Getting Started with Amazon Aurora

April 2016

Notices

This document is provided for informational purposes only. It represents AWS's current product offerings and practices as of the date of issue of this document, which are subject to change without notice. Customers are responsible for making their own independent assessment of the information in this document and any use of AWS's products or services, each of which is provided "as is" without warranty of any kind, whether express or implied. This document does not create any warranties, representations, contractual commitments, conditions or assurances from AWS, its affiliates, suppliers or licensors. The responsibilities and liabilities of AWS to its customers are controlled by AWS agreements, and this document is not part of, nor does it modify, any agreement between AWS and its customers.

Contents

Abstract

Amazon Aurora is a MySQL-compatible, enterprise-grade relational database engine built for the cloud.[1] In many ways, Amazon Aurora is a game changer and helps overcome the limitations of traditional relational database engines. The goal of this whitepaper is to help you understand the benefits of Amazon Aurora and to walk you through the steps required to create and connect to your first Amazon Aurora database. This whitepaper will also cover Amazon Aurora architecture, scalability, performance, and migration paths from other databases.

Introduction

Cloud adoption among enterprises is growing quickly, with many adopting a cloud-first strategy. Relational databases like MySQL, Oracle, and Microsoft SQL Server are still a critical part of most enterprise solutions and figure prominently in the considerations while planning for an enterprise cloud migration. With respect to database migrations, the focus is changing from a "lift and shift" approach to migrating (that is, migrating as-is and running databases on virtual servers in the cloud) to fully managed, cloud-native database services like Amazon Aurora.

The primary reason for this trend is because traditional databases are not designed to take full advantage of the key benefits of the cloud like scalability, reliability, performance, and so on. Most traditional databases have been built and optimized for on-premises environments, and the fundamentals of their architecture have not changed much in the last few decades.

One of the key objectives of Amazon Aurora is to overcome the performance, scalability, and availability limitations of traditional databases in a cost-effective manner similar to open-source databases. Amazon uses service-oriented

principles and a distributed systems design to overcome many of these limitations, as detailed in the subsequent sections of this whitepaper.

Amazon Aurora: A Primer

Amazon Aurora is a MySQL 5.6–compatible database designed with the same service-oriented thinking that created AWS: decoupled architecture accessible through web services. The logging and storage layer have been moved into a high-performance, solid state drive (SSD)–based, multitenant, scale-out database-optimized storage service.

The key features of Amazon Aurora are the following:

- Highly durable – Amazon Aurora database volumes are divided into 10 GB segments; each segment is replicated six ways across three Availability Zones.
- Fault-tolerant – Amazon Aurora transparently handles the loss of up to two out of six data copies without losing write availability or three out of six copies without losing read availability.
- Self-healing – Amazon Aurora monitors disks and nodes for failures and automatically replaces or repairs the disks and nodes without the need to interrupt read or write processing from the database node.
- Storage autoscaling – Amazon Aurora will automatically grow the size of the database volume as storage needs grow. The volume will grow in increments of 10 GB up to a maximum of 64 TB.
- Continuous backup – Amazon Aurora backups are automatic, incremental, and continuous and have no impact on database performance. Automated backups are stored in Amazon Simple Storage Service (Amazon S3), which is designed for 99.999999999 percent durability.
- High performance – Amazon Aurora is designed to be compatible with MySQL 5.6, and it delivers up to five times the throughput of standard MySQL running on the same hardware.
- Read replicas – Each Amazon Aurora cluster can have up to 15 read replicas across Availability Zones to scale out read operations or act as failover targets. The replicas share the same data volume as the primary instance, and replication lag is very low, typically in the tens of milliseconds.

- Instant crash recovery – Amazon Aurora uses log-structured storage and doesn't require crash recovery replay of database redo logs, greatly reducing restart times.

- Survivable buffer cache – Amazon Aurora also isolates the database buffer cache from the database process, allowing the cache to survive a database restart.

- Highly secure – Amazon Aurora runs in a VPC based on the Amazon Virtual Private Cloud (Amazon VPC) service by default, using Secure Sockets Layer (SSL) to secure data in transit.[2] Amazon Aurora also supports encryption of data at rest.

Amazon Aurora Architecture

When you create an Amazon Aurora instance, you create a DB cluster. An Amazon Aurora DB cluster consists of the following:

- A primary instance – An instance that supports read-write workloads and performs all of the data modifications to the cluster volume. Each Amazon Aurora DB cluster has one primary instance.

- A cluster volume – An all-SSD virtual database storage volume that spans multiple Availability Zones, with each Availability Zone having two copies of the cluster data. The primary instance and any Amazon Aurora Replicas share the same cluster volume.

Additionally, you can create an Amazon Aurora Replica:

- An Aurora Replica supports only read operations, and each DB cluster can have up to 15 Aurora Replicas. Multiple Aurora Replicas distribute the read workload, and by locating Aurora Replicas in separate Availability Zones you can increase database availability. In case the primary instance fails, one of the Aurora Replicas is promoted as the primary.

