional information is now logged on read nodes when active connections conflict with the relay process.

Version 1.5.1

You can find the following improvements in this engine update.

Improvements

1. Fixed multiple bugs related to I/O prefetching, which caused engine crashes.

Version 1.5.0

You can find the following improvements in this engine update.

New features

1. Aurora PostgreSQL now performs I/O prefetching while scanning B-tree indexes. This results in significantly improved performance for B-tree scans over uncached data.

Improvements

1. Addressed numerous issues that caused read nodes to fail to startup while the cluster is under heavy write workload.
2. Fixed a bug whereby usage of the `aurora_stat_memctx_usage()` function could lead to a crash.
3. Improved the cache replacement strategy used by table scans to minimize thrashing of the buffer cache.

Version 1.4

This version of Aurora PostgreSQL is compatible with PostgreSQL 9.6.11. For more information about the improvements in release 9.6.11, see [PostgreSQL Release 9.6.11](#).

You can find the following improvements in this engine update.

New features

1. Support is added for the `pg_similarity` extension version 1.0.

Improvements

1. This release contains all fixes, features, and improvements present in [Version 1.3 \(p. 981\)](#).
2. Network traffic between the writer and reader nodes is now compressed to reduce network utilization. This reduces the chance of read node unavailability due to network saturation.
3. Performance of subtransactions has improved under high concurrency workloads.
4. An update for the `pg_hint_plan` extension to version 1.2.3.
5. Fixed an issue where on a busy system, a commit with millions of subtransactions (and sometimes with commit timestamps enabled) can cause Aurora to crash.
6. Fixed an issue where an `INSERT` statement with `VALUES` could fail with the message "Attempting to read past EOF of relation".
7. An upgrade of the `apg_plan_mgmt` extension to version 1.0.1. The `apg_plan_mgmt` extension is used with query plan management. For more about how to install, upgrade, and use the `apg_plan_mgmt` extension, see [Managing Query Execution Plans for Aurora PostgreSQL \(p. 916\)](#).

The `apg_plan_mgmt` extension new features include the following:

- a. A new `update_plan_hash` parameter is available for the `validate_plans` function. This parameter updates the `plan_hash` for plans that can't be reproduced exactly. The `update_plan_hash` parameter also enables you to fix a plan by rewriting the SQL. You can then register the good plan as an Approved plan for the original SQL. Following is an example of using `update_plan_hash`.

```
UPDATE apg_plan_mgmt.dba_plans SET plan_hash = new_plan_hash, plan_outline
= good_plan_outline
WHERE sql_hash = bad_plan_sql_hash AND plan_hash = bad_plan_plan_hash;
```

```
SELECT apg_plan_mgmt.validate_plans(bad_plan_sql_hash, bad_plan_plan_hash,  
    'update_plan_hash');  
SELECT apg_plan_mgmt.reload();
```

- b. A new `get_explain_stmt` function is available that generates the text of an `EXPLAIN` statement for the specified SQL statement. It includes the parameters `sql_hash`, `plan_hash` and `explain_options`.

The parameter `explain_options` can be any comma-separated list of valid `EXPLAIN` options, as shown following.

```
analyze,verbose,buffers,hashes,format json
```

If the parameter `explain_options` is `NULL` or an empty string, the `get_explain_stmt` function generates a simple `EXPLAIN` statement.

To create an `EXPLAIN` script for your workload or a portion of it, use the `\a`, `\t`, and `\o` options to redirect the output to a file. For example, you can create an `EXPLAIN` script for the top-ranked (top-K) statements by using the PostgreSQL `pg_stat_statements` view sorted by `total_time` in `DESC` order.

- c. The precise location of the Gather parallel query operator is determined by costing, and may change slightly over time. To prevent these differences from invalidating the entire plan, query plan management now computes the same `plan_hash` even if the Gather operators move to different places in the plan tree.
- d. Support is added for nonparameterized statements inside pl/pgsql functions.
- e. Overhead is reduced when the `apg_plan_mgmt` extension is installed on multiple databases in the same cluster while two or more databases are being accessed concurrently. Also, this release fixed a bug in this area that caused plans to not be stored in shared memory.

The `apg_plan_mgmt` extension improvements include the following:

- a. Improvements to the `evolve_plan_baselines` function.
 - i. The `evolve_plan_baselines` function now computes a `cardinality_error` metric over all nodes in the plan. Using this metric, you can identify any plan where the cardinality estimation error is large, and the plan quality is more doubtful. Long-running statements with high `cardinality_error` values are high-priority candidates for query tuning.
 - ii. Reports generated by `evolve_plan_baselines` now include `sql_hash`, `plan_hash`, and the `plan status`.
 - iii. You can now allow `evolve_plan_baselines` to approve previously `Rejected` plans.
 - iv. The meaning of `speedup_factor` for `evolve_plan_baselines` is now always relative to the baseline plan. For example, a value of 1.1 now means 10 percent faster than the baseline plan. A value of 0.9 means 10 percent slower than the baseline plan. The comparison is made using execution time alone instead of total time.
 - v. The `evolve_plan_baselines` function now warms the cache in a new way. It does this by running the baseline plan, then running the baseline plan one more time, and then running the candidate plan once. Previously, `evolve_plan_baselines` ran the candidate plan twice. This approach added significantly to execution time, especially for slow candidate plans. However, running the candidate plan twice is more reliable when the candidate plan uses an index that isn't used in the baseline plan.
- b. Query plan management no longer saves plans that refer to system tables or views, temporary tables, or the query plan management's own tables.
- c. Bug fixes include caching a plan immediately when saved and fixing a bug that caused the back end to terminate.

Version 1.3

This version of Aurora PostgreSQL is compatible with PostgreSQL 9.6.9. For more information about the improvements in version 9.6.9, see [PostgreSQL Release 9.6.9](#).

Patch Versions

- [Version 1.3.2 \(p. 981\)](#)
- [Version 1.3.0 \(p. 981\)](#)

Version 1.3.2

You can find the following improvements in this engine update.

New features

1. Added the `ProcArrayGroupUpdate` wait event.

Improvements

1. Fixed a bug which could cause an error running queries. The message reported would be of the form "CLOG segment 123 does not exist: No such file or directory".
2. Increased the supported size of IAM passwords to 8KB.
3. Improved consistency of performance under high throughput write workloads.
4. Fixed a bug which could cause a read replica to crash during a restart.
5. Fixed a bug which could cause an error running queries. The message reported would be of the form "SQL ERROR: Attempting to read past EOF of relation".
6. Fixed a bug which could cause an increase in memory usage after a restart.
7. Fixed a bug which could cause a transaction with a large number of subtransactions to fail.
8. Merged a patch from community PostgreSQL which addresses potential failures when using GIN indexes. For more information see <https://git.postgresql.org/gitweb/?p=postgresql.git;a=commit;h=f9e66f2fbbb49a493045c8d8086a9b15d95b8f18>.
9. Fixed a bug which could cause a snapshot import from RDS for PostgreSQL to fail.

Version 1.3.0

You can find the following improvements in this engine update.

Improvements

1. This release contains all fixes, features, and improvements present in [Version 1.2 \(p. 982\)](#).
2. Updated the GDAL library, which is used by the PostGIS extension.
3. Updated the following PostgreSQL extensions:
 - `ip4r` updated to version 2.1.1.
 - `pgaudit` updated to version 1.1.1.
 - `pg_repack` updated to version 1.4.3.
 - `plv8` updated to version 2.1.2.
4. Fixed an issue in the monitoring system that could incorrectly cause a failover when local disk usage is high.
5. Fixed a bug whereby Aurora PostgreSQL can repeatedly crash, reporting:

PANIC: new_record_total_len (8201) must be less than BLCKSZ (8192), rmid (6),
info (32)

6. Fixed a bug whereby an Aurora PostgreSQL read node might be unable to rejoin a cluster due to recovery of a large buffer cache. This issue is unlikely to occur on instances other than **r4.16xlarge**.
7. Fixed a bug whereby inserting into an empty GIN index leaf page imported from pre-9.4 engine versions can cause the Aurora storage volume to become unavailable.
8. Fixed a bug whereby, in rare circumstances, a crash during transaction commit could result in the loss of CommitTs data for the committing transaction. The actual durability of the transaction was not impacted by this bug.
9. Fixed a bug in the PostGIS extension whereby PostGIS can crash in the function gserialized_gist_picksplit_2d().
10. Improved the stability of read-only nodes during heavy write traffic on instances smaller than **r4.8xl**. This specifically addresses a situation where the network bandwidth between the writer and the reader is constrained.
11. Fixed a bug whereby an Aurora PostgreSQL instance acting as a replication target of an RDS for PostgreSQL instance crashed with the following error:

FATAL: could not open file "base/16411/680897_vm": No such file or directory
during "xlog redo at 782/3122D540 for Storage/TRUNCATE"

12. Fixed a memory leak on read-only nodes whereby the heap size for the "aurora wal replay process" will continue to grow. This is observable via Enhanced Monitoring.
13. Fixed a bug whereby Aurora PostgreSQL can fail to start, with the following message reported in the PostgreSQL log:

FATAL: Storage initialization failed.

