AWS® Certified Cloud Practitioner

STUDY GUIDE

FOUNDATIONAL (CLF-C01) EXAM

Includes interactive online learning environment and study tools:

Two custom practice exams

100 electronic flashcards

Searchable key term glossary

BEN PIPER
DAVID CLINTON

SYBEX®
A Wiley Brand

define in advance is a primary key to uniquely identify each item. For example, to store customer data in a table, you might use a unique customer ID number as the primary key.

In exchange for the flexibility of storing unstructured data, the types of queries you can perform against that data are more limited. Nonrelational databases are designed to let you query items based on the primary key. Because the rest of the data doesn't follow a predictable structure, a particular piece of data could be anywhere in the database. Hence, trying to query against any other data requires searching through every item in the entire table—a process that gets slower as the table grows. Nonrelational databases are best suited for applications that need to perform just a few well-defined queries.

Amazon Relational Database Service

The Amazon Relational Database Service (RDS) is Amazon's managed relational database service. RDS lets you provision a number of popular relational database management systems (RDBMSs) including Microsoft SQL Server, Oracle, MySQL, and PostgreSQL.

You can always install and configure your own database server on an EC2 instance. But RDS offers several advantages over this. When you create an RDS database instance, Amazon sets up one or more compute instances and takes care of installing and configuring the RDBMS of your choice. These compute instances are not EC2 instances that you can secure shell (SSH) into, but they are connected to a virtual private cloud (VPC) of your choice, allowing your applications running on AWS or on-premises to take full advantage of an RDS-hosted database. Like EC2 instances, RDS instances use Elastic Block Service (EBS) volumes for storage.

To achieve the level of performance and availability you need, you can choose a multi-Availability Zone (multi-AZ) deployment to have database instances in multiple Availability Zones. RDS can also perform manual or automatic EBS snapshots that you can easily restore to new RDS instances. RDS can also handle the hard work of installing patches and upgrades during scheduled maintenance windows.

Database Engines

When you create an RDS instance, you must choose a database engine, which is the specific RDBMS that will be installed on your instance. You can have only one database engine per instance, but you can provision multiple instances if need be. Amazon RDS supports the following six database engines:

- MySQL
- MariaDB
- Oracle
- PostgreSQL
- Microsoft SQL Server
- Amazon Aurora

With the exception of Amazon Aurora, these database engines are either open source or commercially available products found in many data center environments. Amazon Aurora is a proprietary database designed for RDS, but it's compatible with existing MySQL and PostgreSQL databases. Being able to use RDS to deploy an RDBMS that you're already familiar with makes migrating such databases from on-premises to RDS much easier.

Licensing

Depending on the database engine you choose, you must choose one of two licensing options: *license included* or *bring your own license* (BYOL):

License included The license is included in the pricing for each RDS instance. The Microsoft SQL Server and Oracle database engine options offer this license model. The free database engines—MariaDB, MySQL, and PostgreSQL—exclusively use the license included model.

Bring your own license In this model, you must provide your own license to operate the database engine you choose. Unlike the license-included option, licensing costs are not built into RDS pricing. This model is currently available only for Oracle databases.

Instance Classes

Implementing a relational database—even one backed by RDS—requires some capacity planning to ensure the database gives you the level of availability and performance your application needs.

When you deploy an RDS instance, you must choose a database instance class that defines the number of virtual CPUs (vCPU), the amount of memory, and the maximum network and storage throughput the instance can support. There are three instances classes you can choose from: Standard, Memory Optimized, and Burstable Performance.

Standard

The Standard instance class will meet the requirements of most applications. The latest-generation Standard instance class offers the following specs:

- Between 2 and 96 vCPU
- 8–384 GB memory

Memory Optimized

The Memory Optimized instance class is for applications with the most demanding database requirements. This class offers the most disk throughput and network bandwidth. The latest-generation instance class provides the following:

- Between 4 and 128 vCPU
- 122–3,904 GB memory

Database instances use EBS storage. Both the Standard and Memory Optimized instance class types are EBS-optimized, meaning they provide dedicated bandwidth for transfers to and from EBS storage.

Burstable Performance

The Burstable Performance instance class is for nonproduction databases that have minimal performance requirements, such as those for test and development purposes. The latest-generation Burstable Performance instance class has the lowest network bandwidth and disk throughput and offers the following:

- Between 2 and 8 vCPU
- 1–32 GB memory

It can be difficult to predict exactly how many RDS instances you need and how much compute power, memory, and network and storage throughput each of those instances needs. Thankfully, RDS makes it easy to right-size your database deployments in two ways: scaling vertically and scaling horizontally.

Scaling Vertically

Scaling vertically refers to changing the way resources are allocated to a specific instance. After creating an instance, you can scale up to a more powerful instance class to add more memory or improve computing or networking performance. Or you can scale down to a less powerful class to save on costs.

Storage

The level of performance an RDS instance can achieve depends not only on the instance class you choose but also on the type of storage. New RDS instances use EBS volumes, and the maximum throughput a volume can achieve is a function of both the instance class and the number of *input/output operations per second* (IOPS) the EBS volume supports. IOPS measure how fast you can read from and write to a volume. Higher IOPS generally means faster reads and writes. RDS offers three types of storage: general-purpose SSD, provisioned IOPS SSD, and magnetic.

General-Purpose SSD

General-purpose SSD storage is good enough for most databases. You can allocate a volume of between 20 GB and 32 TB. The number of IOPS per volume depends on how much storage you allocate. The more storage you allocate, the better your read and write performance will be.

If you're not sure how much storage to provision, don't worry. General-purpose SSD volumes can temporarily achieve a higher number of IOPS through a process called *bursting*. During spikes of heavy read or write activity, bursting will kick in automatically

and give your volume an added performance boost. This way, you don't have to allocate an excessive amount of storage just to get enough IOPS to meet peak demand.

Provisioned IOPS SSD

Provisioned IOPS SSD storage allows you to specify the exact number of IOPS (in thousands) that you want to allocate per volume. Like general-purpose SSD storage, you can allocate up to 32 TB. But unlike general-purpose SSD storage, provisioned IOPS SSD storage doesn't offer bursting, so it's necessary to decide beforehand the maximum number of IOPS you'll need. However, even if your needs change, you can always adjust the number of IOPS later.

Magnetic

Magnetic storage is available for backward compatibility with legacy RDS instances. Unlike the other storage options, it doesn't use EBS, and you can't change the size of a magnetic volume after you create it. Magnetic volumes are limited to 4 TB in size and 1,000 IOPS.

You can increase the size of an EBS volume after creating it without causing an outage or degrading performance. You can't, however, decrease the amount of storage allocated, so be careful not to go overboard.

You can also migrate from one storage type to another, but doing so can result in a short outage of typically a few minutes. But when migrating from magnetic to EBS storage, the process can take up to several days. During this time, the instance is still usable but may not perform optimally.

Scaling Horizontally with Read Replicas

In addition to scaling up by choosing a more powerful instance type or selecting high-IOPS storage, you can improve the performance of a database-backed application by adding additional RDS instances that perform only reads from the database. These instances are called *read replicas*.

In a relational database, only the master database instance can write to the database. A read replica helps with performance by removing the burden of read-only queries from the master instance, freeing it up to focus on writes. Hence, read replicas provide the biggest benefit for applications that need to perform a high number of reads. Read replicas are also useful for running computationally intensive queries, such as monthly or quarterly reports that require reading and processing large amounts of data from the database.

High Availability with Multi-AZ

Even if you use read replicas, only the master database instance can perform writes against your database. If that instance goes down, your database-backed application won't be able to write data until it comes back online. To ensure that you always have a master database instance up and running, you can configure high availability by enabling the multi-AZ feature on your RDS instance.

With multi-AZ enabled, RDS creates an additional instance called a *standby database instance* that runs in a different Availability Zone than your primary database instance. The primary instance instantly or synchronously replicates data to the secondary instance, ensuring that every time your application writes to the database, that data exists in multiple Availability Zones.

If the primary fails, RDS will automatically fail over to the secondary. The failover can result in an outage of up to two minutes, so your application will experience some interruption, but you won't lose any data.

With multi-AZ enabled, you can expect your database to achieve a monthly availability of 99.95 percent. It's important to understand that an instance outage may occur for reasons other than an Availability Zone outage. Routine maintenance tasks such as patching or upgrading the instance can result in a short outage and trigger a failover.

If you use the Amazon Aurora database engine—Amazon's proprietary database engine designed for and available exclusively with RDS—you can take advantage of additional benefits when using multi-AZ. When you use Aurora, your RDS instances are part of an Aurora cluster. All instances in the cluster use a shared storage volume that's synchronously replicated across three different Availability Zones. Also, if your storage needs increase, the cluster volume will automatically expand up to 64 TB.

Backup and Recovery

Whether or not you use multi-AZ, RDS can take manual or automatic EBS snapshots of your instances. Snapshots are stored across multiple Availability Zones. If you ever need to restore from a snapshot, RDS will restore it to a new instance. This makes snapshots useful not only for backups but also for creating copies of a database for testing or development purposes.

You can take a manual snapshot at any time. You can configure automatic snapshots to occur daily during a 30-minute backup window. RDS will retain automatic snapshots between 1 day and 35 days, with a default of 7 days. Manual snapshots are retained until you delete them.

Enabling automatic snapshots also enables point-in-time recovery, a feature that saves your database change logs every 5 minutes. Combined with automated snapshots, this gives you the ability to restore a failed instance to within 5 minutes before the failure—losing no more than 5 minutes of data.

Determining Your Recovery Point Objective

Do you need snapshots *and* multi-AZ? It's important to understand that although both snapshots and multi-AZ protect your databases, they serve slightly different purposes. Snapshots are good for letting you restore an entire database instance. If your database encounters corruption, such as malicious deletion of records, snapshots let you recover that data, even if the corruption occurred days ago (provided you're retaining the snapshots). Multi-AZ is designed to keep your database up and running in the event of an instance failure. To achieve this, data is synchronously replicated to a secondary instance.

How much data loss you can sustain in the event of a failure is called the *recovery point objective* (RPO). If you can tolerate losing an hour's worth of data, then your RPO would be 1 hour. To achieve such an RPO, simply using automatic snapshots with point-in-time recovery is sufficient. For an RPO of less than 5 minutes, you would also want to use multi-AZ to synchronously replicate your data to a secondary instance.

DynamoDB

DynamoDB is Amazon's managed nonrelational database service. It's designed for highly transactional applications that need to read from or write to a database tens of thousands of times a second.