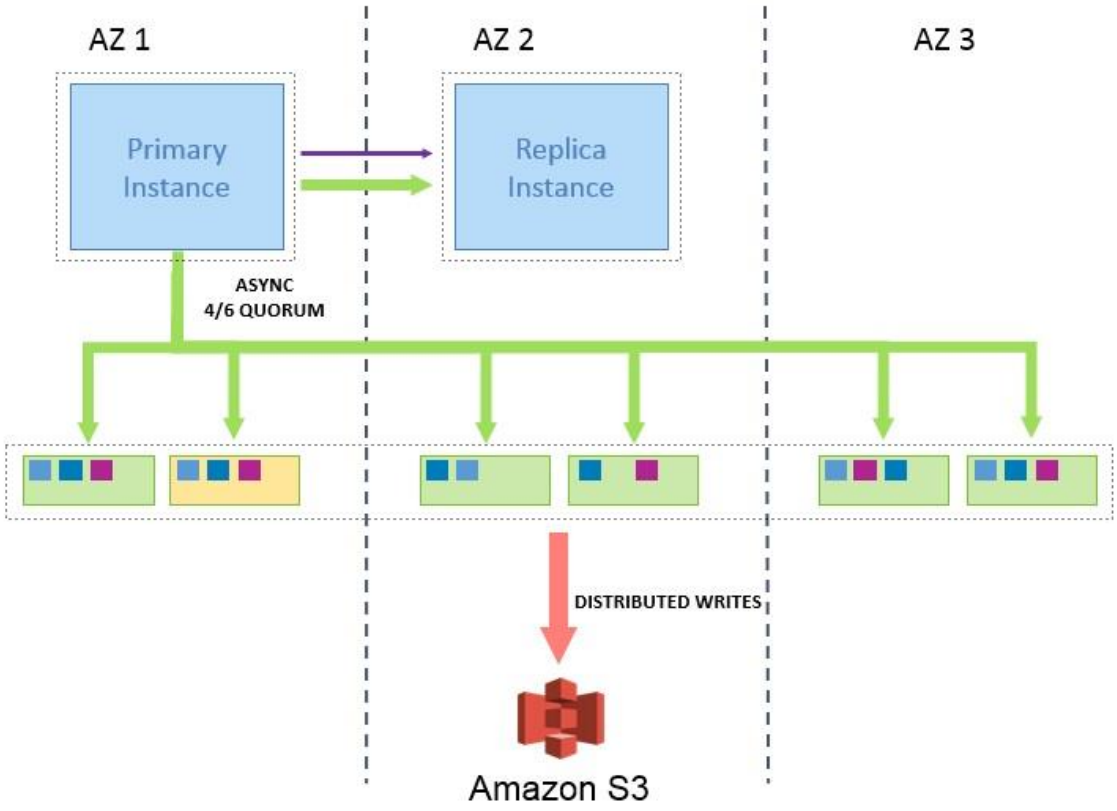


Figure 1: Amazon Aurora Architecture

Self-Healing, Fault-Tolerant Design

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

Amazon Aurora divides its database volume into 10 GB segments, each spread widely across the cluster, isolating the blast radius of disk failures. Each segment is replicated six ways across three Availability Zones. Amazon Aurora can transparently handle the loss of up to two data copies or an Availability Zone failure without losing write availability, or the loss of up to three data copies without losing read availability.

Amazon Aurora storage is also self-healing; data blocks and disks are continuously scanned for errors and replaced automatically. Amazon Aurora monitors disks and nodes for failures and automatically replaces or repairs the disks and nodes without the need to interrupt read or write processing from the database node.

Automatic, Continuous Backups

Amazon Aurora continuously backs up data to Amazon S3, which is designed for 99.999999999 percent durability. Amazon Aurora backups are automatic, incremental, and continuous and have no impact on database performance.

Amazon Aurora's backup capability enables point-in-time recovery for your instance. This functionality allows you to restore your database to any second during your retention period, up to the last 5 minutes, with only a few clicks. Your automatic backup retention period can be configured for up to 35 days. Automated backups are stored in Amazon S3.

High Performance

Amazon Aurora delivers significant increases in performance due to the use of log-structured storage and database engine modifications. The database engine is tightly integrating with a SSD-based virtualized storage layer purpose-built for database workloads, reducing write operations to the storage system, minimizing lock contention, and eliminating delays created by database process threads. Tests with SysBench on r3.8xlarge instances show that Amazon Aurora delivers over 500,000 SELECTs/second and 100,000 updates/second. Detailed instructions on this benchmark and how to replicate it yourself are provided in the Amazon Aurora Performance Benchmarking Guide.[3]

Amazon Aurora storage is organized as many small segments, each with their own redo logs. Unlike traditional databases, where the compute node must periodically checkpoint the data and flush dirty blocks from the buffers to the disk, in Amazon Aurora only the log pages are written to the storage nodes. The data pages are generated from the log pages at the storage layer, eliminating unnecessary chatter between the compute and storage nodes, enabling significantly more efficient use of network I/O.

I/O operations use distributed systems techniques such as quorums to improve performance consistency and tolerance to outliers. Data write operations are acknowledged as soon as they are committed by four out of the six storage nodes, and the individual storage nodes acknowledge the write operations as soon as the log records are persisted to disk. The storage nodes coalesce the log records into data blocks in the background asynchronously; any identified gaps are filled in using peer-to-peer gossip replication with the other storage nodes.

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

Autoscaling Storage

With Amazon Aurora, unlike with traditional databases, you don't have to provision storage space explicitly while creating the database. Amazon Aurora data is stored in a single SSD-backed virtual volume called the cluster volume that automatically grows as the amount of data in the database increases. This process is completely transparent to your application without any impact on application availability.

Amazon Aurora automatically manages the performance of the storage and can deliver consistent low-latency I/O. Additionally, Amazon Aurora manages hotspots and moves data around to ensure consistent performance of the storage layer.

Low-Latency Read Replicas

You can create up to 15 Aurora Replicas across multiple Availability Zones to serve high-volume application read traffic, thereby increasing aggregate read throughput. Aurora Replicas share the same underlying storage as the source instance, lowering costs and avoiding the need to replay logs at the replica nodes. This approach frees up more processing power to serve read requests and reduces the replica lag time—often down to single-digit milliseconds.

Failure Testing

With a traditional database, testing and validating your database cluster's ability to handle node, disk, and networking failures can be expensive, tedious, and time-consuming. However, with Amazon Aurora you can test the fault tolerance of your Amazon Aurora DB cluster by using simple 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 the master instance or an Aurora Replica
- A failure of an Aurora Replica
- A disk failure
- Disk congestion

Here is an example of a fault injection query to simulate a crash:

```
ALTER SYSTEM CRASH [INSTANCE | DISPATCHER | NODE];
```

Fault injection queries with the `CRASH` option, like the one preceding, force a crash of the Amazon Aurora instance. Other fault injection queries result in simulations of failure events, but don't cause the event to occur. When you submit a fault injection query, you also specify an amount of time for the failure event simulation to occur. You can also submit a fault injection query to one of your Aurora Replica instances by connecting to the endpoint for the Aurora Replica.

Multiple Failover Targets

Amazon Aurora uses automated checks to detect failure of the primary database node in a cluster. Unlike other database engines, Amazon Aurora does not require a separate standby instance to fail over. If a primary node fails, Amazon Aurora will automatically fail over to any of up to 15 Aurora Replicas with minimal downtime and availability impact on applications. These read replicas can be in any of the three Availability Zones. For high availability, we recommend placing at least one read replica in an alternate Availability Zone. Failover happens with no data loss, and log replay is not required, because the replicas and the primary instance share the same storage.

Instant Crash Recovery

Amazon Aurora is designed to recover from a crash almost instantaneously and continue to serve your application data. Unlike other databases, after a crash Amazon Aurora does not need to replay the redo log from the last database checkpoint before making the database available for operations.

Amazon Aurora performs crash recovery asynchronously on parallel threads, so your database is open and available immediately after a crash. Because the storage is organized in many small segments, each with its own redo log, the underlying storage can replay redo records on demand in parallel and asynchronously as part of a disk read after a crash. This approach reduces database restart times to less than 60 seconds in most cases.