14. Fixed a performance limitation on heavy write workloads that caused waits on the LWLock:buffer_content and IO:ControlFileSyncUpdate events.

15. Fixed a bug whereby read nodes could crash following a specific type of free space change from the write node.

Version 1.2

This version of Aurora PostgreSQL is compatible with PostgreSQL 9.6.8. For more information about the improvements in version 9.6.8, see [PostgreSQL Release 9.6.8](#).

Patch Versions

- [Version 1.2.2 \(p. 982\)](#)
- [Version 1.2.0 \(p. 983\)](#)

Version 1.2.2

You can find the following improvements in this engine update.

New features

1. Added the ProcArrayGroupUpdate wait event.

Improvements

1. Fixed a bug which could cause an error running queries. The message reported would be of the form "CLOG segment 123 does not exist: No such file or directory".

2. Increased the supported size of IAM passwords to 8KB.
3. Improved consistency of performance under high throughput write workloads.
4. Fixed a bug which could cause a read replica to crash during a restart.
5. Fixed a bug which could cause an error running queries. The message reported would be of the form "SQL ERROR: Attempting to read past EOF of relation".
6. Fixed a bug which could cause an increase in memory usage after a restart.
7. Fixed a bug which could cause a transaction with a large number of subtransactions to fail.
8. Merged a patch from community PostgreSQL which addresses potential failures when using GIN indexes. For more information see <https://git.postgresql.org/gitweb/?p=postgresql.git;a=commit;h=f9e66f2fb9b49a493045c8d8086a9b15d95b8f18>.
9. Fixed a bug which could cause a snapshot import from RDS for PostgreSQL to fail.

Version 1.2.0

You can find the following improvements in this engine update.

New features

1. Introduced the `aurora_stat_memctx_usage()` function. This function reports internal memory context usage for each PostgreSQL backend. You can use this function to help determine why certain backends are consuming large amounts of memory.

Improvements

1. This release contains all fixes, features, and improvements present in [Version 1.1 \(p. 984\)](#).
2. Updates the following PostgreSQL extensions:
 - `pg_hint_plan` updated to version 1.2.2
 - `plv8` updated to version 2.1.0
3. Improves efficiency of traffic between writer and reader nodes.
4. Improves connection establishment performance.
5. Improve the diagnostic data provided in the PostgreSQL error log when an out-of-memory error is encountered.
6. Multiple fixes to improve the reliability and performance of snapshot import from Amazon RDS for PostgreSQL to Aurora with PostgreSQL compatibility.
7. Multiple fixes to improve the reliability and performance of Aurora PostgreSQL read nodes.
8. Fixes a bug in which an otherwise idle instance can generate unnecessary read traffic on an Aurora storage volume.
9. Fixes a bug in which duplicate sequence values can be encountered during insert. The problem only occurs when migrating a snapshot from RDS for PostgreSQL to Aurora PostgreSQL. The fix prevents the problem from being introduced when performing the migration. Instances migrated before this release can still encounter duplicate key errors.
10Fixes a bug in which an RDS for PostgreSQL instance migrated to Aurora PostgreSQL using replication can run out of memory doing insert/update of GIST indexes, or cause other issues with GIST indexes.
11Fixes a bug in which vacuum can fail to update the corresponding `pg_database.datfrozenxid` value for a database.
12Fixes a bug in which a crash while creating a new MultiXact (contended row level lock) can cause Aurora PostgreSQL to hang indefinitely on the first access to the same relation after the engine restarts.
13Fixes a bug in which a PostgreSQL backend can't be terminated or canceled while invoking an `fdw` call.

- 14Fixes a bug in which one vCPU is fully utilized at all times by the Aurora storage daemon. This issue is especially noticeable on smaller instance classes, such as r4.large, where it can lead to 25–50 percent CPU usage when idle.
- 15Fixes a bug in which an Aurora PostgreSQL writer node can fail over spuriously.
- 16Fixes a bug in which, in a rare scenario, an Aurora PostgreSQL read node can report:
"FATAL: lock buffer_io is not held"
- 17Fixes a bug in which stale relcache entries can halt vacuuming of relations and push the system close to transaction ID wraparound. The fix is a port of a PostgreSQL community patch scheduled to be released in a future minor version.
- 18Fixes a bug in which a failure while extending a relation can cause Aurora to crash while scanning the partially extended relation.

Version 1.1

This version of Aurora PostgreSQL is compatible with PostgreSQL 9.6.6. For more information about the improvements in version 9.6.6 see, [PostgreSQL Release 9.6.6](#).

You can find the following improvements in this engine update:

New features

1. Introduced the `aurora_stat_utils` extension. This extension includes two functions:
 - `aurora_wait_report()` function for wait event monitoring
 - `aurora_log_report()` for log record write monitoring
2. Added support for the following extensions:
 - `orafce` 3.6.1
 - `pgrouting` 2.4.2
 - `postgresql-hll` 2.10.2
 - `prefix` 1.2.6

Improvements

1. This release contains all fixes, features, and improvements present in [Version 1.0.11 \(p. 985\)](#)
2. Updates for the following PostgreSQL extensions:
 - `postgis` extension updated to version 2.3.4
 - `geos` library updated to version 3.6.2
 - `pg_repack` updated to version 1.4.2
3. Access to the `pg_statistic` relation enabled.
4. Disabled the '`effective_ioConcurrency`' guc parameter, as it does not apply to Aurora storage.
5. Changed the '`hot_standby_feedback`' guc parameter to not-modifiable and set the value to '1'.
6. Improved heap page read performance during a vacuum operation.
7. Improved performance of snapshot conflict resolution on read nodes.
8. Improved performance of transaction snapshot acquisition on read nodes.
9. Improved write performance for GIN meta page updates.
- 10Improved buffer cache recovery performance during startup.
- 11Fixes a bug that caused a database engine crash at startup while recovering prepared transactions.
- 12Fixes a bug that could result in the inability to start a read node when there are a large number of prepared transactions.

13Fixes a bug that could cause a read node to report:

ERROR: could not access status of transaction 6080077

DETAIL: Could not open file "pg_subtrans/005C": No such file or directory.

14Fixes a bug that could cause the error below when replicating from RDS PostgreSQL to Aurora PostgreSQL:

FATAL: lock buffer_content is not held

CONTEXT: xlog redo at 46E/F1330870 for Storage/TRUNCATE: base/13322/8058750 to 0 blocks flags 7

15Fixes a bug that could cause Aurora PostgreSQL to hang while replaying a multixact WAL record when replicating from RDS PostgreSQL to Aurora PostgreSQL.

16Multiple improvements to the reliability of importing snapshots from RDS PostgreSQL to Aurora PostgreSQL.

Version 1.0

This version of Aurora PostgreSQL is compatible with PostgreSQL 9.6.3. For more information about the improvements in version 9.6.3 see, [PostgreSQL Release 9.6.3](#).