Items and Tables

The basic unit of organization in DynamoDB is an item, which is analogous to a row or record in a relational database. DynamoDB stores items in tables. Each DynamoDB table is stored across one or more partitions. Each partition is backed by solid-state drives, and partitions are replicated across multiple Availability Zones in a region, giving you a monthly availability of 99.99 percent.

Each item must have a unique value for the primary key. An item can also consist of other key-value pairs called *attributes*. Each item can store up to 400 KB of data, more than enough to fill a book! To understand this better, consider the sample shown in Table 9.2.

TABLE 9.2 A Sample DynamoDB Table

Username (Primary Key)	LastName	FirstName	FavoriteColor
hburger	Burger	Hamilton	
dstreet	Street	Della	Fuchsia
pdrake	Drake	Paul	Silver
perry		Perry	

Username, LastName, FirstName, and FavoriteColor are all attributes. In this table, the Username attribute is the primary key. Each item must have a value for the primary key, and it must be unique within the table. Good candidates for primary keys are things that tend to be unique, such as randomly generated identifiers, usernames, and email addresses.

Exam Essentials

Understand the major differences between relational and nonrelational databases. Relational databases are designed for structured data that contains a defined number of attributes per record. They let you perform complex queries against a variety of dimensions, making them ideal for reporting and analytics. Nonrelational databases are designed for data that doesn't follow a predictable structure. Each item in a nonrelational database must have a primary key, and you can query based on that key.

Know the vertical and horizontal scaling options for RDS. You can scale an RDS instance vertically by upgrading to a larger instance class to give it more processing power, memory, or disk or network throughput. You can also select provisioned IOPS SSD storage to ensure your instance always achieves the storage performance it needs. For horizontal scaling of reads, your only option is to use read replicas.

Be able to describe the components of RDS. An RDS deployment consists of at least one instance. You must select an instance class that defines the vCPUs and memory for the instance. You must also select a database engine. For storage, you must select general-purpose or provisioned IOPS SSD. Magnetic storage is a legacy option that's not available for new deployments. You can also add read replicas to scale horizontally to improve read performance. In a multi-AZ deployment, you can add additional secondary instances that the primary synchronously replicates data to.

Know the backup and recovery options for RDS. You can schedule automatic snapshots for your RDS instance to occur daily during a 30-minute backup window of your choice. Backups are retained between 1 day and 35 days. Enabling automatic backups also enables point-in-time recovery, allowing the restoration of a failed database up to 5 minutes prior to failure. Restoring from a snapshot entails creating a new instance from the snapshot. You can also take a manual snapshot at any time.

Understand how DynamoDB stores data. DynamoDB stores data as items in tables. Each item must have primary key whose values are unique within the table. This is how DynamoDB uniquely identifies an item. The primary key's name and data type must be defined when the table is created. When you create an item, you can also add other attributes in addition to the primary key. DynamoDB uses the primary key to distribute items across different partitions. The number of partitions allocated to a table depends on the number of WCU and RCU you configure.

Be able to identify scenarios for using Redshift. Redshift is a data-warehousing service for storing and analyzing structured data from multiple sources, including relational databases and S3. Redshift can store much more data than RDS, up to 2 PB!

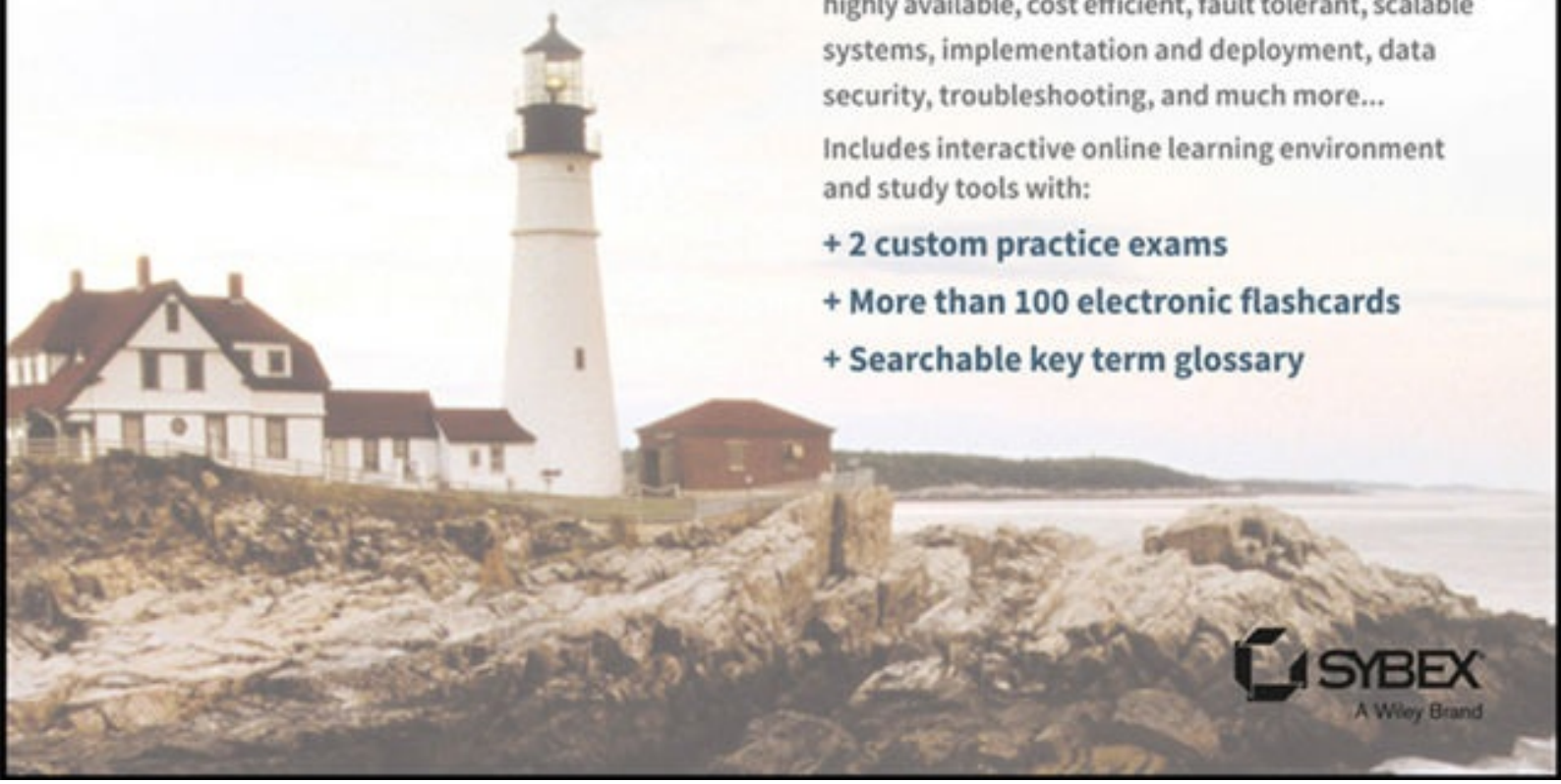
amazon web services

Joe Baron, Hisham Baz, Tim Bixler, Biff Gaut,
Kevin E. Kelly, Sean Senior, John Stamper

AWS Certified Solutions Architect

OFFICIAL STUDY GUIDE

ASSOCIATE EXAM



Covers exam objectives, including designing highly available, cost efficient, fault tolerant, scalable systems, implementation and deployment, data security, troubleshooting, and much more...

Includes interactive online learning environment and study tools with:

+ 2 custom practice exams
+ More than 100 electronic flashcards
+ Searchable key term glossary

SYBEX
A Wiley Brand

availability and durability. By delivering consistent and low-latency performance, Amazon EBS provides the disk storage needed to run a wide variety of workloads.

AWS Storage Gateway

AWS Storage Gateway is a service connecting an on-premises software appliance with cloud-based storage to provide seamless and secure integration between an organization's on-premises IT environment and the AWS storage infrastructure. The service supports industry-standard storage protocols that work with existing applications. It provides low-latency performance by maintaining a cache of frequently accessed data on-premises while securely storing all of your data encrypted in Amazon S3 or Amazon Glacier.

Amazon CloudFront

Amazon CloudFront is a content delivery web service. It integrates with other AWS Cloud services to give developers and businesses an easy way to distribute content to users across the world with low latency, high data transfer speeds, and no minimum usage commitments. Amazon CloudFront can be used to deliver your entire website, including dynamic, static, streaming, and interactive content, using a global network of edge locations. Requests for content are automatically routed to the nearest edge location, so content is delivered with the best possible performance to end users around the globe.

Database Services

AWS provides fully managed relational and NoSQL database services, and in-memory caching as a service and a petabyte-scale data warehouse solution. This section provides an overview of the products that the database services comprise.

Amazon Relational Database Service (Amazon RDS)

Amazon Relational Database Service (Amazon RDS) provides a fully managed relational database with support for many popular open source and commercial database engines. It's a cost-efficient service that allows organizations to launch secure, highly available, fault-tolerant, production-ready databases in minutes. Because Amazon RDS manages time-consuming administration tasks, including backups, software patching, monitoring, scaling, and replication, organizational resources can focus on revenue-generating applications and business instead of mundane operational tasks.

Amazon DynamoDB

Amazon DynamoDB is a fast and flexible NoSQL database service for all applications that need consistent, single-digit millisecond latency at any scale. It is a fully managed database and supports both document and key/value data models. Its flexible data model and reliable performance make it a great fit for mobile, web, gaming, ad-tech, Internet of Things, and many other applications.

Amazon Redshift

Amazon Redshift is a fast, fully managed, petabyte-scale data warehouse service that makes it simple and cost effective to analyze structured data. Amazon Redshift provides a standard SQL interface that lets organizations use existing business intelligence tools. By leveraging

Chapter 7
Databases and AWS

THE AWS CERTIFIED SOLUTIONS ARCHITECT ASSOCIATE EXAM OBJECTIVES COVERED IN THIS CHAPTER MAY INCLUDE, BUT ARE NOT LIMITED TO, THE FOLLOWING:

Domain 1.0: Designing highly available, cost-efficient, fault-tolerant, and scalable systems

✓ **1.1 Identify and recognize cloud architecture considerations, such as fundamental components and effective designs.**

Content may include the following:

- Planning and design
- Architectural trade-off decisions (Amazon Relational Database Service [Amazon RDS] vs. installing on Amazon Elastic Compute Cloud [Amazon EC2])
- Best practices for AWS architecture
- Recovery Time Objective (RTO) and Recovery Point Objective (RPO) Disaster Recovery (DR) design
- Elasticity and scalability

Domain 3.0: Data Security

✓ **3.1 Recognize and implement secure practices for optimum cloud deployment and maintenance.**

Content may include the following:

- AWS administration and security services
- Design patterns

✓ **3.2 Recognize critical disaster recovery techniques and their implementation.**

This chapter will cover essential database concepts and introduce three of Amazon's managed database services: Amazon Relational Database Service (Amazon RDS), Amazon DynamoDB, and Amazon Redshift. These managed services simplify the setup and operation of relational databases, NoSQL databases, and data warehouses.