Survivable Caches

In Amazon Aurora, the database buffer cache has been moved out of the database process. If a database restarts, the cache remains warm, and performance is not impacted due to a cold cache as is the case with traditional databases. This approach lets you resume fully loaded operations much faster.

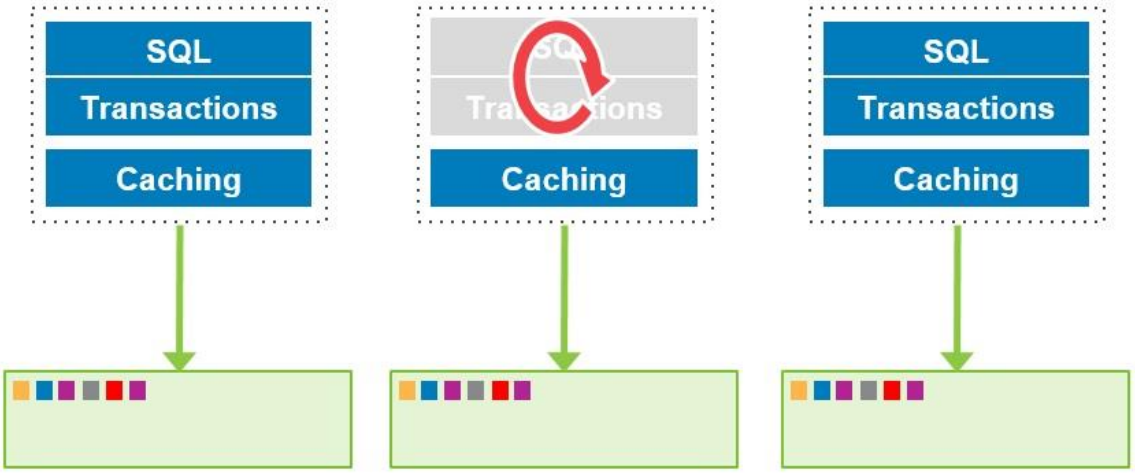


Figure 2: Amazon Aurora Cache Architecture

Security

Amazon Aurora DB instances must be created in a VPC based on Amazon VPC. Amazon VPC lets you provision a logically isolated section of the Amazon Web Services (AWS) cloud where you can launch AWS resources in a virtual network that you define. You have complete control over your virtual networking environment, including selection of your own IP address range, creation of subnets, and configuration of route tables and network gateways. You can leverage multiple layers of security, including security groups and network access control lists, to help control access to your instances in each subnet. This approach gives you complete control over who can access your Amazon Aurora database.

Amazon Aurora DB supports Secure Sockets Layer (SSL) connections from applications, using SSL (AES-256) to secure data in transit. Amazon Aurora also supports encryption of data at rest. Data is encrypted using AES-256 with hardware acceleration support. All the blocks on disk and the backups on Amazon S3 are encrypted. Encryption keys are managed by AWS Key Management Service (AWS KMS), which is a highly available, durable, and secure solution for managing sensitive encryption keys.[4]

Getting Started

The first step in getting started with Amazon Aurora is creating a database.

Creating an Amazon Aurora Database

The AWS Management Console is the easiest way to create your first Amazon Aurora cluster. Log in to the AWS management console, and navigate to the Amazon Relational Database (Amazon RDS) section. From the top-right corner, choose the AWS Region in which you want to create the Amazon Aurora cluster, and then choose **Get Started Now**. On the next screen, choose **Amazon Aurora**, and then choose **Select** as shown following.

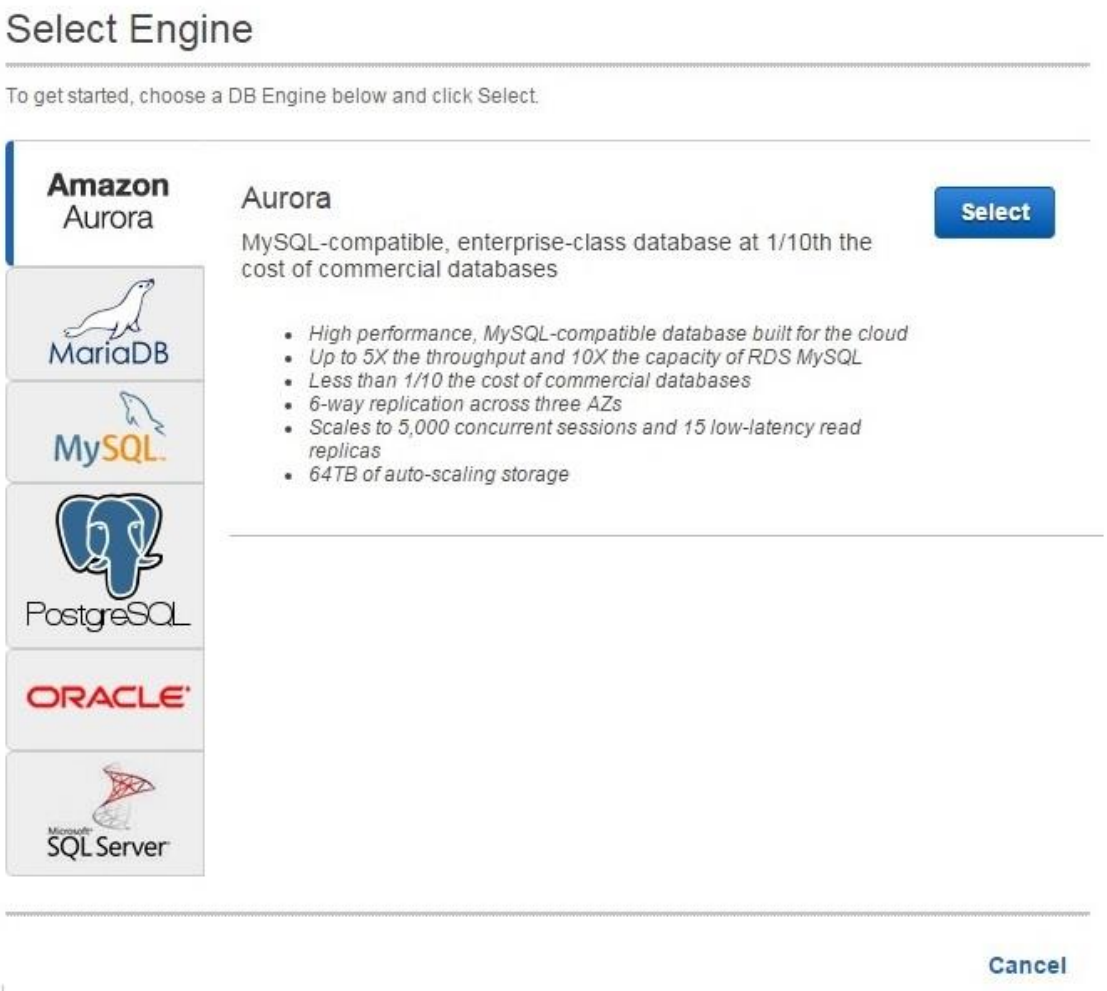


Figure 3: Amazon RDS Engine Selection Wizard in the Console

Next, on the DB details page, you select the size for your Amazon Aurora primary instance and provide other details like the database identifier, master user name, and password, as shown following. You then choose **Next Step**.