This version includes the following patch versions:

Patch Versions

- [Version 1.0.11 \(p. 985\)](#)
- [Version 1.0.10 \(p. 985\)](#)
- [Version 1.0.9 \(p. 986\)](#)
- [Version 1.0.8 \(p. 986\)](#)
- [Version 1.0.7 \(p. 986\)](#)

Version 1.0.11

You can find the following improvements in this engine update:

1. Fixes an issue with parallel query execution that can lead to incorrect results.
2. Fixes an issue with visibility map handling during replication from Amazon RDS for PostgreSQL that can cause the Aurora storage volume to become unavailable.
3. Corrects the pg-repack extension.
4. Implements improvements to maintain fresh nodes.
5. Fixes issues that can lead to an engine crash.

Version 1.0.10

This update includes a new feature. You can now replicate an Amazon RDS PostgreSQL DB instance to Aurora PostgreSQL. For more information, see [Replication with Amazon Aurora PostgreSQL \(p. 911\)](#).

You can find the following improvements in this engine update:

1. Adds error logging when a cache exists and a parameter change results in a mismatched buffer cache, storage format, or size.
2. Fixes an issue that causes an engine reboot if there is an incompatible parameter value for huge pages.

3. Improves handling of multiple truncate table statements during a replay of a write ahead log (WAL) on a read node.
 4. Reduces static memory overhead to reduce out-of-memory errors.
 5. Fixes an issue that can lead to out-of-memory errors while performing an insert with a GiST index.
 6. Improves snapshot import from RDS PostgreSQL, removing the requirement that a vacuum be performed on uninitialized pages.
 7. Fixes an issue that causes prepared transactions to return to the previous state following an engine crash.
 8. Implements improvements to prevent read nodes from becoming stale.
 9. Implements improvements to reduce downtime with an engine restart.
- 10 Fixes issues that can cause an engine crash.

Version 1.0.9

In this engine update, we fix an issue that can cause the Aurora storage volume to become unavailable when importing a snapshot from RDS PostgreSQL that contained uninitialized pages.

Version 1.0.8

You can find the following improvements in this engine update:

1. Fixes an issue that prevented the engine from starting if the `shared_preload_libraries` instance parameter contained `pg_hint_plan`.
2. Fixes the error "Attempt to fetch heap block XXX is beyond end of heap (YYY blocks)," which can occur during parallel scans.
3. Improves the effectiveness of prefetching on reads for a vacuum.
4. Fixes issues with snapshot import from RDS PostgreSQL, which can fail if there are incompatible `pg_internal.init` files in the source snapshot.
5. Fixes an issue that can cause a read node to crash with the message "aurora wal replay process (PID XXX) was terminated by signal 11: Segmentation fault". This issue occurs when the reader applied a visibility map change for an uncached visibility map page.

Version 1.0.7

This is the first generally available release of Amazon Aurora with PostgreSQL compatibility.

Best Practices with Amazon Aurora

This topic includes information on general best practices and options for using or migrating data to an Amazon Aurora DB cluster.

Some of the best practices for Amazon Aurora are specific to a particular database engine. For more information about Aurora best practices specific to a database engine, see the following:

Database Engine	Best Practices
Amazon Aurora MySQL	See Best Practices with Amazon Aurora MySQL (p. 763)
Amazon Aurora PostgreSQL	See Best Practices with Amazon Aurora PostgreSQL (p. 953)

Note

For common recommendations for Aurora, see [Using Amazon Aurora Recommendations \(p. 520\)](#).

Topics

- [Amazon Aurora Basic Operational Guidelines \(p. 987\)](#)
- [DB Instance RAM Recommendations \(p. 987\)](#)
- [Monitoring Amazon Aurora \(p. 988\)](#)
- [Working with DB Parameter Groups and DB Cluster Parameter Groups \(p. 988\)](#)
- [Amazon Aurora Best Practices Presentation Video \(p. 988\)](#)

Amazon Aurora Basic Operational Guidelines

The following are basic operational guidelines that everyone should follow when working with Amazon Aurora. Note that the Amazon RDS Service Level Agreement requires that you follow these guidelines:

- Monitor your memory, CPU, and storage usage. Amazon CloudWatch can be set up to notify you when usage patterns change or when you approach the capacity of your deployment, so that 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. Because the underlying IP address of a DB instance can change after a failover, 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 and to 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 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. If you find your read IO is spiking or is higher than usual, you should investigate the DB instances in your DB cluster to see which DB instances are causing the increased IO.
- **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 is a good indication that your queries on this DB instance are going to disk more often than not. In this case, you should investigate your workload to see which queries are causing this behavior.

If, after investigating your workload, you find that you need more memory, scaling up the DB instance class to a class with more RAM could be beneficial. After doing so, you can investigate the metrics above and continue to scale up as necessary. For more information about monitoring a DB cluster, see [Monitoring Amazon Aurora DB Cluster Metrics \(p. 457\)](#).

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 Amazon RDS Management Console, AWS CLI, and CloudWatch API, to view Aurora metrics. For more information, see [Monitoring an Amazon Aurora DB Cluster \(p. 433\)](#).

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 exercise 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 \(p. 379\)](#).

Amazon Aurora Best Practices Presentation Video

The 2016 AWS Summit conference in Chicago included a presentation on best practices for creating and configuring a secure, highly available Amazon Aurora DB Cluster. A video of the presentation is available [here](#).

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 \(p. 284\)](#), [Amazon Aurora MySQL Performance Enhancements \(p. 578\)](#), [Managing Amazon Aurora PostgreSQL \(p. 910\)](#), and [Using Amazon RDS Performance Insights \(p. 476\)](#).

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 is wire-compatible with MySQL 5.6 and MySQL 5.7 and also PostgreSQL 9.6 and PostgreSQL 10.4, 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 \(p. 351\)](#) 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 \(p. 47\)](#), 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 \(p. 47\)](#). 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 \(p. 100\)](#).

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 5.6 compatibility. This combination of database engine and version has the widest compatibility with other Aurora features.
- Turn off **quick 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 **Instance Size**, choose **Memory Optimized** and one of the **xlarge** instance classes. You can adjust the instance class up or down later.
- Under **Multi-AZ Deployment**, choose **Create Replica in Different Zone**. 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.

- Enable 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 \(p. 166\)](#).

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 AWS whitepaper [Aurora Migration Handbook](#).

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 \(p. 587\)](#) or [Migrating Data to Amazon Aurora with PostgreSQL Compatibility \(p. 887\)](#) 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 MySQL or 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 \(p. 995\)](#).

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 \(p. 288\)](#). For the configuration settings that are or aren't applicable to Aurora clusters, see [Aurora MySQL Parameters \(p. 773\)](#) or [Amazon Aurora PostgreSQL Parameters \(p. 960\)](#) 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 \(p. 166\)](#). 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 \(p. 994\)](#).

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 \(p. 383\)](#) and [Restoring from a DB Cluster Snapshot \(p. 385\)](#).

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.