This chapter focuses on key topics you need to understand for the exam, including:

- The differences among a relational database, a NoSQL database, and a data warehouse
- The benefits and tradeoffs between running a database on Amazon EC2 or on Amazon RDS
- How to deploy database engines into the cloud
- How to back up and recover your database and meet your Recovery Point Objective (RPO) and Recovery Time Objective (RTO) requirements
- How to build highly available database architectures
- How to scale your database compute and storage vertically
- How to select the right type of storage volume
- How to use read replicas to scale horizontally
- How to design and scale an Amazon DynamoDB table
- How to read and write from an Amazon DynamoDB table
- How to use secondary indexes to speed queries
- How to design an Amazon Redshift table
- How to load and query an Amazon Redshift data warehouse
- How to secure your databases, tables, and clusters

Database Primer

Almost every application relies on a database to store important data and records for its users. A database engine allows your application to access, manage, and search large volumes of data records. In a well-architected application, the database will need to meet the performance demands, the availability needs, and the recoverability characteristics of the system.

Database systems and engines can be grouped into two broad categories: Relational Database Management Systems (RDBMS) and NoSQL (or non-relational) databases. It is not uncommon to build an application using a combination of RDBMS and NoSQL databases. A strong understanding of essential database concepts, Amazon RDS, and Amazon DynamoDB are required to pass this exam.

Relational Databases

The most common type of database in use today is the *relational database*. The relational database has roots going back to the 1970s when Edgar F. Codd, working for IBM, developed the concepts of the relational model. Today, relational databases power all types of applications from social media apps, e-commerce websites, and blogs to complex enterprise applications. Commonly used relational database software packages include MySQL, PostgreSQL, Microsoft SQL Server, and Oracle.

Relational databases provide a common interface that lets users read and write from the database using commands or queries written using *Structured Query Language (SQL)*. A relational database consists of one or more tables, and a table consists of columns and rows similar to a spreadsheet. A database column contains a specific attribute of the record, such as a person's name, address, and telephone number. Each attribute is assigned a data type such as text, number, or date, and the database engine will reject invalid inputs.

A database row comprises an individual record, such as the details about a student who attends a school. Consider the example in Table 7.1.

TABLE 7.1 Students Table

StudentID	FirstName	LastName	Gender	Age
1001	Joe	Dusty	M	29
1002	Andrea	Romanov	F	20
1003	Ben	Johnson	M	30
1004	Beth	Roberts	F	30

This is an example of a basic table that would sit in a relational database. There are five fields with different data types:

`StudentID` = Number or integer

`FirstName` = String

`LastName` = String

`Gender` = String (Character Length = 1)

`Age` = Integer

This sample table has four records, with each record representing an individual student. Each student has a `StudentID` field, which is usually a unique number per student. A unique number that identifies each student can be called a *primary key*.

One record in a table can relate to a record in another table by referencing the primary key of a record. This pointer or reference is called a foreign key. For example, the Grades table that records scores for each student would have its own primary key and an additional column known as a foreign key that refers to the primary key of the student record. By referencing the primary keys of other tables, relational databases minimize duplication of data in associated tables. With relational databases, it is important to note that the structure of the table (such as the number of columns and data type of each column) must be defined prior to data being added to the table.

A relational database can be categorized as either an *Online Transaction Processing (OLTP)* or *Online Analytical Processing (OLAP)* database system, depending on how the tables are organized and how the application uses the relational database. OLTP refers to transaction-oriented applications that are frequently writing and changing data (for example, data entry and e-commerce). OLAP is typically the domain of data warehouses and refers to reporting or analyzing large data sets. Large applications often have a mix of both OLTP and OLAP databases.

Amazon Relational Database Service (Amazon RDS) significantly simplifies the setup and maintenance of OLTP and OLAP databases. Amazon RDS provides support for six popular relational database engines: MySQL, Oracle, PostgreSQL, Microsoft SQL Server, MariaDB, and Amazon Aurora. You can also choose to run nearly any database engine using Windows or Linux Amazon Elastic Compute Cloud (Amazon EC2) instances and manage the installation and administration yourself.

Data Warehouses

A *data warehouse* is a central repository for data that can come from one or more sources. This data repository is often a specialized type of relational database that can be used for reporting and analysis via OLAP. Organizations typically use data warehouses to compile reports and search the database using highly complex queries.

Data warehouses are also typically updated on a batch schedule multiple times per day or per hour, compared to an OLTP relational database that can be updated thousands of times per second. Many organizations split their relational databases into two different databases: one database as their main production database for OLTP transactions, and the other database as their data warehouse for OLAP. OLTP transactions occur frequently and are relatively simple. OLAP transactions occur much less frequently but are much more complex.

Amazon RDS is often used for OLTP workloads, but it can also be used for OLAP. *Amazon Redshift* is a high-performance data warehouse designed specifically for OLAP use cases. It is also common to combine Amazon RDS with Amazon Redshift in the same application and periodically extract recent transactions and load them into a reporting database.

NoSQL Databases

NoSQL databases have gained significant popularity in recent years because they are often simpler to use, more flexible, and can achieve performance levels that are difficult or impossible with traditional relational databases. Traditional relational databases are difficult to scale beyond a single server without significant engineering and cost, but a NoSQL architecture allows for horizontal scalability on commodity hardware.

NoSQL databases are non-relational and do not have the same table and column semantics of a relational database. NoSQL databases are instead often key/value stores or document stores with flexible schemas that can evolve over time or vary. Contrast that to a relational database, which requires a very rigid schema.

Many of the concepts of NoSQL architectures trace their foundational concepts back to whitepapers published in 2006 and 2007 that described distributed systems like Dynamo at Amazon. Today, many application teams use Hbase, MongoDB, Cassandra, CouchDB, Riak, and *Amazon DynamoDB* to store large volumes of data with high transaction rates. Many of these database engines support clustering and scale horizontally across many machines for performance and fault tolerance. A common use case for NoSQL is managing user session state, user profiles, shopping cart data, or time-series data.

You can run any type of NoSQL database on AWS using Amazon EC2, or you can choose a managed service like Amazon DynamoDB to deal with the heavy lifting involved with building a distributed cluster spanning multiple data centers.

Amazon Relational Database Service (Amazon RDS)

Amazon RDS is a service that simplifies the setup, operations, and scaling of a relational database on AWS. With Amazon RDS, you can spend more time focusing on the application and the schema and let Amazon RDS offload common tasks like backups, patching, scaling, and replication.

Amazon RDS helps you to streamline the installation of the database software and also the provisioning of infrastructure capacity. Within a few minutes, Amazon RDS can launch one of many popular database engines that is ready to start taking SQL transactions. After the initial launch, Amazon RDS simplifies ongoing maintenance by automating common administrative tasks on a recurring basis.

With Amazon RDS, you can accelerate your development timelines and establish a consistent operating model for managing relational databases. For example, Amazon RDS makes it easy to replicate your data to increase availability, improve durability, or scale up or beyond a single database instance for read-heavy database workloads.

Amazon RDS exposes a database endpoint to which client software can connect and execute SQL. Amazon RDS does not provide shell access to Database (DB) Instances, and it restricts access to certain system procedures and tables that require advanced privileges. With Amazon RDS, you can typically use the same tools to query, analyze, modify, and administer the database. For example, current Extract, Transform, Load (ETL) tools and reporting tools can connect to Amazon RDS databases in the same way with the same drivers, and often all it takes to reconfigure is changing the hostname in the connection string.

Database (DB) Instances

The Amazon RDS service itself provides an Application Programming Interface (API) that lets you create and manage one or more *DB Instances*. A DB Instance is an isolated database environment deployed in your private network segments in the cloud. Each DB Instance runs and manages a popular commercial or open source database engine on your behalf. Amazon RDS currently supports the following database engines: MySQL, PostgreSQL, MariaDB, Oracle, SQL Server, and Amazon Aurora.

You can launch a new DB Instance by calling the `CreateDBInstance` API or by using the AWS Management Console. Existing DB Instances can be changed or resized using the `ModifyDBInstance` API. A DB Instance can contain multiple different databases, all of which you create and manage within the DB Instance itself by executing SQL commands with the Amazon RDS endpoint. The different databases can be created, accessed, and managed using the same SQL client tools and applications that you use today.

The compute and memory resources of a DB Instance are determined by its DB Instance class. You can select the DB Instance class that best meets your needs for compute and memory. The range of DB Instance classes extends from a db.t2.micro with 1 virtual CPU (vCPU) and 1 GB of memory, up to a db.r3.8xlarge with 32 vCPUs and 244 GB of memory. As your needs change over time, you can change the instance class and the balance of compute of memory, and Amazon RDS will migrate your data to a larger or smaller instance class. Independent from the DB Instance class that you select, you can also control the size and

performance characteristics of the storage used.

Amazon RDS supports a large variety of engines, versions, and feature combinations. Check the Amazon RDS documentation to determine support for specific features. Many features and common configuration settings are exposed and managed using *DB parameter groups* and *DB option groups*. A DB parameter group acts as a container for engine configuration values that can be applied to one or more DB Instances. You may change the DB parameter group for an existing instance, but a reboot is required. A DB option group acts as a container for engine features, which is empty by default. In order to enable specific features of a DB engine (for example, Oracle Statspack, Microsoft SQL Server Mirroring), you create a new DB option group and configure the settings accordingly.


Existing databases can be migrated to Amazon RDS using native tools and techniques that vary depending on the engine. For example with MySQL, you can export a backup using mysqldump and import the file into Amazon RDS MySQL. You can also use the AWS Database Migration Service, which gives you a graphical interface that simplifies the migration of both schema and data between databases. AWS Database Migration Service also helps convert databases from one database engine to another.