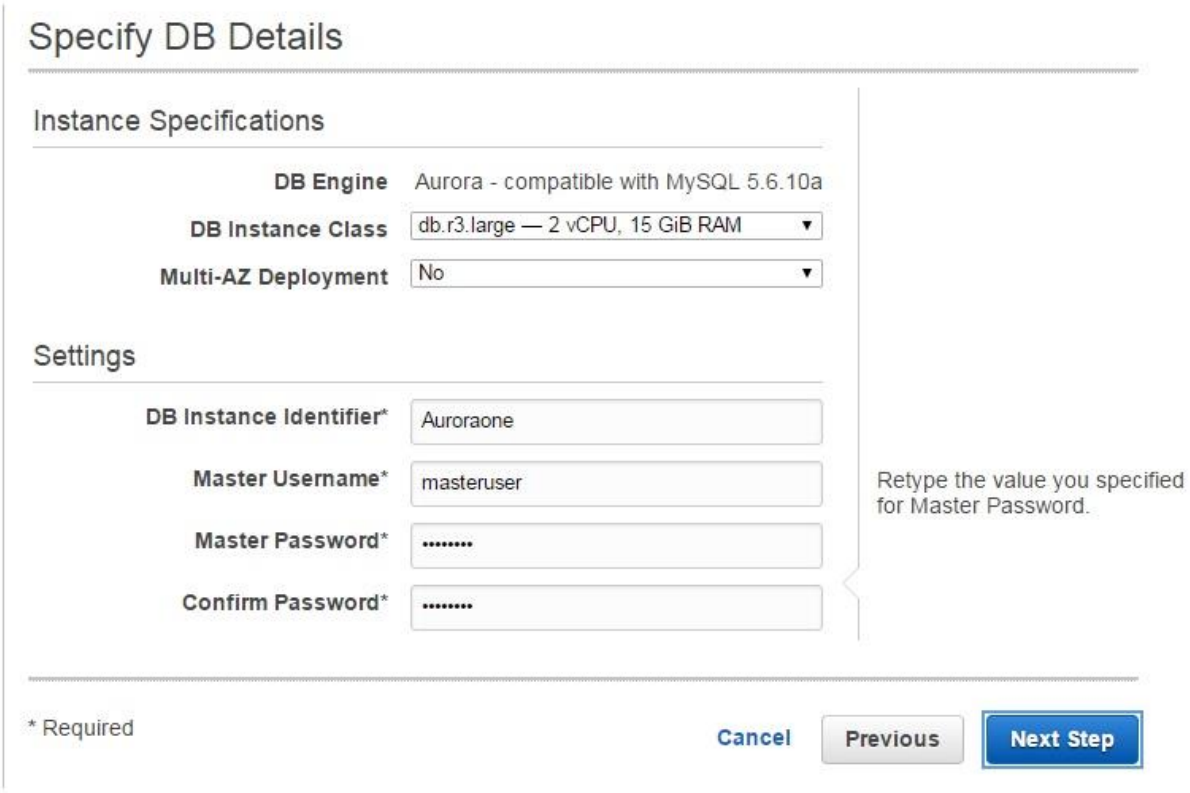


Figure 4: Specifying DB Details

On the next screen, you can customize additional settings for your Amazon Aurora database cluster like VPC selection, database name, port number, and so on. Accept the default settings, and choose **Launch DB Instance**. Your Amazon Aurora instance will launch in a few minutes.