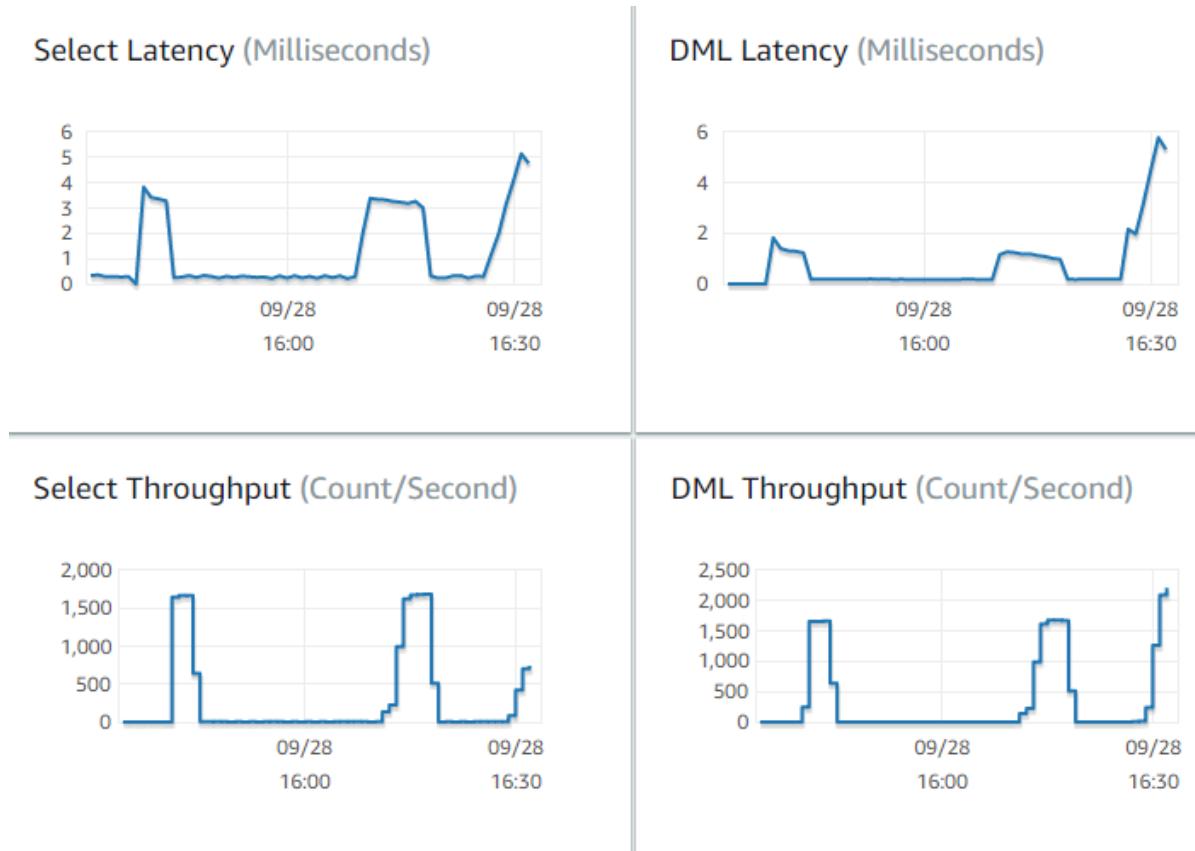
To have more choices on the **Monitoring** tab, enable Enhanced Monitoring and Performance Insights in the cluster settings. You can also enable those choices later if you didn't choose them when setting up the cluster.

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 [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 \(p. 639\)](#).

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 \(p. 284\)](#), [Amazon Aurora MySQL Performance Enhancements \(p. 578\)](#), [Managing Amazon Aurora PostgreSQL \(p. 910\)](#), and [Using Amazon RDS Performance Insights \(p. 476\)](#). 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

Limits for Amazon Aurora

This topic describes the resource limits and naming constraints for Amazon Aurora.

Topics

- [Limits in Amazon Aurora \(p. 1001\)](#)
- [Naming Constraints in Amazon Aurora \(p. 1002\)](#)
- [Amazon Aurora File Size Limits \(p. 1003\)](#)

Limits in Amazon Aurora

Each AWS account has limits, for each AWS Region, on the number of Amazon Aurora resources that can be created. Once a limit for a resource has been reached, additional calls to create that resource fail with an exception.

The following table lists the resources and their limits per region.

Resource	Default Limit
Clusters	40
Cluster parameter groups	50
Cross-region snapshots copy requests	5
DB Instances	40
Event subscriptions	20
Manual snapshots	100
Manual cluster snapshots	100
Parameter groups	50
Reserved instances	40
Rules per VPC security group	50 inbound 50 outbound
VPC Security groups	5
Subnet groups	50
Subnets per subnet group	20
Tags per resource	50

Note

By default, you can have up to a total of 40 DB instances. If your application requires more DB instances, you can request additional DB instances using this request form [Request RDS DB instance limit](#).

Naming Constraints in Amazon Aurora

The following table describes naming constraints in Amazon Aurora.

DB instance identifier	<ul style="list-style-type: none"> Must contain 1 to 63 alphanumeric characters or hyphens. First character must be a letter. Cannot end with a hyphen or contain two consecutive hyphens. Must be unique for all DB instances per AWS account, per region.
Database name	<p>Database name constraints differ for each database engine.</p> <p>Amazon Aurora MySQL</p> <ul style="list-style-type: none"> Must contain 1 to 64 alphanumeric characters. Cannot be a word reserved by the database engine. <p>Amazon Aurora PostgreSQL</p> <ul style="list-style-type: none"> Must contain 1 to 63 alphanumeric characters. Must begin with a letter or an underscore. Subsequent characters can be letters, underscores, or digits (0-9). Cannot be a word reserved by the database engine.
Master user name	<p>Master user name constraints differ for each database engine.</p> <p>Amazon Aurora MySQL</p> <ul style="list-style-type: none"> Must contain 1 to 16 alphanumeric characters. First character must be a letter. Cannot be a word reserved by the database engine. <p>Amazon Aurora PostgreSQL</p> <ul style="list-style-type: none"> Must contain 1 to 63 alphanumeric characters. First character must be a letter. Cannot be a word reserved by the database engine.
Master password	<p>The password for the master database user can be any printable ASCII character except "/", "", or "@". Master password constraints differ for each database engine.</p> <p>Amazon Aurora MySQL</p> <ul style="list-style-type: none"> Must contain 8 to 41 characters. <p>Amazon Aurora PostgreSQL</p> <ul style="list-style-type: none"> Must contain 8 to 128 characters.
DB parameter group name	<ul style="list-style-type: none"> Must contain from 1 to 255 alphanumeric characters.

	<ul style="list-style-type: none">First character must be a letter.Hyphens are allowed, but the name cannot end with a hyphen or contain two consecutive hyphens.
DB subnet group name	<ul style="list-style-type: none">Must contain from 1 to 255 characters.Alphanumeric characters, spaces, hyphens, underscores, and periods are allowed.

Amazon Aurora File Size Limits

With Aurora, the table size limit is only constrained by the size of the Aurora cluster volume. As a result, the maximum table size for an Aurora MySQL DB cluster is 64 tebibytes (TiB) and for an Aurora PostgreSQL DB cluster it is 32 TiB. We recommend that you follow table design best practices, such as partitioning of large tables.

Troubleshooting for Aurora

Use the following sections to help troubleshoot problems you have with DB instances in Amazon RDS and Aurora.

Topics

- [Cannot Connect to Amazon RDS DB Instance \(p. 1004\)](#)
- [Amazon RDS Security Issues \(p. 1005\)](#)
- [Resetting the DB Instance Owner Role Password \(p. 1005\)](#)
- [Amazon RDS DB Instance Outage or Reboot \(p. 1006\)](#)
- [Amazon RDS DB Parameter Changes Not Taking Effect \(p. 1006\)](#)
- [Amazon Aurora MySQL Out of Memory Issues \(p. 1006\)](#)
- [Amazon Aurora MySQL Replication Issues \(p. 1007\)](#)
- [No Space Left on Device Error \(p. 1010\)](#)

For information about debugging problems using the Amazon RDS API, see [Troubleshooting Applications on Aurora \(p. 1012\)](#).

Cannot Connect to Amazon RDS DB Instance

When you cannot connect to a DB instance, the following are common causes:

- The access rules enforced by your local firewall and the ingress IP addresses that you authorized to access your DB instance in the instance's security group aren't in sync. The problem is most likely the ingress rules in your security group.

By default, DB instances don't allow access; access is granted through a security group. To grant access, you must create your own security group with specific ingress and egress rules for your situation. If necessary, add rules to the security group associated with the VPC that allow traffic related to the source in and out of the DB instance. You can specify an IP address, a range of IP addresses, or another VPC security group.

For more information about setting up a security group, see [Provide Access to the DB Cluster in the VPC by Creating a Security Group \(p. 52\)](#).

- The port you specified when you created the DB instance cannot be used to send or receive communications due to your local firewall restrictions. In this case, check with your network administrator to determine if your network allows the specified port to be used for inbound and outbound communication.
- Your DB instance is still being created and is not yet available. Depending on the size of your DB instance, it can take up to 20 minutes before an instance is available.

Testing a Connection to an Amazon RDS DB Instance

You can test your connection to a DB instance using common Linux or Windows tools.