Operational Benefits

Amazon RDS increases the operational reliability of your databases by applying a very consistent deployment and operational model. This level of consistency is achieved in part by limiting the types of changes that can be made to the underlying infrastructure and through the extensive use of automation. For example with Amazon RDS, you cannot use Secure Shell (SSH) to log in to the host instance and install a custom piece of software. You can, however, connect using SQL administrator tools or use DB option groups and DB parameter groups to change the behavior or feature configuration for a DB Instance. If you want full control of the Operating System (OS) or require elevated permissions to run, then consider installing your database on Amazon EC2 instead of Amazon RDS.

Amazon RDS is designed to simplify the common tasks required to operate a relational database in a reliable manner. It's useful to compare the responsibilities of an administrator when operating a relational database in your data center, on Amazon EC2, or with Amazon RDS (see Table 7.2).

TABLE 7.2 Comparison of Operational Responsibilities

Responsibility	Database On-Premise	Database on Amazon EC2	Database on Amazon RDS
App Optimization	You	You	You
Scaling	You	You	AWS
High Availability	You	You	AWS
Backups	You	You	AWS
DB Engine Patches	You	You	AWS
Software Installation	You	You	AWS
OS Patches	You	You	AWS
OS Installation	You	AWS	AWS
Server Maintenance	You	AWS	AWS
Rack and Stack	You	AWS	AWS
Power and Cooling	You	AWS	AWS

Database Engines

Amazon RDS supports six database engines: MySQL, PostgreSQL, MariaDB, Oracle, SQL Server, and Amazon Aurora. Features and capabilities vary slightly depending on the engine that you select.

MySQL

MySQL is one of the most popular open source databases in the world, and it is used to power a wide range of applications, from small personal blogs to some of the largest websites in the world. As of the time of this writing, Amazon RDS for MySQL currently supports MySQL 5.7, 5.6, 5.5, and 5.1. The engine is running the open source Community Edition with InnoDB as the default and recommended database storage engine. Amazon RDS MySQL allows you to connect using standard MySQL tools such as MySQL Workbench or SQL Workbench/J. Amazon RDS MySQL supports *Multi-AZ* deployments for high availability and *read replicas* for horizontal scaling.

PostgreSQL

PostgreSQL is a widely used open source database engine with a very rich set of features and advanced functionality. Amazon RDS supports DB Instances running several versions of PostgreSQL. As of the time of this writing, Amazon RDS supports multiple releases of PostgreSQL, including 9.5.x, 9.4.x, and 9.3.x. Amazon RDS PostgreSQL can be managed using standard tools like pgAdmin and supports standard JDBC/ODBC drivers. Amazon RDS PostgreSQL also supports Multi-AZ deployment for high availability and read replicas for

horizontal scaling.

MariaDB

Amazon RDS recently added support for DB Instances running MariaDB. MariaDB is a popular open source database engine built by the creators of MySQL and enhanced with enterprise tools and functionality. MariaDB adds features that enhance the performance, availability, and scalability of MySQL. As of the time of this writing, AWS supports MariaDB version 10.0.17. Amazon RDS fully supports the XtraDB storage engine for MariaDB DB Instances and, like Amazon RDS MySQL and PostgreSQL, has support for Multi-AZ deployment and read replicas.

Oracle

Oracle is one of the most popular relational databases used in the enterprise and is fully supported by Amazon RDS. As of the time of this writing, Amazon RDS supports DB Instances running several editions of Oracle 11g and Oracle 12c. Amazon RDS supports access to schemas on a DB Instance using any standard SQL client application, such as Oracle SQL Plus.

Amazon RDS Oracle supports three different editions of the popular database engine: Standard Edition One, Standard Edition, and Enterprise Edition. Table 7.3 outlines some of the major differences between editions:

TABLE 7.3 Amazon RDS Oracle Editions Compared

Edition	Performance	Multi-AZ	Encryption
Standard One	++++	Yes	KMS
Standard	++++++++	Yes	KMS
Enterprise	++++++++	Yes	KMS and TDE

Microsoft SQL Server

Microsoft SQL Server is another very popular relational database used in the enterprise. Amazon RDS allows Database Administrators (DBAs) to connect to their SQL Server DB Instance in the cloud using native tools like SQL Server Management Studio. As of the time of this writing, Amazon RDS provides support for several versions of Microsoft SQL Server, including SQL Server 2008 R2, SQL Server 2012, and SQL Server 2014.

Amazon RDS SQL Server also supports four different editions of SQL Server: Express Edition, Web Edition, Standard Edition, and Enterprise Edition. Table 7.4 highlights the relative performance, availability, and encryption differences among these editions.

TABLE 7.4 Amazon RDS SQL Server Editions Compared

Edition	Performance	Multi-AZ	Encryption
Express	+	No	KMS
Web	++++	No	KMS
Standard	++++	Yes	KMS
Enterprise	++++++++	Yes	KMS and TDE

Licensing

Amazon RDS Oracle and Microsoft SQL Server are commercial software products that require appropriate licenses to operate in the cloud. AWS offers two licensing models: *License Included* and *Bring Your Own License (BYOL)*.

License Included In the License Included model, the license is held by AWS and is included in the Amazon RDS instance price. For Oracle, License Included provides licensing for Standard Edition One. For SQL Server, License Included provides licensing for SQL Server Express Edition, Web Edition, and Standard Edition.

Bring Your Own License (BYOL) In the BYOL model, you provide your own license. For Oracle, you must have the appropriate Oracle Database license for the DB Instance class and Oracle Database edition you want to run. You can bring over Standard Edition One, Standard Edition, and Enterprise Edition.

For SQL Server, you provide your own license under the Microsoft License Mobility program. You can bring over Microsoft SQL Standard Edition and also Enterprise Edition. You are responsible for tracking and managing how licenses are allocated.

Amazon Aurora

Amazon Aurora offers enterprise-grade commercial database technology while offering the simplicity and cost effectiveness of an open source database. This is achieved by redesigning the internal components of MySQL to take a more service-oriented approach.

Like other Amazon RDS engines, Amazon Aurora is a fully managed service, is MySQL-compatible out of the box, and provides for increased reliability and performance over standard MySQL deployments. Amazon Aurora can deliver up to five times the performance of MySQL without requiring changes to most of your existing web applications. You can use the same code, tools, and applications that you use with your existing MySQL databases with Amazon Aurora.

When you first create an Amazon Aurora instance, you create a DB cluster. A DB cluster has one or more instances and includes a cluster volume that manages the data for those instances. An Amazon Aurora cluster volume is a virtual database storage volume that spans multiple Availability Zones, with each Availability Zone having a copy of the cluster data. An Amazon Aurora DB cluster consists of two different types of instances:

Primary Instance This is the main instance, which supports both read and write workloads. When you modify your data, you are modifying the primary instance. Each Amazon Aurora DB cluster has one primary instance.

Amazon Aurora Replica This is a secondary instance that supports only read operations. Each DB cluster can have up to 15 Amazon Aurora Replicas in addition to the primary instance. By using multiple Amazon Aurora Replicas, you can distribute the read workload among various instances, increasing performance. You can also locate your Amazon Aurora Replicas in multiple Availability Zones to increase your database availability.

Storage Options

Amazon RDS is built using Amazon Elastic Block Store (Amazon EBS) and allows you to select the right storage option based on your performance and cost requirements. Depending on the database engine and workload, you can scale up to 4 to 6TB in provisioned storage and up to 30,000 IOPS. Amazon RDS supports three storage types: Magnetic, General Purpose (Solid State Drive [SSD]), and Provisioned IOPS (SSD). Table 7.5 highlights the relative size, performance, and cost differences between types.

TABLE 7.5 Amazon RDS Storage Types

	Magnetic	General Purpose (SSD)	Provisioned IOPS (SSD)
Size	+++	+++++	+++++
Performance	+	+++	+++++
Cost	++	+++	+++++

Magnetic Magnetic storage, also called standard storage, offers cost-effective storage that is ideal for applications with light I/O requirements.

General Purpose (SSD) General purpose (SSD)-backed storage, also called gp2, can provide faster access than magnetic storage. This storage type can provide burst performance to meet spikes and is excellent for small- to medium-sized databases.

Provisioned IOPS (SSD) Provisioned IOPS (SSD) storage is designed to meet the needs of I/O-intensive workloads, particularly database workloads, that are sensitive to storage performance and consistency in random access I/O throughput.


For most applications, General Purpose (SSD) is the best option and provides a good mix of lower-cost and higher-performance characteristics.

Backup and Recovery

Amazon RDS provides a consistent operational model for backup and recovery procedures across the different database engines. Amazon RDS provides two mechanisms for backing up the database: automated backups and manual snapshots. By using a combination of both techniques, you can design a backup recovery model to protect your application data.

Each organization typically will define a *Recovery Point Objective (RPO)* and *Recovery Time Objective (RTO)* for important applications based on the criticality of the application and the expectations of the users. It's common for enterprise systems to have an RPO measured in minutes and an RTO measured in hours or even days, while some critical applications may have much lower tolerances.

RPO is defined as the maximum period of data loss that is acceptable in the event of a failure or incident. For example, many systems back up transaction logs every 15 minutes to allow them to minimize data loss in the event of an accidental deletion or hardware failure.

RTO is defined as the maximum amount of downtime that is permitted to recover from backup and to resume processing. For large databases in particular, it can take hours to restore from a full backup. In the event of a hardware failure, you can reduce your RTO to minutes by failing over to a secondary node. You should create a recovery plan that, at a minimum, lets you recover from a recent backup.

Automated Backups

An *automated backup* is an Amazon RDS feature that continuously tracks changes and backs up your database. Amazon RDS creates a storage volume snapshot of your DB Instance, backing up the entire DB Instance and not just individual databases. You can set the backup retention period when you create a DB Instance. One day of backups will be retained by default, but you can modify the retention period up to a maximum of 35 days. Keep in mind that when you delete a DB Instance, all automated backup snapshots are deleted and cannot be recovered. Manual snapshots, however, are not deleted.

Automated backups will occur daily during a configurable 30-minute maintenance window called the backup window. Automated backups are kept for a configurable number of days, called the *backup retention period*. You can restore your DB Instance to any specific time during this retention period, creating a new DB Instance.

Manual DB Snapshots

In addition to automated backups, you can perform manual *DB snapshots* at any time. A DB snapshot is initiated by you and can be created as frequently as you want. You can then restore the DB Instance to the specific state in the DB snapshot at any time. DB snapshots can be created with the Amazon RDS console or the `CreateDBSnapshot` action. Unlike automated snapshots that are deleted after the retention period, manual DB snapshots are kept until you explicitly delete them with the Amazon RDS console or the `DeleteDBSnapshot` action.