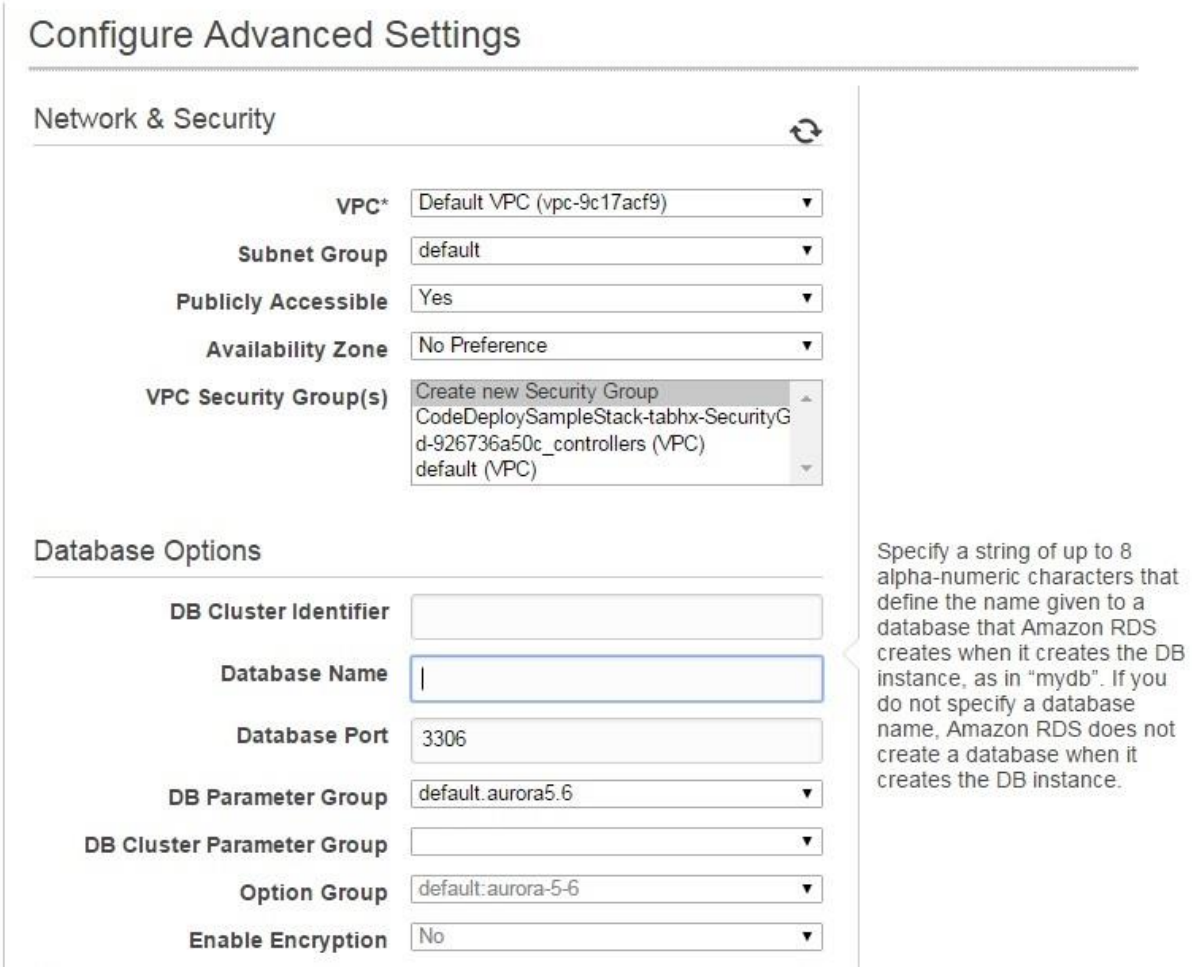


Figure 5: Advanced Settings

Connecting to Your Amazon Aurora Database

Navigate to the **Instances** tab in the RDS dashboard, and you will see your Amazon Aurora instance listed there. You can see the details of your Amazon Aurora cluster like the cluster endpoint and the port number, as shown following.

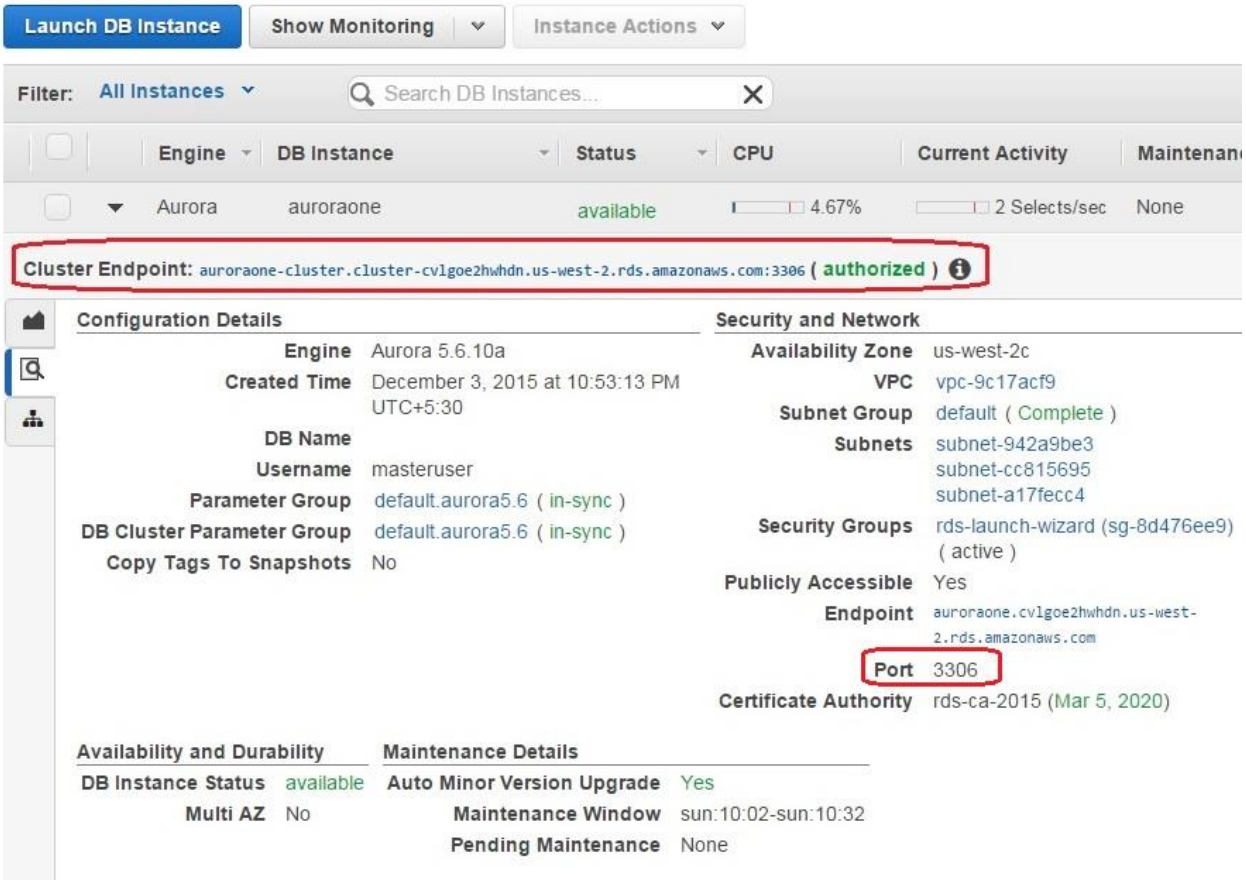


Figure 6: Amazon Aurora Cluster Details

Use the endpoint and port number in your JDBC and ODBC connection strings to connect from your application or standard tools. You can use your favorite tools like MySQL Workbench, Navicat, Webyog, Toad, or Oracle SQL Developer to connect and work with your Amazon Aurora database.

Instance Sizing

Amazon Aurora instances are available in various sizes, starting from the db.r3.large instance with 2 vCPUs and 15 GiB RAM, to the db.r3.8xlarge instance with 32 vCPUs and 244 GiB RAM. The complete list of Amazon Aurora instance types is available on the Aurora pricing page.[5]

You choose an appropriate instance type based on the RAM, vCPU, and network throughput required. You can start with a smaller instance type like db.r3.large or db.r3.xlarge and upgrade to a larger instance type as your application grows. With Amazon Aurora, it's very simple to scale up or down; you just stop the

instance and restart it on a larger or smaller instance type. Compute scaling operations typically complete in a few minutes.

Scalability

With Amazon Aurora, you have the option of scaling up using a larger instance type or scaling out using Aurora Replicas.

Scaling Up

As demand for your application grows, you can scale up your Amazon Aurora instances by upgrading to a larger instance type. For example, you can start with a db.r3.large instance type with 2 vCPUs and 15 GiB RAM and scale up all the way to a db.r3.8xlarge instance type with 32 vCPUs and 244 GiB RAM.