From a Linux or Unix terminal, you can test the connection by typing 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.c6c8mn7tsdgv0.us-west-2.rds.amazonaws.com 8299
Connection to postgresql1.c6c8mn7tsdgv0.us-west-2.rds.amazonaws.com 8299 port [tcp/vvv-data] succeeded!
```

Windows users can use Telnet to test the connection to a DB instance. Note that Telnet actions are not supported other than for testing the connection. If a connection is successful, the action returns no message. If a connection is not successful, you receive an error message such as the following:

```
C:\>telnet sg-postgresql1.c6c8mntzhgv0.us-west-2.rds.amazonaws.com 819
Connecting To sg-postgresql1.c6c8mntzhgv0.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 does not 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 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 \(p. 49\)](#).

Error Message "Failed to retrieve account attributes, certain console functions may be impaired."

There are several reasons you would get this error; it could be because your account is missing permissions, or your account has not been properly set up. If your account is new, you may not have waited for the account to be ready. If this is an existing account, you could 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 Role Password

You can reset the assigned permissions for your DB instance by resetting the master password. For example, if you lock yourself out of the db_owner role on your SQL Server database, you can reset the db_owner role password by modifying the DB instance master password. By changing the DB instance password, you can regain access to the DB instance, access databases using the modified password for the db_owner, and restore privileges for the db_owner role that may have been accidentally revoked. 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](#) operation.

Amazon RDS DB Instance Outage or Reboot

A DB instance outage can occur when a DB instance is rebooted, 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 manually reboot your DB instance or when you change a DB instance setting that requires a reboot before it can take effect.

When you modify a setting for a DB instance, you can determine when the change is applied by using the **Apply Immediately** setting.

A DB instance reboot only 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 will not take effect until the DB instance associated with the parameter group is rebooted. The change requires a manual reboot; the DB instance will not automatically be rebooted during the maintenance window.

Amazon RDS DB Parameter Changes Not Taking Effect

If you change a parameter in a DB parameter group but you don't see the changes take effect, you most 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 won'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, such as calling [ModifyDBInstance](#) to change DB instance class. For more information, see [Modifying Parameters in a DB Parameter Group \(p. 291\)](#).

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 there should be no actions taken and effectively disables the feature.

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` – Kills the queries in descending order of memory consumption until the instance memory surfaces above the low threshold. DDL statements are not killed.
- `print, tune` – Performs actions described for both `print` and `tune`.
- `tune, decline, kill_query` – Performs the actions described for `tune`, `decline`, and `kill_query`.

For 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 and MariaDB replication problems also apply to Aurora MySQL. You can diagnose and correct these problems.

Diagnosing and Resolving Lag Between Read Replicas

After you create a MySQL or MariaDB Read Replica and the Read Replica is available, Amazon RDS first replicates the changes made to the source DB instance from the time the create Read Replica operation was initiated. During this phase, the replication lag time for the Read Replica will be 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 or MariaDB `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 or MariaDB Read Replication Failure \(p. 1008\)](#). A `AuroraBinlogReplicaLag` value of -1 can also mean that the `Seconds_Behind_Master` value cannot be determined or is `NULL`.

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.

Because the MySQL and MariaDB read replication technology is asynchronous, 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, a high volume of write operations to the source DB instance can occur in parallel, while write operations to the Read Replica are serialized using a single I/O thread, can lead to a lag between the source instance and Read Replica. For more information about Read Replicas and MySQL, go to [Replication Implementation Details](#) in the MySQL documentation. For more information about Read Replicas and MariaDB, go to [Replication Overview](#) in the MariaDB 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.
- Ensure 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, InnoDB for MariaDB 10.2 or higher, or XtraDB for MariaDB 10.1 or lower. If you have a small set of tables that are being updated often, and you are using the InnoDB or XtraDB table schema, then 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, which can reduce replica lag. The following shows an example.

For Linux, OS X, 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 or MariaDB 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 or MariaDB engines by viewing the **Replication Error** field. Events that indicate the status of the Read Replica are also generated, including [RDS-EVENT-0045 \(p. 548\)](#), [RDS-EVENT-0046 \(p. 548\)](#), and [RDS-EVENT-0047 \(p. 547\)](#). For more information about events and subscribing to events, see [Using Amazon RDS Event Notification \(p. 543\)](#). If a MySQL error message is returned, review the error in the [MySQL error message documentation](#). If a MariaDB error message is returned, review the error in the [MariaDB 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 that is used to specify the maximum size of data manipulation language (DML) that can be executed on the database. If the `max_allowed_packet` parameter value for the source DB instance is smaller than the `max_allowed_packet` parameter 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 are creating indexes on a Read Replica, you need to have the `read_only` parameter set to `0` to create the indexes. If you are writing to tables on the Read Replica, it can break replication.
- Using a non-transactional storage engine such as MyISAM. Read replicas require a transactional storage engine. Replication is only supported for the following storage engines: InnoDB for MySQL, InnoDB for MariaDB 10.2 or higher, or XtraDB for MariaDB 10.1 or lower.

You can convert a MyISAM table to InnoDB with the following command:

```
alter table <schema>.<table_name> engine=innodb;
```

- Using unsafe non-deterministic queries such as `SYSDATE()`. For more information, see [Determination of Safe and Unsafe Statements in Binary Logging](#).

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 MySQL or MariaDB 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 binlog position issue, you can change the slave replay position with the `mysql_rds_next_master_log` command. Your MySQL or MariaDB DB instance must be running a version that supports the `mysql_rds_next_master_log` command in order to change the slave 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 or MariaDB Read Replica Problem](#).

Slave Down or Disabled Error

When you call the `mysql.rds_skip_repl_error` command, you might receive the following error message: `Slave is down or disabled`.

This error message appears because replication has stopped and could not 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 master. 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 and specify a configuration parameter of 'binlog retention hours' along with the number of hours to retain binlog files

on the DB cluster, up to 720 (30 days). The following example sets the retention period for binlog files to 48 hours:

```
CALL mysql.rds_set_configuration('binlog retention hours', 48);
```

No Space Left on Device Error

You might encounter the following error message from Amazon Aurora:

```
ERROR 3 (HY000): Error writing file '/rdsdbdata/tmp/XXXXXXXX' (Errcode: 28 - No space left on device)
```

Each DB instance in an Amazon Aurora DB cluster uses local SSD storage to store temporary tables for a session. This local storage for temporary tables does not autogrow like the Aurora cluster volume. Instead, the amount of local storage is limited. The limit is based on the DB instance class for DB instances in your DB cluster.

If your workload cannot be modified to reduce the amount temporary storage required, then you can scale your DB instances up to use a DB instance class that has more local SSD storage.

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 \(p. 1011\)](#)
- [Troubleshooting Applications on Aurora \(p. 1012\)](#)

Using the Query API

The following sections discuss the parameters and request authentication used with the Query API.

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 \(p. 1012\)](#)
- [Troubleshooting Tips \(p. 1012\)](#)

For information about troubleshooting for Amazon RDS DB instances, see [Troubleshooting for Aurora \(p. 1004\)](#).

Retrieving Errors

Typically, you want your application to check whether a request generated an error before you spend any time processing results. The easiest way to find out if an error occurred is to look for an `Error` node in the response from the Amazon RDS API.

XPath syntax provides a simple way to search for the presence of an `Error` node, as well as an easy way to retrieve the error code and message. The following code snippet uses Perl and the `XML::XPath` module to determine if an error occurred during a request. If an error occurred, the code prints the first error code and message in the response.

```
use XML::XPath;
my $xp = XML::XPath->new(xml =>$response);
if ( $xp->find("//Error") )
{print "There was an error processing your request:\n", " Error code: ",
$xp->findvalue("//Error[1]/Code"), "\n", " ",
$xp->findvalue("//Error[1]/Message"), "\n\n"; }
```

Troubleshooting Tips

We recommend the following processes to diagnose and resolve problems with the Amazon RDS API.

- Verify that Amazon RDS is operating normally in the AWS Region you are targeting by visiting <http://status.aws.amazon.com>.
- Check the structure of your request

Each Amazon RDS operation has a reference page in the *Amazon RDS API Reference*. Double-check that you are using parameters correctly. In order to give you ideas regarding what might be wrong, look at the sample requests or user scenarios to see if those examples are doing similar operations.