For busy databases, use Multi-AZ to minimize the performance impact of a snapshot. During the backup window, storage I/O may be suspended while your data is being backed up, and you may experience elevated latency. This I/O suspension typically lasts for the duration of the snapshot. This period of I/O suspension is shorter for Multi-AZ DB deployments because the backup is taken from the standby, but latency can occur during the backup process.

Recovery

Amazon RDS allows you to recover your database quickly whether you are performing automated backups or manual DB snapshots. You cannot restore from a DB snapshot to an existing DB Instance; a new DB Instance is created when you restore. When you restore a DB Instance, only the default DB parameter and security groups are associated with the restored

instance. As soon as the restore is complete, you should associate any custom DB parameter or security groups used by the instance from which you restored. When using automated backups, Amazon RDS combines the daily backups performed during your predefined maintenance window in conjunction with transaction logs to enable you to restore your DB Instance to any point during your retention period, typically up to the last five minutes.

High Availability with Multi-AZ

One of the most powerful features of Amazon RDS is Multi-AZ deployments, which allows you to create a database cluster across multiple Availability Zones. Setting up a relational database to run in a highly available and fault-tolerant fashion is a challenging task. With Amazon RDS Multi-AZ, you can reduce the complexity involved with this common administrative task; with a single option, Amazon RDS can increase the availability of your database using replication. Multi-AZ lets you meet the most demanding RPO and RTO targets by using synchronous replication to minimize RPO and fast failover to minimize RTO to minutes.

Multi-AZ allows you to place a secondary copy of your database in another Availability Zone for disaster recovery purposes. Multi-AZ deployments are available for all types of Amazon RDS database engines. When you create a Multi-AZ DB Instance, a primary instance is created in one Availability Zone and a secondary instance is created in another Availability Zone. You are assigned a database instance endpoint such as the following:

```
my_app_db.ch6fe7ykq1zd.us-west-2.rds.amazonaws.com
```

This endpoint is a Domain Name System (DNS) name that AWS takes responsibility for resolving to a specific IP address. You use this DNS name when creating the connection to your database. Figure 7.1 illustrates a typical Multi-AZ deployment spanning two Availability Zones.

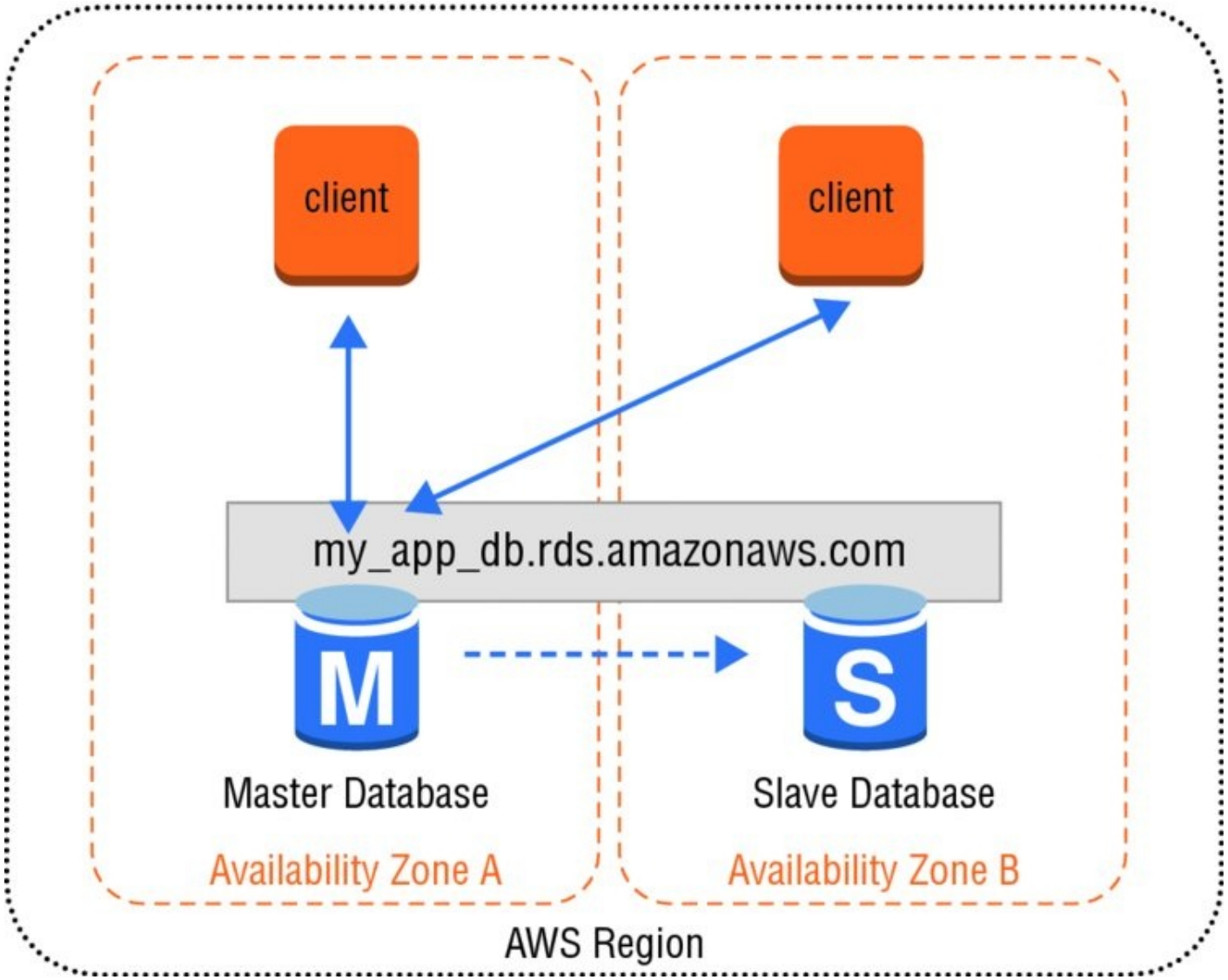


FIGURE 7.1 Multi-AZ Amazon RDS architecture

Amazon RDS automatically replicates the data from the master database or primary instance to the slave database or secondary instance using synchronous replication. Each Availability Zone runs on its own physically distinct, independent infrastructure and is engineered to be highly reliable. Amazon RDS detects and automatically recovers from the most common failure scenarios for Multi-AZ deployments so that you can resume database operations as quickly as possible without administrative intervention. Amazon RDS automatically performs a failover in the event of any of the following:

- Loss of availability in primary Availability Zone
- Loss of network connectivity to primary database
- Compute unit failure on primary database
- Storage failure on primary database

Amazon RDS will automatically fail over to the standby instance without user intervention. The DNS name remains the same, but the Amazon RDS service changes the CNAME to point to the standby. The primary DB Instance switches over automatically to the standby replica if there was an Availability Zone service disruption, if the primary DB Instance fails, or if the instance type is changed. You can also perform a manual failover of the DB Instance. Failover

between the primary and the secondary instance is fast, and the time automatic failover takes to complete is typically one to two minutes.

 It is important to remember that Multi-AZ deployments are for disaster recovery only; they are not meant to enhance database performance. The standby DB Instance is not available to offline queries from the primary master DB Instance. To improve database performance using multiple DB Instances, use read replicas or other DB caching technologies such as Amazon ElastiCache.

Scaling Up and Out

As the number of transactions increase to a relational database, scaling up, or vertically, by getting a larger machine allows you to process more reads and writes. Scaling out, or horizontally, is also possible, but it is often more difficult. Amazon RDS allows you to scale compute and storage vertically, and for some DB engines, you can scale horizontally.

Vertical Scalability

Adding additional compute, memory, or storage resources to your database allows you to process more transactions, run more queries, and store more data. Amazon RDS makes it easy to scale up or down your database tier to meet the demands of your application. Changes can be scheduled to occur during the next maintenance window or to begin immediately using the `ModifyDBInstance` action.

To change the amount of compute and memory, you can select a different DB Instance class of the database. After you select a larger or smaller DB Instance class, Amazon RDS automates the migration process to a new class with only a short disruption and minimal effort.

You can also increase the amount of storage, the storage class, and the storage performance for an Amazon RDS Instance. Each database instance can scale from 5GB up to 6TB in provisioned storage depending on the storage type and engine. Storage for Amazon RDS can be increased over time as needs grow with minimal impact to the running database. Storage expansion is supported for all of the database engines except for SQL Server.

Horizontal Scalability with Partitioning

A relational database can be scaled vertically only so much before you reach the maximum instance size. Partitioning a large relational database into multiple instances or shards is a common technique for handling more requests beyond the capabilities of a single instance.

Partitioning, or *sharding*, allows you to scale horizontally to handle more users and requests but requires additional logic in the application layer. The application needs to decide how to route database requests to the correct shard and becomes limited in the types of queries that can be performed across server boundaries. NoSQL databases like Amazon DynamoDB or Cassandra are designed to scale horizontally.

Horizontal Scalability with Read Replicas

Another important scaling technique is to use *read replicas* to offload read transactions from the primary database and increase the overall number of transactions. Amazon RDS supports read replicas that allow you to scale out elastically beyond the capacity constraints of a single DB Instance for read-heavy database workloads.

There are a variety of use cases where deploying one or more read replica DB Instances is helpful. Some common scenarios include:

- Scale beyond the capacity of a single DB Instance for read-heavy workloads.
- Handle read traffic while the source DB Instance is unavailable. For example, due to I/O suspension for backups or scheduled maintenance, you can direct read traffic to a replica.
- Offload reporting or data warehousing scenarios against a replica instead of the primary DB Instance.

For example, a blogging website may have very little write activity except for the occasional comment, and the vast majority of database activity will be read-only. By offloading some or all of the read activity to one or more read replicas, the primary database instance can focus on handling the writes and replicating the data out to the replicas.

Read replicas are currently supported in Amazon RDS for MySQL, PostgreSQL, MariaDB, and Amazon Aurora. Amazon RDS uses the MySQL, MariaDB, and PostgreSQL DB engines' built-in replication functionality to create a special type of DB Instance, called a read replica, from a source DB Instance. Updates made to the source DB Instance are asynchronously copied to the read replica. You can reduce the load on your source DB Instance by routing read queries from your applications to the read replica.


You can create one or more replicas of a database within a single AWS Region or across multiple AWS Regions. To enhance your disaster recovery capabilities or reduce global latencies, you can use cross-region read replicas to serve read traffic from a region closest to your global users or migrate your databases across AWS Regions.