The following steps illustrate how easy it is to scale up an Amazon Aurora instance from db.r3.2xlarge to db.r3.4xlarge:

1. Select the instance, and from **Instance Actions**, choose **Modify**, as shown following.

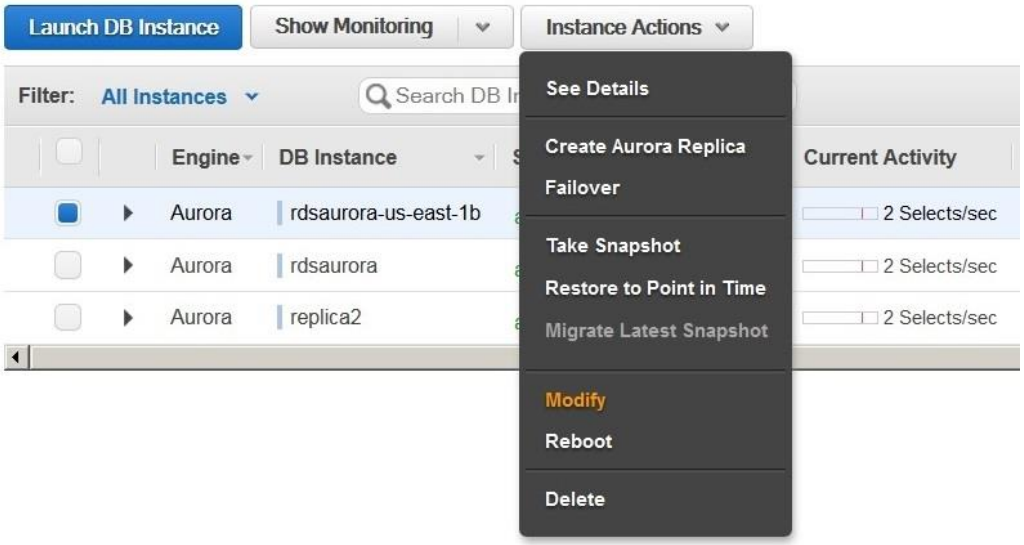


Figure 7: Scaling Up an Amazon Aurora Instance

2. Select the new instance type, for example db.r3.4xlarge. Choose **Apply Immediately** if you want to make the changes right now, or else the changes will be made during the next maintenance window. Now choose **Continue**.

3. Review the modifications on the next screen, and then choose **Modify DB Instance**. The status will change to **modifying**, and within a few minutes the instance type will be changed and your Amazon Aurora database will be available again.

Scaling Out

You can scale out an Amazon Aurora database by adding up to 15 Aurora Replicas, which are read-only. You can configure your applications to send read/write traffic to the primary instance and read-only traffic to the Aurora Replicas.

The following steps illustrate how to scale out an Aurora database by adding Aurora Replicas:

1. Select your instance, and from **Instance Actions**, choose **Create Aurora Replica**, as shown following.

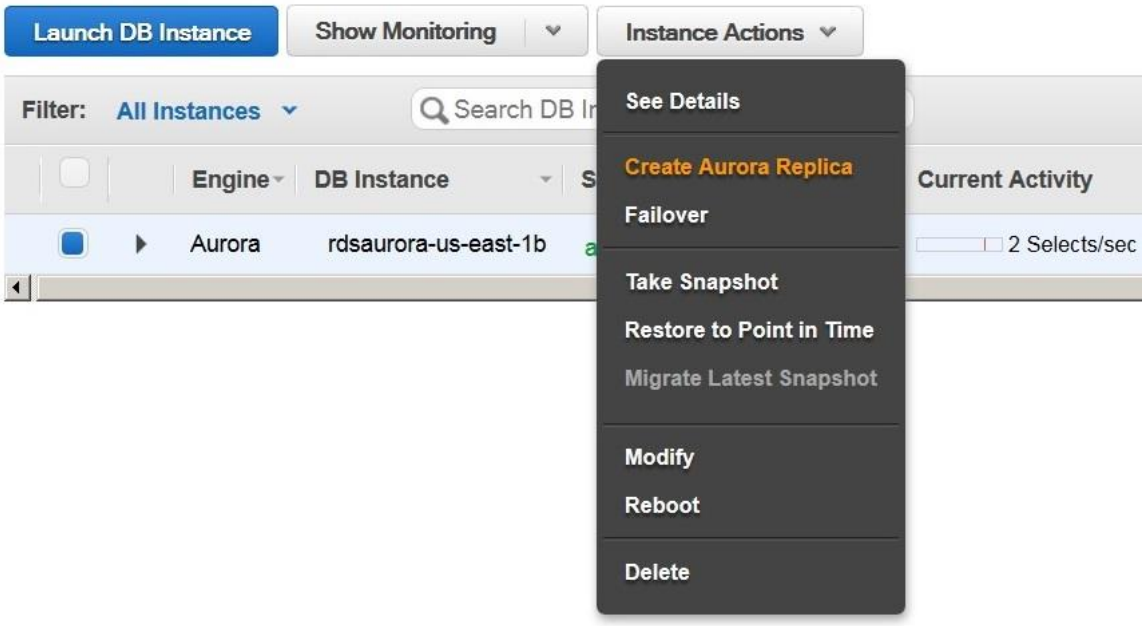


Figure 8: Creating an Aurora Replica

2. Select the instance type, type a value for **DB Instance Identifier**, and choose **Create Aurora Replica**. The Aurora Replica will be created within a few minutes.

Backup and Restore

Amazon Aurora backs up your cluster volume automatically and retains backup data for the length of the backup retention period (between 1 to 35 days). Amazon Aurora's backup capability enables point-in-time recovery of your instance to any second during your retention period, up to the last 5 minutes, with just a few clicks.

If you want to retain a backup beyond the maximum retention period, you can take a snapshot of the database. DB snapshots are user-initiated backups of your instance that are kept until you explicitly delete them.

Backups are stored in Amazon S3, which is designed for 99.999999999 percent durability.[6] Backups are automatic, incremental, and continuous and have no impact on database performance. No interruption of database service occurs as backup data is being written.

To restore your data, you can create a new instance quickly from the backup Amazon Aurora maintains or from a DB snapshot. When you request a restore of your DB cluster, the new cluster volume is immediately available for both read and write operations. However, as the copy is being created, you might encounter some latency for read operations. This latency will only occur if a query requests data that has not yet been restored to the cluster volume. In that case, the data will be immediately restored to the cluster volume and returned to the query request.

Managing Amazon Aurora

You can access and manage your Amazon Aurora cluster in several ways. When you are getting started with Aurora, the easiest and most popular way is to use the AWS Management Console.

In addition to using the AWS Management Console, you can manage Amazon Aurora using the RDS Command Line Interface (CLI), or you can programmatically interact with and manage your Amazon Aurora cluster using the AWS SDKs and libraries. AWS SDKs and libraries are available for many popular languages like Java, PHP, Python, Ruby, and .NET.