- Check the forum

Amazon RDS has a development community forum where you can search for solutions to problems others have experienced along the way. To view the forum, go to

<https://forums.aws.amazon.com/>

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*.

update-history-change	update-history-description	update-history-date
Aurora MySQL release notes in Document History (p. 1013)	This section now includes history entries for Aurora MySQL 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. 1013)	You can reduce the overhead of connection management on your cluster, and reduce the chance of "too many connections" errors, by using the Amazon RDS Proxy. You associate each proxy with an RDS DB instance or Aurora DB cluster. Then you use the proxy endpoint in the connection string for your application. The Amazon RDS Proxy is currently in a public preview state. It supports the Aurora MySQL database engine. For more information, see Managing Connections with Amazon RDS Proxy (Preview) .	December 3, 2019
Data API for Aurora Serverless supports data type mapping hints (p. 1013)	You can now use a hint to instruct the Data API for Aurora Serverless to send a String value to the database as a different type. For more information, see Calling the Data API .	November 26, 2019
Data API for Aurora Serverless supports a Java client library (Preview) (p. 1013)	You can download and use a Java client library for Data API. It allows you to map your client-side classes to requests and responses of the Data API. For	November 26, 2019

	more information, see Using the Java Client Library for Data API .	
Aurora PostgreSQL version 3.0 (p. 1013)	Amazon Aurora with PostgreSQL compatibility version 3.0 is available and compatible with PostgreSQL 11.4. Supported AWS Regions include us-east-1, us-east-2, us-west-2, eu-west-1, ap-northeast-1, and ap-northeast-2. For more information, see Database Engine Versions for Amazon Aurora PostgreSQL .	November 26, 2019
Aurora PostgreSQL is FedRAMP HIGH eligible (p. 1013)	Aurora PostgreSQL is FedRAMP HIGH eligible. For details about AWS and compliance efforts, see AWS Services In Scope by Compliance Program .	November 26, 2019
READ COMMITTED isolation level enabled for Amazon Aurora MySQL Replicas (p. 1013)	You can now enable the READ COMMITTED isolation level on Aurora MySQL Replicas. Doing so requires enabling the <code>aurora_read_replica_read_committed_isolation_enabled</code> configuration setting at the instance level. Using the READ COMMITTED isolation level for long-running queries on OLTP clusters can help address issues with history list length. Before enabling this setting, be sure to understand how the isolation behavior on Aurora Replicas differs from the usual MySQL implementation of READ COMMITTED. For more information, see Aurora MySQL Isolation Levels .	November 25, 2019
Performance Insights supports analyzing statistics of running Aurora PostgreSQL queries (p. 1013)	You can now analyze statistics of running queries with Performance Insights for Aurora PostgreSQL DB instances. For more information, see Analyzing Statistics of Running Queries .	November 25, 2019
More clusters in an Aurora global database (p. 1013)	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 Database, see Working with Amazon Aurora Global Database .	November 25, 2019

Aurora machine learning support in Aurora MySQL (p. 1013)	In Aurora MySQL 2.07 and higher, you can call Amazon Comprehend for sentiment analysis and Amazon SageMaker for a wide variety of machine learning algorithms. You use the results directly in your database application by embedding calls to stored functions in your queries. For more information, see Using Machine Learning (ML) Capabilities with Aurora .	November 25, 2019
Aurora MySQL version 2.07.0 (p. 798)	Aurora MySQL version 2.07.0 is available.	November 25, 2019
Aurora MySQL version 1.22.0 (p. 831)	Aurora MySQL version 1.22.0 is available.	November 25, 2019
Aurora MySQL version 1.21.0 (p. 833)	Aurora MySQL version 1.21.0 is available.	November 25, 2019
Aurora Global Database no longer requires engine mode setting (p. 1013)	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 Database, see Working with Amazon Aurora Global Database .	November 25, 2019
Aurora Global Database is available for Aurora MySQL version 2 (p. 1013)	Starting in Aurora MySQL 2.07, you can create an Aurora Global Database with MySQL 5.7 compatibility. You don't need to specify the <code>global</code> engine mode for the primary or secondary clusters. You can add any new provisioned cluster with Aurora MySQL 2.07 or higher to an Aurora Global Database. For information about Aurora Global Database, see Working with Amazon Aurora Global Database .	November 25, 2019
Aurora MySQL version 2.06.0 (p. 801)	Aurora MySQL version 2.06.0 is available.	November 22, 2019
Aurora MySQL version 2.04.8 (p. 805)	Aurora MySQL version 2.04.8 is available.	November 20, 2019

Aurora MySQL hot row contention optimization available without lab mode (p. 1013)	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. 1013)	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 equi-join. For more information about using this feature, see Working with Hash Joins in Aurora MySQL .	November 19, 2019
Aurora MySQL 2.* support for more db.r5 instance classes (p. 1013)	Aurora MySQL clusters now support the instance types db.r5.8xlarge, db.r5.16xlarge, and db.r5.24xlarge. For more information about instance types for Aurora MySQL clusters, see Choosing the DB Instance Class .	November 19, 2019
Aurora MySQL 2.* support for backtracking (p. 1013)	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. Read the AWS blog for an overview, and refer to Backtracking an Aurora DB Cluster , for more details.	November 19, 2019
Aurora MySQL version 2.04.7 (p. 807)	Aurora MySQL version 2.04.7 is available.	November 14, 2019
Aurora MySQL version 2.05.0 (p. 803)	Aurora MySQL version 2.05.0 is available.	November 11, 2019
Aurora MySQL version 1.20.0 (p. 835)	Aurora MySQL version 1.20.0 is available.	November 11, 2019

Billing tag support for Aurora (p. 1013)	You can now use tags to keep track of cost allocation for resources such as Aurora clusters, DB instances within Aurora clusters, I/O, backups, snapshots, and so on. You can see costs associated with each tag using AWS Cost Explorer. For more information about using tags with Aurora, see Tagging Amazon RDS Resources . For general information about tags and ways to use them for cost analysis, see Using Cost Allocation Tags and User-Defined Cost Allocation Tags	October 23, 2019
Data API for Aurora PostgreSQL (p. 1013)	Aurora PostgreSQL now supports using the Data API with Amazon Aurora Serverless DB clusters. For more information, see Using the Data API for Aurora Serverless .	September 23, 2019
Aurora MySQL version 2.04.6 (p. 808)	Aurora MySQL version 2.04.6 is available.	September 19, 2019
Aurora MySQL version 1.19.5 (p. 836)	Aurora MySQL version 1.19.5 is available.	September 19, 2019
Aurora PostgreSQL supports uploading database logs to CloudWatch logs (p. 1013)	You can configure your Aurora PostgreSQL DB cluster to publish log data to a log group in Amazon CloudWatch Logs. With CloudWatch Logs, you can perform real-time analysis of the log data, and use CloudWatch to create alarms and view metrics. You can use CloudWatch Logs to store your log records in highly durable storage. For more information, see Publishing Aurora PostgreSQL Logs to Amazon CloudWatch Logs .	August 9, 2019
Multi-master clusters for Aurora MySQL (p. 1013)	You can set up Aurora MySQL multi-master clusters. In these clusters, each DB instance has read-write capability. For more information, see Working with Aurora Multi-Master Clusters .	August 8, 2019