Security

Securing your Amazon RDS DB Instances and relational databases requires a comprehensive plan that addresses the many layers commonly found in database-driven systems. This includes the infrastructure resources, the database, and the network.

Protect access to your infrastructure resources using AWS Identity and Access Management (IAM) policies that limit which actions AWS administrators can perform. For example, some key administrator actions that can be controlled in IAM include `CreateDBInstance` and `DeleteDBInstance`.

Another security best practice is to deploy your Amazon RDS DB Instances into a private subnet within an Amazon Virtual Private Cloud (Amazon VPC) that limits network access to the DB Instance. Before you can deploy into an Amazon VPC, you must first create a *DB subnet group* that predefines which subnets are available for Amazon RDS deployments. Further, restrict network access using network Access Control Lists (ACLs) and security groups to limit inbound traffic to a short list of source IP addresses.

At the database level, you will also need to create users and grant them permissions to read and write to your databases. Access to the database is controlled using the database engine-specific access control and user management mechanisms. Create users at the database level with strong passwords that you rotate frequently.

Finally, protect the confidentiality of your data in transit and at rest with multiple encryption capabilities provided with Amazon RDS. Security features vary slightly from one engine to another, but all engines support some form of in-transit encryption and also at-rest encryption. You can securely connect a client to a running DB Instance using Secure Sockets Layer (SSL) to protect data in transit. Encryption at rest is possible for all engines using the Amazon Key Management Service (KMS) or *Transparent Data Encryption (TDE)*. All logs, backups, and snapshots are encrypted for an encrypted Amazon RDS instance.

Summary

In this chapter, you learned the basic concepts of relational databases, data warehouses, and NoSQL databases. You also learned about the benefits and features of AWS managed database services Amazon RDS, Amazon Redshift, and Amazon DynamoDB.

Amazon RDS manages the heavy lifting involved in administering a database infrastructure and software and lets you focus on building the relational schemas that best fit your use case and the performance tuning to optimize your queries.

Amazon RDS supports popular open-source and commercial database engines and provides a consistent operational model for common administrative tasks. Increase your availability by running a master-slave configuration across Availability Zones using Multi-AZ deployment. Scale your application and increase your database read performance using read replicas.

Amazon Redshift allows you to deploy a data warehouse cluster that is optimized for analytics and reporting workloads within minutes. Amazon Redshift distributes your records using columnar storage and parallelizes your query execution across multiple compute nodes to deliver fast query performance. Amazon Redshift clusters can be scaled up or down to support large, petabyte-scale databases using SSD or magnetic disk storage.

Connect to Amazon Redshift clusters using standard SQL clients with JDBC/ODBC drivers and execute SQL queries using many of the same analytics and ETL tools that you use today. Load data into your Amazon Redshift clusters using the `COPY` command to bulk import flat files stored in Amazon S3, then run standard `SELECT` commands to search and query the table.

Back up both your Amazon RDS databases and Amazon Redshift clusters using automated and manual snapshots to allow for point-in-time recovery. Secure your Amazon RDS and Amazon Redshift databases using a combination of IAM, database-level access control, network-level access control, and data encryption techniques.

Amazon DynamoDB simplifies the administration and operations of a NoSQL database in the cloud. Amazon DynamoDB allows you to create tables quickly that can scale to an unlimited number of items and configure very high levels of provisioned read and write capacity.

Amazon DynamoDB tables provide a flexible data storage mechanism that only requires a primary key and allows for one or more attributes. Amazon DynamoDB supports both simple scalar data types like String and Number, and also more complex structures using List and Map. Secure your Amazon DynamoDB tables using IAM and restrict access to items and attributes using fine-grained access control.

Amazon DynamoDB will handle the difficult task of cluster and partition management and provide you with a highly available database table that replicates data across Availability Zones for increased durability. Track and process recent changes by tapping into Amazon DynamoDB Streams.

Exam Essentials

Know what a relational database is. A relational database consists of one or more tables. Communication to and from relational databases usually involves simple SQL queries, such as "Add a new record," or "What is the cost of product *x*?" These simple queries are often referred to as OLTP.

Understand which databases are supported by Amazon RDS. Amazon RDS currently supports six relational database engines:

- Microsoft SQL Server
- MySQL Server
- Oracle
- PostgreSQL
- MariaDB
- Amazon Aurora

Understand the operational benefits of using Amazon RDS. Amazon RDS is a managed service provided by AWS. AWS is responsible for patching, antivirus, and management of the underlying guest OS for Amazon RDS. Amazon RDS greatly simplifies the process of setting a secondary slave with replication for failover and setting up read replicas to offload queries.

Remember that you cannot access the underlying OS for Amazon RDS DB instances. You cannot use Remote Desktop Protocol (RDP) or SSH to connect to the underlying OS. If you need to access the OS, install custom software or agents, or want to use a database engine not supported by Amazon RDS, consider running your database on Amazon EC2 instead.

Know that you can increase availability using Amazon RDS Multi-AZ deployment. Add fault tolerance to your Amazon RDS database using Multi-AZ deployment. You can quickly set up a secondary DB Instance in another Availability Zone with Multi-AZ for rapid failover.

Understand the importance of RPO and RTO. Each application should set RPO and RTO targets to define the amount of acceptable data loss and also the amount of time required to recover from an incident. Amazon RDS can be used to meet a wide range of RPO and RTO requirements.

Understand that Amazon RDS handles Multi-AZ failover for you. If your primary Amazon RDS Instance becomes unavailable, AWS fails over to your secondary instance in another Availability Zone automatically. This failover is done by pointing your existing database endpoint to a new IP address. You do not have to change the connection string manually; AWS handles the DNS change automatically.

Remember that Amazon RDS read replicas are used for scaling out and increased performance. This replication feature makes it easy to scale out your read-intensive databases. Read replicas are currently supported in Amazon RDS for MySQL, PostgreSQL,

and Amazon Aurora. You can create one or more replicas of a database within a single AWS Region or across multiple AWS Regions. Amazon RDS uses native replication to propagate changes made to a source DB Instance to any associated read replicas. Amazon RDS also supports cross-region read replicas to replicate changes asynchronously to another geography or AWS Region.

Know what a NoSQL database is. NoSQL databases are non-relational databases, meaning that you do not have to have an existing table created in which to store your data. NoSQL databases come in the following formats:

- Document databases
- Graph stores
- Key/value stores
- Wide-column stores

Remember that Amazon DynamoDB is AWS NoSQL service. You should remember that for NoSQL databases, AWS provides a fully managed service called Amazon DynamoDB. Amazon DynamoDB is an extremely fast NoSQL database with predictable performance and high scalability. You can use Amazon DynamoDB to create a table that can store and retrieve any amount of data and serve any level of request traffic. Amazon DynamoDB automatically spreads the data and traffic for the table over a sufficient number of partitions to handle the request capacity specified by the customer and the amount of data stored, while maintaining consistent and fast performance.

Know what a data warehouse is. A data warehouse is a central repository for data that can come from one or more sources. This data repository would be used for query and analysis using OLAP. An organization's management typically uses a data warehouse to compile reports on specific data. Data warehouses are usually queried with highly complex queries.

Remember that Amazon Redshift is AWS data warehouse service. You should remember that Amazon Redshift is Amazon's data warehouse service. Amazon Redshift organizes the data by column instead of storing data as a series of rows. Because only the columns involved in the queries are processed and columnar data is stored sequentially on the storage media, column-based systems require far fewer I/Os, which greatly improves query performance. Another advantage of columnar data storage is the increased compression, which can further reduce overall I/O.

the AWS Security Token Service. Use the temporary security credentials to sign your requests to Amazon DynamoDB. Amazon DynamoDB is accessible via SSL-encrypted endpoints, and the encrypted endpoints are accessible from both the Internet and from within Amazon EC2.

Amazon Relational Database Service (Amazon RDS) Security

Amazon Relational Database Service (Amazon RDS) allows you to quickly create a relational Database Instance (DB Instance) and flexibly scale the associated compute resources and storage capacity to meet application demand. Amazon RDS manages the database instance on your behalf by performing backups, handling failover, and maintaining the database software. As of the time of this writing, Amazon RDS is available for MySQL, Oracle, Microsoft SQL Server, MariaDB, Amazon Aurora, and PostgreSQL database engines.

Amazon RDS has multiple features that enhance reliability for critical production databases, including DB security groups, permissions, SSL connections, automated backups, DB snapshots, and multiple Availability Zone (Multi-AZ) deployments. DB Instances can also be deployed in an Amazon VPC for additional network isolation.

Access Control When you first create a DB Instance within Amazon RDS, you will create a master user account, which is used only within the context of Amazon RDS to control access to your DB Instance(s). The master user account is a native database user account that allows you to log on to your DB Instance with all database privileges. You can specify the master user name and password you want associated with each DB Instance when you create the DB Instance. After you have created your DB Instance, you can connect to the database using the master user credentials. Subsequently, you can create additional user accounts so that you can restrict who can access your DB Instance.

You can control Amazon RDS DB Instance access via *DB security groups*, which are similar to Amazon EC2 security groups but not interchangeable. DB security groups act like a firewall controlling network access to your DB Instance. DB security groups default to deny all access mode, and customers must specifically authorize network ingress. There are two ways of doing this:

- Authorizing a network IP range
- Authorizing an existing Amazon EC2 security group

DB security groups only allow access to the database server port (all others are blocked) and can be updated without restarting the Amazon RDS DB Instance, which gives you seamless control of their database access.

Using AWS IAM, you can further control access to your Amazon RDS DB instances. AWS IAM enables you to control what Amazon RDS operations each individual AWS IAM user has permission to call.

Network Isolation For additional network access control, you can run your DB Instances in an Amazon VPC. Amazon VPC enables you to isolate your DB Instances by specifying the IP range you want to use and connect to your existing IT infrastructure through industry-standard encrypted IPsec VPN. Running Amazon RDS in a VPC enables you to have a DB instance within a private subnet. You can also set up a virtual private gateway that extends your corporate network into your VPC, and allows access to the RDS DB instance in that VPC.

For Multi-AZ deployments, defining a subnet for all Availability Zones in a region, will allow

Amazon RDS to create a new standby in another Availability Zone should the need arise. You can create DB subnet groups, which are collections of subnets that you may want to designate for your Amazon RDS DB Instances in an Amazon VPC. Each DB subnet group should have at least one subnet for every Availability Zone in a given region. In this case, when you create a DB Instance in an Amazon VPC, you select a DB subnet group; Amazon RDS then uses that DB subnet group and your preferred Availability Zone to select a subnet and an IP address within that subnet. Amazon RDS creates and associates an Elastic Network Interface to your DB Instance with that IP address.