Monitoring

Amazon RDS provides metrics to monitor the health of your DB instances and DB clusters. These metrics include DB instance metrics and also operating system (OS) metrics. You can monitor RDS using Amazon CloudWatch and Enhanced Monitoring. CloudWatch gathers metrics about CPU utilization from the hypervisor for a DB instance. Enhanced Monitoring gathers its metrics from a lightweight agent on the instance. Enhanced Monitoring metrics are useful when you want to see how different processes or threads on a DB instance use the CPU.

Amazon CloudWatch Monitoring

You can monitor your Amazon Aurora instance using Amazon CloudWatch metrics at no additional charge.[7] You can use the AWS Management Console to view over 20 key operational metrics for your DB instances, including compute, memory, storage, network throughput, and replica lag.[8]

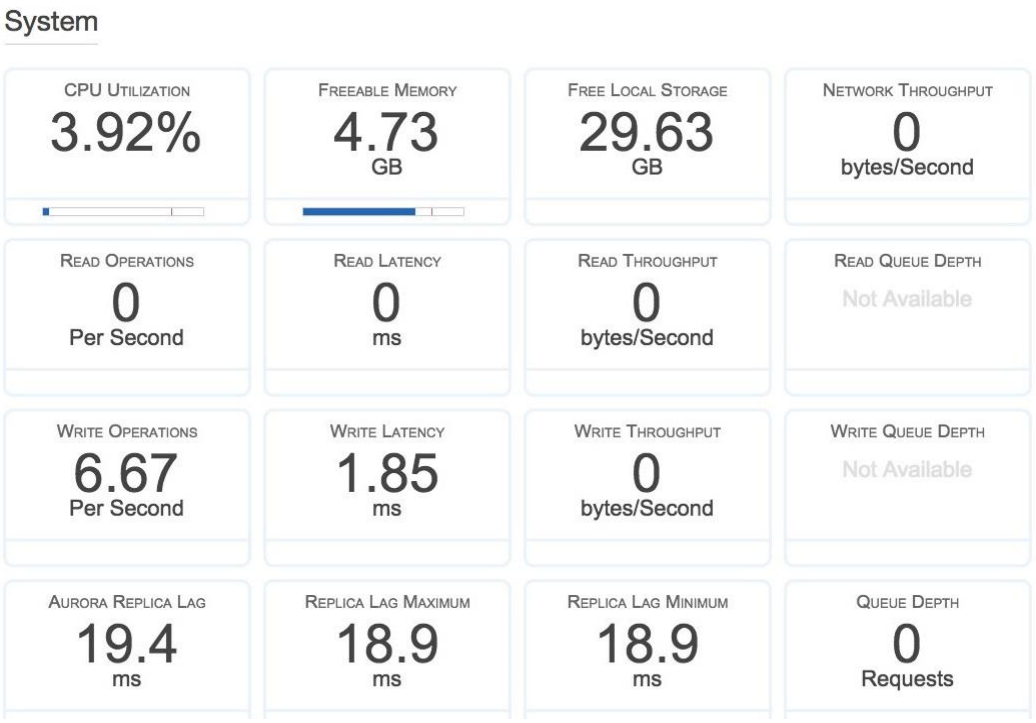


Figure 9: Amazon Aurora CloudWatch Metrics—System

In addition to these, you also have access to the SQL-related metrics like average query throughput, latency, logins, transactions, and so on, as shown following.

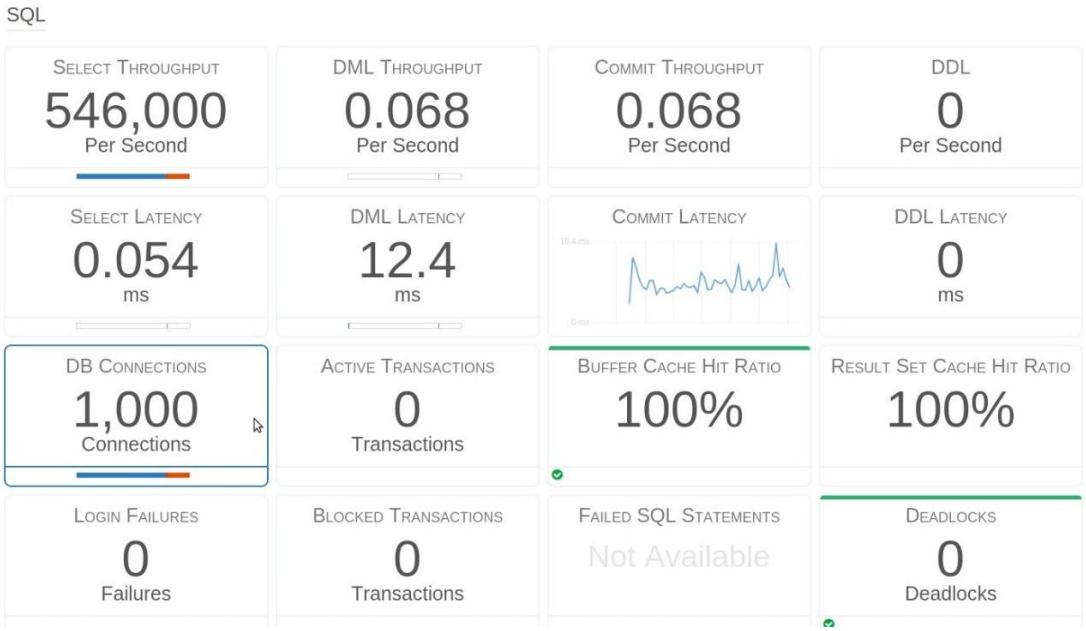


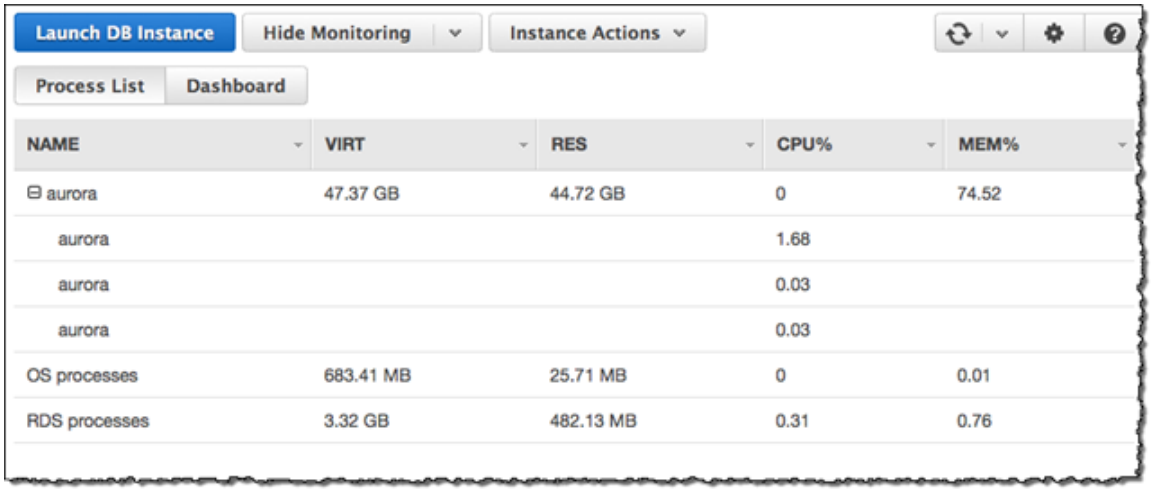
Figure 10: Amazon Aurora CloudWatch Metrics—SQL Operations