Aurora PostgreSQL supports Aurora Serverless (p. 1013)	You can now use Amazon Aurora Serverless with Aurora PostgreSQL. An Aurora Serverless DB cluster automatically starts up, shuts down, and scales up or down its compute capacity based on your application's needs. For more information, see Using Amazon Aurora Serverless .	July 9, 2019
Aurora MySQL version 2.04.5 (p. 810)	Aurora MySQL version 2.04.5 is available.	July 8, 2019
Aurora PostgreSQL versions 2.3.3 and 1.5.2 (p. 1013)	Amazon Aurora with PostgreSQL compatibility version 2.3.3 is available and compatible with PostgreSQL 10.7. Amazon Aurora with PostgreSQL compatibility version 1.5.2 is available and compatible with PostgreSQL 9.6.12. For more information, see Database Engine Versions for Amazon Aurora PostgreSQL .	July 3, 2019
Cross-account cloning for Aurora MySQL (p. 1013)	You can now clone the cluster volume for an Aurora MySQL DB cluster between AWS accounts. You authorize the sharing through AWS Resource Access Manager (AWS RAM). The cloned cluster volume uses a copy-on-write mechanism, which only requires additional storage for new or changed data. For more information about cloning for Aurora, see Cloning Databases in an Aurora DB Cluster .	July 2, 2019
Aurora PostgreSQL versions 2.3.1 and 1.5.1 (p. 1013)	Amazon Aurora with PostgreSQL compatibility version 2.3.1 is available and compatible with PostgreSQL 10.7. Amazon Aurora with PostgreSQL compatibility version 1.5.1 is available and compatible with PostgreSQL 9.6.12. For more information, see Database Engine Versions for Amazon Aurora PostgreSQL .	July 2, 2019
Aurora PostgreSQL supports db.t3 DB instance classes (p. 1013)	You can now create Aurora PostgreSQL DB clusters that use the db.t3 DB instance classes. For more information, see DB Instance Class .	June 20, 2019

Support for importing data from Amazon S3 for Aurora PostgreSQL (p. 1013)	You can now import data from an Amazon S3 file into a table in an Aurora PostgreSQL DB cluster. For more information, see Importing Amazon S3 Data into an Aurora PostgreSQL DB Cluster .	June 19, 2019
Aurora PostgreSQL now provides fast failover recovery with cluster cache management (p. 1013)	Aurora with PostgreSQL compatibility now provides cluster cache management to ensure fast recovery of the primary DB instance in the event of a failover. For more information, see Fast Recovery after Failover with Cluster Cache Management .	June 11, 2019
Aurora MySQL version 1.19.2 (p. 837)	Aurora MySQL version 1.19.2 is available.	June 5, 2019
Data API for Aurora Serverless generally available (p. 1013)	You can access Aurora Serverless clusters with web services-based applications using the Data API. For more information, see Using the Data API for Aurora Serverless .	May 30, 2019
Aurora PostgreSQL version 2.3 (p. 1013)	Version 2.3 of Amazon Aurora with PostgreSQL compatibility is available and compatible with PostgreSQL 10.7. For more information, see Version 2.3 .	May 30, 2019
Aurora PostgreSQL supports database monitoring with database activity streams (p. 1013)	Aurora with PostgreSQL compatibility now includes database activity streams, which provide a near real-time data stream of the database activity in your relational database. For more information, see Using Database Activity Streams .	May 30, 2019
Aurora MySQL version 2.04.4 (p. 811)	Aurora MySQL version 2.04.4 is available.	May 29, 2019
Amazon Aurora recommendations (p. 1013)	Amazon Aurora now provides automated recommendations for Aurora resources. For more information, see Using Amazon Aurora Recommendations .	May 22, 2019

Aurora PostgreSQL versions 1.2.2, 1.3.2, 2.0.1, 2.1.1, 2.2.1 (p. 1013)	The following patch versions for Amazon Aurora with PostgreSQL compatibility are now available and include versions 1.2.2, 1.3.2, 2.0.1, 2.1.1, and 2.2.1. For more information, see Database Engine Versions for Amazon Aurora PostgreSQL .	May 21, 2019
Performance Insights support for Aurora Global Database (p. 1013)	You can now use Performance Insights with Aurora Global Database. For information about Performance Insights for Aurora, see Using Amazon RDS Performance Insights . For information about Aurora global databases, see Working with Aurora Global Database .	May 13, 2019
Aurora PostgreSQL version 1.4 (p. 1013)	Version 1.4 of Amazon Aurora with PostgreSQL compatibility is available and compatible with PostgreSQL 9.6.11. For more information, see Version 1.4 .	May 9, 2019
Aurora MySQL version 2.04.3 (p. 813)	Aurora MySQL version 2.04.3 is available.	May 9, 2019
Aurora MySQL version 1.19.1 (p. 838)	Aurora MySQL version 1.19.1 is available.	May 9, 2019
Performance Insights is available for Aurora MySQL 5.7 (p. 1013)	Amazon RDS Performance Insights is now available for Aurora MySQL 2.x versions, which are compatible with MySQL 5.7. For more information, see Using Amazon RDS Performance Insights .	May 3, 2019
Aurora MySQL version 2.04.2 (p. 814)	Aurora MySQL version 2.04.2 is available.	May 2, 2019
Aurora global databases available in more AWS Regions (p. 1013)	You can now create Aurora global databases in most AWS Regions where Aurora is available. For information about Aurora global databases, see Working with Amazon Aurora Global Databases .	April 30, 2019

Minimum Capacity of 1 for Aurora Serverless (p. 1013)	The minimum capacity setting you can use for an Aurora Serverless cluster is 1. Formerly, the minimum was 2. For information about specifying Aurora Serverless capacity values, see Setting the Capacity of an Aurora Serverless DB Cluster .	April 29, 2019
Aurora Serverless timeout action (p. 1013)	You can now specify the action to take when an Aurora Serverless capacity change times out. For more information, see Timeout Action for Capacity Changes .	April 29, 2019
Per-second billing (p. 1013)	Amazon RDS is now billed in 1-second increments in all AWS Regions except AWS GovCloud (US) for on-demand instances. For more information, see DB Instance Billing for Aurora .	April 25, 2019
Sharing Aurora Serverless snapshots across AWS Regions (p. 1013)	With Aurora Serverless, snapshots are always encrypted. If you encrypt the snapshot with your own AWS Key Management Service key, you can now copy or share the snapshot across AWS Regions. For information about snapshots of Aurora Serverless DB clusters, see Aurora Serverless and Snapshots .	April 17, 2019
Restore MySQL 5.7 backups from Amazon S3 (p. 1013)	You can now create a backup of your MySQL version 5.7 database, store it on Amazon S3, and then restore the backup file onto a new Aurora MySQL DB cluster. For more information, see Migrating Data from an External MySQL Database to an Aurora MySQL DB Cluster .	April 17, 2019
Sharing Aurora Serverless snapshots across regions (p. 1013)	With Aurora Serverless, snapshots are always encrypted. If you encrypt the snapshot with your own KMS key, you can now copy or share the snapshot across regions. For information about snapshots of Aurora Serverless DB clusters, see Aurora Serverless and Snapshots .	April 16, 2019

Aurora proof-of-concept tutorial (p. 1013)	You can learn how to perform a proof of concept to try your application and workload with Aurora. For the full tutorial, see Performing an Aurora Proof of Concept .	April 16, 2019
Aurora Serverless supports restoring from an Amazon S3 backup (p. 1013)	You can now import backups from Amazon S3 to an Aurora Serverless cluster. For details about that procedure, see Migrating Data from MySQL by Using an Amazon S3 Bucket .	April 16, 2019
New modifiable parameters for Aurora Serverless (p. 1013)	You can now modify the following DB parameters for an Aurora Serverless cluster: <code>innodb_file_format</code> , <code>innodb_file_per_table</code> , <code>innodb_large_prefix</code> , <code>innodb_lock_wait_timeout</code> , <code>innodb_monitor_disable</code> , <code>innodb_monitor_enable</code> , <code>innodb_monitor_reset</code> , <code>innodb_monitor_reset_all</code> , <code>innodb_print_all_deadlocks</code> , <code>log_warnings</code> , <code>net_read_timeout</code> , <code>net_retry_count</code> , <code>net_write_timeout</code> , <code>sql_mode</code> , and <code>tx_isolation</code> . For more information about configuration parameters for Aurora Serverless clusters, see Aurora Serverless and Parameter Groups .	April 4, 2019
Aurora PostgreSQL supports db.r5 DB instance classes (p. 1013)	You can now create Aurora PostgreSQL DB clusters that use the db.r5 DB instance classes. For more information, see DB Instance Class .	April 4, 2019
Aurora PostgreSQL logical replication (p. 1013)	You can now use PostgreSQL logical replication to replicate parts of a database for an Aurora PostgreSQL DB cluster. For more information, see Using PostgreSQL Logical Replication .	March 28, 2019