DB Instances deployed within an Amazon VPC can be accessed from the Internet or from Amazon EC2 instances outside the Amazon VPC via VPN or bastion hosts that you can launch in your public subnet. To use a bastion host, you will need to set up a public subnet with an Amazon EC2 instance that acts as a SSH Bastion. This public subnet must have an Internet gateway and routing rules that allow traffic to be directed via the SSH host, which must then forward requests to the private IP address of your Amazon RDS DB Instance.

DB security groups can be used to help secure DB Instances within an Amazon VPC. In addition, network traffic entering and exiting each subnet can be allowed or denied via network ACLs. All network traffic entering or exiting your Amazon VPC via your IPsec VPN connection can be inspected by your on-premises security infrastructure, including network firewalls and intrusion detection systems.

Encryption You can encrypt connections between your application and your DB Instance using SSL. For MySQL and SQL Server, Amazon RDS creates an SSL certificate and installs the certificate on the DB Instance when the instance is provisioned. For MySQL, you launch the MySQL client using the `--ssl_ca` parameter to reference the public key in order to encrypt connections. For SQL Server, download the public key and import the certificate into your Windows operating system. Oracle RDS uses Oracle native network encryption with a DB Instance. You simply add the native network encryption option to an option group and associate that option group with the DB Instance. After an encrypted connection is established, data transferred between the DB Instance and your application will be encrypted during transfer. You can also require your DB Instance to accept only encrypted connections.

Amazon RDS supports Transparent Data Encryption (TDE) for SQL Server (SQL Server Enterprise Edition) and Oracle (part of the Oracle Advanced Security option available in Oracle Enterprise Edition). The TDE feature automatically encrypts data before it is written to storage and automatically decrypts data when it is read from storage. If you require your MySQL data to be encrypted while at rest in the database, your application must manage the encryption and decryption of data.

Note that SSL support within Amazon RDS is for encrypting the connection between your application and your DB Instance; it should not be relied on for authenticating the DB Instance itself. While SSL offers security benefits, be aware that SSL encryption is a compute intensive operation and will increase the latency of your database connection.

Automated Backups and DB Snapshots Amazon RDS provides two different methods for backing up and restoring your DB Instance(s): automated backups and Database Snapshots (DB Snapshots). Turned on by default, the automated backup feature of Amazon RDS enables point-in-time recovery for your DB Instance. Amazon RDS will back up your database and transaction logs and store both for a user-specified retention period. This allows

you to restore your DB Instance to any second during your retention period, up to the last five minutes. Your automatic backup retention period can be configured to up to 35 days.

DB Snapshots are user-initiated backups of your DB Instance. These full database backups are stored by Amazon RDS until you explicitly delete them. You can copy DB snapshots of any size and move them between any of AWS public regions, or copy the same snapshot to multiple regions simultaneously. You can then create a new DB Instance from a DB Snapshot whenever you desire.

During the backup window, storage I/O may be suspended while your data is being backed up. This I/O suspension typically lasts a few minutes. This I/O suspension is avoided with Multi-AZ DB deployments, because the backup is taken from the standby.

DB Instance Replication AWS Cloud computing resources are housed in highly available data center facilities in different regions of the world, and each region contains multiple distinct locations called Availability Zones. Each Availability Zone is engineered to be isolated from failures in other Availability Zones and provide inexpensive, low-latency network connectivity to other Availability Zones in the same region.

To architect for high availability of your Oracle, PostgreSQL, or MySQL databases, you can run your Amazon RDS DB Instance in several Availability Zones, an option called a *Multi-AZ deployment*. When you select this option, AWS automatically provisions and maintains a synchronous standby replica of your DB Instance in a different Availability Zone. The primary DB Instance is synchronously replicated across Availability Zones to the standby replica. In the event of DB Instance or Availability Zone failure, Amazon RDS will automatically failover to the standby so that database operations can resume quickly without administrative intervention.

For customers who use MySQL and need to scale beyond the capacity constraints of a single DB Instance for read-heavy database workloads, Amazon RDS provides a read replica option. After you create a read replica, database updates on the source DB Instance are replicated to the read replica using MySQL's native, asynchronous replication. You can create multiple read replicas for a given source DB instance and distribute your application's read traffic among them. Read replicas can be created with Multi-AZ deployments to gain read scaling benefits in addition to the enhanced database write availability and data durability provided by Multi-AZ deployments.

Automatic Software Patching Amazon RDS will make sure that the relational database software powering your deployment stays up-to-date with the latest patches. When necessary, patches are applied during a maintenance window that you can control. You can think of the Amazon RDS maintenance window as an opportunity to control when DB Instance modifications (such as scaling DB Instance class) and software patching occur, in the event either are requested or required. If a maintenance event is scheduled for a given week, it will be initiated and completed at some point during the 30-minute maintenance window you identify.

The only maintenance events that require Amazon RDS to take your DB Instance offline are scale compute operations (which generally take only a few minutes from start to finish) or required software patching. Required patching is automatically scheduled only for patches that are related to security and durability. Such patching occurs infrequently (typically once every few months) and should seldom require more than a fraction of your maintenance

window. If you do not specify a preferred weekly maintenance window when creating your DB Instance, a 30-minute default value is assigned. If you want to modify when maintenance is performed on your behalf, you can do so by modifying your DB Instance in the AWS Management Console or by using the `ModifyDBInstance` API. Each of your DB Instances can have different preferred maintenance windows, if you so choose.

Running your DB Instance in a Multi-AZ deployment can further reduce the impact of a maintenance event, as Amazon RDS will conduct maintenance via the following steps:

1. Perform maintenance on standby.
2. Promote standby to primary.
3. Perform maintenance on old primary, which becomes the new standby.

When an Amazon RDS DB Instance deletion API (`DeleteDBInstance`) is run, the DB Instance is marked for deletion. After the instance no longer indicates deleting status, it has been removed. At this point, the instance is no longer accessible, and unless a final snapshot copy was asked for, it cannot be restored and will not be listed by any of the tools or APIs.

Amazon Redshift Security

Amazon Redshift is a petabyte-scale SQL data warehouse service that runs on highly optimized and managed AWS compute and storage resources. The service has been architected not only to scale up or down rapidly, but also to improve query speeds significantly even on extremely large datasets. To increase performance, Amazon Redshift uses techniques such as columnar storage, data compression, and zone maps to reduce the amount of I/O needed to perform queries. It also has a Massively Parallel Processing (MPP) architecture, parallelizing and distributing SQL operations to take advantage of all available resources.

Cluster Access By default, clusters that you create are closed to everyone. Amazon Redshift enables you to configure firewall rules (security groups) to control network access to your data warehouse cluster. You can also run Amazon Redshift inside an Amazon VPC to isolate your data warehouse cluster in your own virtual network and connect it to your existing IT infrastructure using industry-standard encrypted IPsec VPN.

The AWS account that creates the cluster has full access to the cluster. Within your AWS account, you can use AWS IAM to create user accounts and manage permissions for those accounts. By using IAM, you can grant different users permission to perform only the cluster operations that are necessary for their work. Like all databases, you must grant permission in Amazon Redshift at the database level in addition to granting access at the resource level. Database users are named user accounts that can connect to a database and are authenticated when they log in to Amazon Redshift. In Amazon Redshift, you grant database user permissions on a per-cluster basis instead of on a per-table basis. However, users can see data only in the table rows that were generated by their own activities; rows generated by other users are not visible to them.

The user who creates a database object is its owner. By default, only a super user or the owner of an object can query, modify, or grant permissions on the object. For users to use an object, you must grant the necessary permissions to the user or the group that contains the user. In addition, only the owner of an object can modify or delete it.

Amazon Simple Storage Service (Amazon S3) Amazon S3 allows you to upload and retrieve data at any time, from anywhere on the web. Access to data stored in Amazon S3 is restricted by default; only bucket and object owners have access to the Amazon S3 resources they create. You can securely upload and download data to Amazon S3 via the SSL-encrypted endpoints. Amazon S3 supports several methods to encrypt data at rest.

Amazon Glacier Amazon Glacier service provides low-cost, secure, and durable storage. You can securely upload and download data to Amazon Glacier via the SSL-encrypted endpoints, and the service automatically encrypts the data using AES-256 and stores it durably in an immutable form.

AWS Storage Gateway AWS Storage Gateway service connects your on-premises software appliance with cloud-based storage to provide seamless and secure integration between your IT environment and AWS storage infrastructure. Data is asynchronously transferred from your on-premises storage hardware to AWS over SSL and stored encrypted in Amazon S3 using AES-256.

Database

Amazon DynamoDB Amazon DynamoDB is a managed NoSQL database service that provides fast and predictable performance with seamless scalability. You can control access at the database level by creating database-level permissions that allow or deny access to items (rows) and attributes (columns) based on the needs of your application.

Amazon Relational Database Service (RDS) Amazon RDS allows you to quickly create a relational DB Instance and flexibly scale the associated compute resources and storage capacity to meet application demand. You can control Amazon RDS DB Instance access via DB security groups, which act like a firewall controlling network access to your DB Instance. Database security groups default to deny all access mode, and customers must specifically authorize network ingress. Amazon RDS is supported within an Amazon VPC, and for Multi-AZ deployments, defining a subnet for all Availability Zones in a region will allow Amazon RDS to create a new standby in another Availability Zone should the need arise. You can encrypt connections between your application and your DB Instance using SSL, and you can encrypt data at rest within Amazon RDS instances for all database engines.

Amazon Redshift Amazon Redshift is a petabyte-scale SQL data warehouse service that runs on highly optimized and managed AWS compute and storage resources. The service enables you to configure firewall rules (security groups) to control network access to your data warehouse cluster. Database users are named user accounts that can connect to a database and are authenticated when they log in to Amazon Redshift. In Amazon Redshift, you grant database user permissions on a per-cluster basis instead of on a per-table basis. You may choose for Amazon Redshift to store all data in user-created tables in an encrypted format using hardware-accelerated AES-256 block encryption keys. This includes all data written to disk and also any backups. Amazon Redshift uses a four-tier, key-based architecture for encryption. These keys consist of data encryption keys, a database key, a cluster key, and a master key.

Amazon ElastiCache Amazon ElastiCache is a web service that makes it easy to set up, manage, and scale distributed in-memory cache environments in the cloud. Amazon ElastiCache allows you to control access to your Cache Clusters using Cache Security Groups.