Enhanced Monitoring

Enhanced Monitoring gives you access to over 50 metrics, including CPU, memory, file system, and disk I/O. You can enable this monitoring for each of your instances, and you can choose the data granularity, all the way down to 1 second. You can also enable Enhanced Monitoring for an existing instance. Setting it up doesn't require your DB instance to restart.

You can view OS metrics reported by Enhanced Monitoring in the RDS console; two views are available. The Dashboard view shows graphs of the OS metrics, and the Process List view shows the processes running on the DB instance and their related metrics, including percentage of CPU usage and memory usage.

Figure 11: Amazon Aurora Enhanced Monitoring—Dashboard View



NAME	VIRT	RES	CPU%	MEM%
⊟ aurora	47.37 GB	44.72 GB	0	74.52
aurora			1.68	
aurora			0.03	
aurora			0.03	
OS processes	683.41 MB	25.71 MB	0	0.01
RDS processes	3.32 GB	482.13 MB	0.31	0.76

Figure 12: Amazon Aurora Enhanced Monitoring—Process View

Migrating to Amazon Aurora

The following section discusses different migrations and different approaches to migration to Amazon Aurora.

Amazon RDS MySQL to Amazon Aurora

You can migrate a database snapshot of an Amazon RDS MySQL instance to create an Amazon Aurora DB cluster. The new Aurora cluster will be populated with the data from the original Amazon RDS MySQL database. The database snapshot must have been made from an Amazon RDS instance running MySQL 5.6.

You can migrate either a manual or automated database snapshot. After the Aurora cluster is created, you can optionally create Aurora Replicas.

MySQL to Amazon Aurora

If you want to migrate from a MySQL instance running externally on Amazon Elastic Compute Cloud (Amazon EC2) or on-premises, and your database size is small or service interruption on the source MySQL database isn't an issue, you can use the mysqldump command-line utility. The mysqldump command-line utility is commonly used to make backups and transfer data from one MySQL database to another. In this case, you can copy the database with mysqldump and pipe it directly into the Amazon Aurora instance.

Amazon Aurora is compatible with MySQL, and you can set up binary log (binlog) replication between a MySQL database and an Amazon Aurora DB cluster. You can take this approach to migrate a MySQL database from on-premises or on EC2 to Amazon Aurora with reduced downtime. You can find more details on configuring binlog replication in the Aurora documentation.[9] You can also find additional information on migrating to Amazon Aurora in the documentation topic Migrating Data to an Amazon Aurora DB Cluster.[10]

Migration with Minimal Downtime

For migration with minimal downtime or service interruption, AWS Database Migration Service (AWS DMS) is the most appropriate choice.[11] With AWS DMS, the source database remains fully operational during the migration, minimizing downtime to applications that rely on the database. The change data capture capability of AWS DMS can continuously capture and apply all data changes from the source database to the target, after the migration process begins. AWS manages all the complexities of the migration process like compression and parallel transfer for faster data transfer.

AWS DMS is low-cost and simple to use. You only pay for the compute resources used during the migration process; you can migrate a terabyte-sized database for as little as $3.

The service supports homogenous migrations such as Oracle to Oracle, and also heterogeneous migrations between different database platforms, such as Oracle to Aurora or Microsoft SQL Server to MySQL. To help make heterogeneous migrations easier, AWS DMS includes the AWS Schema Conversion Tool. This tool converts the source database schema and objects, including views, stored procedures, and functions, to a format compatible with the target database. Any code that cannot be automatically converted gets clearly marked to help you identify where manual recoding is required.

Conclusion

Amazon Aurora is a high performance, highly available, enterprise-grade database built for the cloud. Amazon Aurora is offered as a managed service without the common and time-consuming administrative tasks associated with managing a database on your own, freeing you to focus on your applications and business. An Amazon Aurora cluster can be created in just a few clicks and is extremely easy to use, manage, and scale.

With Amazon Aurora, there are no license fees or up-front commitment. You simply pay an hourly charge for each Amazon Aurora cluster that you create, and when you're finished you simply delete the cluster and stop paying. You can scale your Amazon Aurora cluster up or down as required with just a few clicks. Your Amazon Aurora primary instance can scale up to 32 vCPUs and 244 GiB memory, and you can add up to 15 Amazon Aurora Replicas each, with up to 32 vCPUs and 244 GiB memory to further scale read capacity. Amazon Aurora storage automatically grows as needed from 10 GB to 64 TB; there is no need to provision or manage storage.

Contributors

The following individuals and organizations contributed to this document:

- Tom Laszewski, Senior Manager—Solutions Architects, Amazon Web Services
- Kamal Arora, Solutions Architect, Amazon Web Services
- Ashok Sundaram, Solutions Architect, Amazon Web Services

Further Reading

For additional help, consult the following sources:

- Amazon Aurora Product Details
- Amazon Aurora FAQ
- Amazon Aurora Documentation
- Amazon Aurora Performance Benchmarking Guide

Notes

[1] https://aws.amazon.com/rds/aurora/

[2] https://aws.amazon.com/vpc/

[3] https://d0.awsstatic.com/product-marketing/Aurora/RDS_Aurora_Performance_Assessment_Benchmarking_v1-2.pdf

[4] https://aws.amazon.com/kms/

[5] http://aws.amazon.com/rds/aurora/pricing/

[6] http://aws.amazon.com/s3/

[7] https://aws.amazon.com/cloudwatch/

[8] http://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/Aurora.Monitoring.html#Aurora.Monitoring.Metrics

[9] http://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/Aurora.Replication.html#Aurora.Overview.Replication.MySQLReplication

[10] http://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/Aurora.Migrate.html

[11] https://aws.amazon.com/dms/