GTID support for Aurora MySQL 2.04 (p. 1013)	You can now use replication with the global transaction ID (GTID) feature of MySQL 5.7. This feature simplifies performing binary log (binlog) replication between Aurora MySQL and an external MySQL 5.7-compatible database. The replication can use the Aurora MySQL cluster as the source or the destination. This feature is available for Aurora MySQL 2.04 and higher. For more information about GTID-based replication and Aurora MySQL, see Using GTID-Based Replication for Aurora MySQL .	March 25, 2019
Aurora MySQL version 2.04.1 (p. 815)	Aurora MySQL version 2.04.1 is available.	March 25, 2019
Aurora MySQL version 2.04 (p. 816)	Aurora MySQL version 2.04 is available.	March 25, 2019
Uploading Aurora Serverless logs to Amazon CloudWatch (p. 1013)	You can now have Aurora upload database logs to CloudWatch for an Aurora Serverless cluster. For more information, see Viewing Aurora Serverless DB Clusters . As part of this enhancement, you can now define values for instance-level parameters in a DB cluster parameter group, and those values apply to all DB instances in the cluster unless you override them in the DB parameter group. For more information, see Working with DB Parameter Groups and DB Cluster Parameter Groups .	February 25, 2019
Aurora MySQL supports db.t3 DB instance classes (p. 1013)	You can now create Aurora MySQL DB clusters that use the db.t3 DB instance classes. For more information, see DB Instance Class .	February 25, 2019
Aurora MySQL supports db.r5 DB instance classes (p. 1013)	You can now create Aurora MySQL DB clusters that use the db.r5 DB instance classes. For more information, see DB Instance Class .	February 25, 2019

Performance Insights counters for Aurora MySQL (p. 1013)	You can now add performance counters to your Performance Insights charts for Aurora MySQL DB instances. For more information, see Performance Insights Dashboard Components .	February 19, 2019
Aurora PostgreSQL version 2.2.0 (p. 1013)	Version 2.2.0 of Aurora with PostgreSQL compatibility is available and compatible with PostgreSQL 10.6. For more information, see Version 2.2.0 .	February 13, 2019
Aurora MySQL version 2.03.4 (p. 817)	Aurora MySQL version 2.03.4 is available.	February 7, 2019
Aurora MySQL version 1.19.0 (p. 838)	Aurora MySQL version 1.19.0 is available.	February 7, 2019
Amazon RDS Performance Insights supports viewing more SQL text for Aurora MySQL (p. 1013)	Amazon RDS Performance Insights now supports viewing more SQL text in the Performance Insights dashboard for Aurora MySQL DB instances. For more information, see Viewing More SQL Text in the Performance Insights Dashboard .	February 6, 2019
Amazon RDS Performance Insights supports viewing more SQL text for Aurora PostgreSQL (p. 1013)	Amazon RDS Performance Insights now supports viewing more SQL text in the Performance Insights dashboard for Aurora PostgreSQL DB instances. For more information, see Viewing More SQL Text in the Performance Insights Dashboard .	January 24, 2019
Aurora MySQL version 2.03.3 (p. 817)	Aurora MySQL version 2.03.3 is available.	January 18, 2019
Aurora MySQL version 1.17.8 (p. 840)	Aurora MySQL version 1.17.8 is available.	January 17, 2019
Aurora MySQL version 2.03.2 (p. 818)	Aurora MySQL version 2.03.2 is available.	January 9, 2019

Aurora backup billing (p. 1013)	You can use the Amazon CloudWatch metrics <code>TotalBackupStorageBilled</code> , <code>SnapshotStorageUsed</code> , and <code>BackupRetentionPeriodStorageUsed</code> to monitor the space usage of your Aurora backups. For more information about how to use CloudWatch metrics, see Overview of Monitoring . For more information about how to manage storage for backup data, see Understanding Aurora Backup Storage Usage .	January 3, 2019
Performance Insights counters (p. 1013)	You can now add performance counters to your Performance Insights charts. For more information, see Performance Insights Dashboard Components .	December 6, 2018
Aurora Global Database (p. 1013)	You can now create Aurora global databases. An Aurora global database spans multiple AWS Regions, enabling low latency global reads and disaster recovery from region-wide outages. For more information, see Working with Amazon Aurora Global Database .	November 28, 2018
Query plan management in Aurora PostgreSQL (p. 1013)	Aurora with PostgreSQL compatibility now provides query plan management that you can use to manage PostgreSQL query execution plans. For more information, see Managing Query Execution Plans for Aurora PostgreSQL .	November 20, 2018
Query editor for Aurora Serverless (beta) (p. 1013)	You can run SQL statements in the Amazon RDS console on Aurora Serverless clusters. For more information, see Using the Query Editor for Aurora Serverless .	November 20, 2018
Data API for Aurora Serverless (beta) (p. 1013)	You can access Aurora Serverless clusters with web services-based applications using the Data API. For more information, see Using the Data API for Aurora Serverless .	November 20, 2018

Aurora PostgreSQL version 2.1 (p. 1013)	Aurora with PostgreSQL compatibility version 2.1 is available and compatible with PostgreSQL 10.5. For more information, see Version 2.1 .	November 20, 2018
TLS support for Aurora Serverless (p. 1013)	Aurora Serverless clusters support TLS/SSL encryption. For more information, see TLS/SSL for Aurora Serverless .	November 19, 2018
Custom endpoints (p. 1013)	You can now create endpoints that are associated with an arbitrary set of DB instances. This feature helps with load balancing and high availability for Aurora clusters where some DB instances have different capacity or configuration than others. You can use custom endpoints instead of connecting to a specific DB instance through its instance endpoint. For more information, see Amazon Aurora Connection Management .	November 12, 2018
IAM authentication support in Aurora PostgreSQL (p. 1013)	Aurora with PostgreSQL compatibility now supports IAM authentication. For more information see IAM Database Authentication .	November 8, 2018
Aurora MySQL version 2.03.1 (p. 819)	Aurora MySQL version 2.03.1 is available.	October 24, 2018
Custom parameter groups for restore and point in time recovery (p. 1013)	You can now specify a custom parameter group when you restore a snapshot or perform a point in time recovery operation. For more information, see Restoring from a DB Cluster Snapshot and Restoring a DB Cluster to a Specified Time .	October 15, 2018
Aurora MySQL version 2.03 (p. 819)	Aurora MySQL version 2.03 is available.	October 11, 2018
Aurora MySQL version 2.02.5 (p. 820)	Aurora MySQL version 2.02.5 is available.	October 8, 2018
Aurora MySQL version 1.17.7 (p. 841)	Aurora MySQL version 1.17.7 is available.	October 8, 2018
Deletion protection for Aurora DB clusters (p. 1013)	When you enable deletion protection for a DB cluster, the database cannot be deleted by any user. For more information, see Deleting a DB Instance in a DB Cluster .	September 26, 2018

Aurora PostgreSQL version 2.0 (p. 1013)	Aurora with PostgreSQL compatibility version 2.0 is available and compatible with PostgreSQL 10.4. For more information, see Version 2.0 .	September 25, 2018
Stop/Start feature Aurora (p. 1013)	You can now stop or start an entire Aurora cluster with a single operation. For more information, see Stopping and Starting an Aurora Cluster .	September 24, 2018
Aurora MySQL version 2.02.4 (p. 821)	Aurora MySQL version 2.02.4 is available.	September 21, 2018
Parallel Query feature for Aurora MySQL (p. 1013)	Aurora MySQL now offers an option to parallelize I/O work for queries across the Aurora storage infrastructure. This feature speeds up data-intensive analytic queries, which are often the most time-consuming operations in a workload. For more information, see Working with Parallel Query for Aurora MySQL .	September 20, 2018
Aurora MySQL version 1.18.0 (p. 839)	Aurora MySQL version 1.18.0 is available.	September 20, 2018
Aurora PostgreSQL version 1.3 (p. 1013)	Aurora PostgreSQL version 1.3 is now available and is compatible with PostgreSQL 9.6.9. For more information, see Version 1.3 .	September 11, 2018
Aurora MySQL version 1.17.6 (p. 842)	Aurora MySQL version 1.17.6 is available.	September 6, 2018
New guide (p. 1013)	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*.