AWS®

Certified Developer

Official Study Guide

Associate (DVA-C01) Exam

Now that you know the purpose of these database services and what they can do, review the type of applications that can be used for each AWS database service. Refer to the application type mappings shown in Table 4.2.

TABLE 4.2 Application Mapping to AWS Database Service

Applications	Product
Transactional applications, such as ERP, CRM, and ecommerce to log transactions and store structured data.	Aurora or Amazon RDS
Internet-scale applications, such as hospitality, dating, and ride sharing, to serve content and store structured and unstructured data.	DynamoDB or Amazon DocumentDB
Analytic applications for operational reporting and querying terabyte- to exabyte-scale data.	Amazon Redshift
Real-time application use cases that require submillisecond latency such as gaming leaderboards, chat, messaging, streaming, and Internet of Things (IoT).	ElastiCache
Applications with use cases that require navigation of highly connected data such as social news feeds, recommendations, and fraud detection.	Neptune
Applications that collect data at millions of inserts per second in a time-series fashion, for example clickstream data and IoT devices.	Timestream
Applications that require an accurate history of their application data; for example, tracking the history of credits and debits in banking transactions or verifying the audit trails created in relational databases.	Amazon QLDB

Relational Databases

Many developers have had to interact with relational databases in their applications. This section describes first what a relational database is. Then, it covers how you can run a relational database in the AWS Cloud with Amazon RDS or on Amazon EC2.

A *relational database* is a collection of data items with predefined relationships between them. These items are organized as a set of tables with columns and rows. Tables store information about the *objects* to be represented in the database. Each *column* in a table

holds certain data, and a *field* stores the actual value of an attribute. The *rows* in the table represent a collection of related values of one object or entity. Each row in a table contains a unique identifier called a *primary key*, and rows among multiple tables can be linked by using *foreign keys*. You can access data in many different ways without reorganizing the database tables.

Characteristics of Relational Databases

Relational databases include four important characteristics: Structured Query Language, data integrity, transactions, and atomic, consistent, isolated, and durable compliance.

Structured Query Language

Structured query language (SQL) is the primary interface that you use to communicate with relational databases. The standard American National Standards Institute (ANSI) SQL is supported by all popular relational database engines. Some of these engines have extensions to ANSI SQL to support functionality that is specific to that engine. You use SQL to add, update, or delete data rows; to retrieve subsets of data for transaction processing and analytics applications; and to manage all aspects of the database.

Data Integrity

Data integrity is the overall completeness, accuracy, and consistency of data. Relational databases use a set of constraints to enforce data integrity in the database. These include primary keys, foreign keys, NOT NULL constraints, unique constraint, default constraints, and check constraints. These integrity constraints help enforce business rules in the tables to ensure the accuracy and reliability of the data. In addition, most relational databases enable you to embed custom code triggers that execute based on an action on the database.

Transactions

A database *transaction* is one or more SQL statements that execute as a sequence of operations to form a single logical unit of work. Transactions provide an all-or-nothing proposition, meaning that the entire transaction must complete as a single unit and be written to the database, or none of the individual components of the transaction will continue. In relational database terminology, a transaction results in a `COMMIT` or a `ROLLBACK`. Each transaction is treated in a coherent and reliable way, independent of other transactions.

ACID Compliance

All database transactions must be *atomic, consistent, isolated, and durable (ACID)*–compliant or be atomic, consistent, isolated, and durable to ensure data integrity.

Atomicity *Atomicity* requires that the transaction as a whole executes successfully, or if a part of the transaction fails, then the entire transaction is invalid.

Consistency *Consistency* mandates that the data written to the database as part of the transaction must adhere to all defined rules and restrictions, including constraints, cascades, and triggers.

Isolation *Isolation* is critical to achieving concurrency control, and it makes sure that each transaction is independent unto itself.

Durability *Durability* requires that all of the changes made to the database be permanent when a transaction is successfully completed.

Managed vs. Unmanaged Databases

Managed database services on AWS, such as Amazon RDS, enable you to offload the administrative burdens of operating and scaling distributed databases to AWS so that you don't have to worry about the following tasks:

- Hardware provisioning
- Setup and configuration
- Throughput capacity planning
- Replication
- Software patching
- Cluster scaling

AWS provides a number of database alternatives for developers. As a managed database, Amazon RDS enables you to run a fully featured relational database while offloading database administration. By contrast, you can run unmanaged databases on Amazon EC2, which gives you more flexibility on the types of databases that you can deploy and configure; however, you are responsible for the administration of the unmanaged databases.

Amazon Relational Database Service

With *Amazon Relational Database Service (Amazon RDS)*, you can set up, operate, and scale a relational database in the AWS Cloud. It provides cost-efficient, resizable capacity for open-standard relational database engines. Amazon RDS is easy to administer, and you do not need to install the database software. Amazon RDS manages time-consuming database administration tasks, which frees you up to focus on your applications and business. For example, Amazon RDS automatically patches the database software and backs up your database. The Amazon RDS managed relational database service works with the popular database engines depicted in Figure 4.1.

FIGURE 4.1 Amazon RDS database engines

Amazon RDS assumes many of the difficult or tedious management tasks of a relational database:

Procurement, configuration, and backup tasks

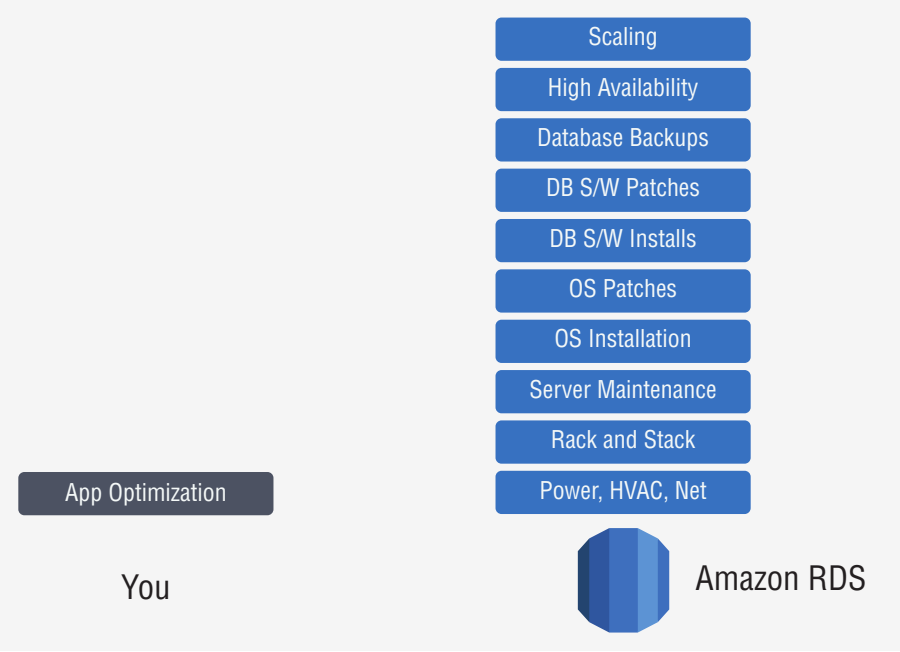
- When you buy a server, you get a central processing unit (CPU), memory, storage, and input/output operations per second (IOPS) all bundled together. With Amazon RDS, these are split apart so that you can scale them independently and allocate your resources as you need them.
- Amazon RDS manages backups, software patches, automatic failure detection, and recovery.
- You can configure automated backups or manually create your own backup snapshot and use these backups to restore a database. The Amazon RDS restore process works reliably and efficiently.
- You can use familiar database products: MySQL, MariaDB, PostgreSQL, Oracle, Microsoft SQL Server, and the MySQL- and PostgreSQL-compatible Amazon Aurora DB engine.

Security and availability

- You can enable the encryption option for your Amazon RDS DB instance.
- You can get high availability with a primary instance and a synchronous secondary instance that you can fail over to when problems occur. You can also use MySQL, MariaDB, or PostgreSQL read replicas to increase read scaling.
- In addition to the security in your database package, you can use AWS Identity and Access Management (IAM) to define users, and permissions help control who can access your Amazon RDS databases. You can also help protect your databases by storing them in a virtual private cloud (VPC).
- To deliver a managed service experience, Amazon RDS does not provide shell access to DB instances, and it restricts access to certain system procedures and tables that require advanced permissions.

When you host databases on Amazon RDS, AWS is responsible for the items in Figure 4.2.

FIGURE 4.2 Amazon RDS host responsibilities



Relational Database Engines on Amazon RDS

Amazon RDS provides six familiar database engines: Amazon Aurora, Oracle, Microsoft SQL Server, PostgreSQL, MySQL, and MariaDB. Because Amazon RDS is a managed service, you gain a number of benefits and features built right into the Amazon RDS service. These features include, but are not limited to, the following:

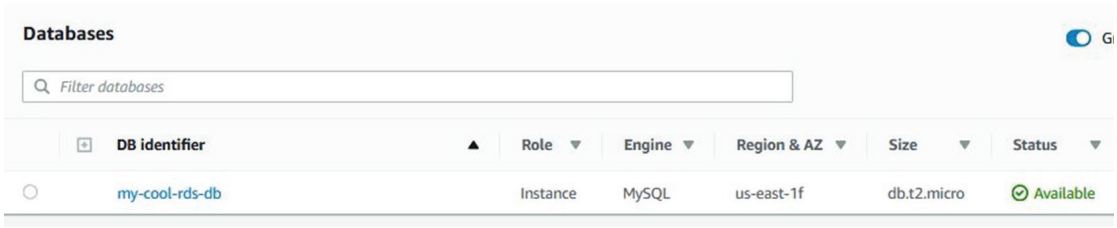
- Automatic software patching
- Easy vertical scaling
- Easy storage scaling
- Read replicas
- Automatic backups
- Database snapshots
- Multi-AZ deployments
- Encryption
- IAM DB authentication
- Monitoring and metrics with Amazon CloudWatch

To create an Amazon RDS instance, you can run the following command from the AWS CLI:

```
aws rds create-db-instance \
--db-instance-class db.t2.micro \
--allocated-storage 30 \
--db-instance-identifier my-cool-rds-db --engine mysql \
--master-username masteruser --master-user-password masterpassword1!
```

Depending on the configurations chosen, the database can take several minutes before it is active and ready for use. You can monitor the Amazon RDS Databases console to view the status. When the status states `Available`, it is ready to be used, as shown in Figure 4.3.

FIGURE 4.3 Amazon RDS Databases console

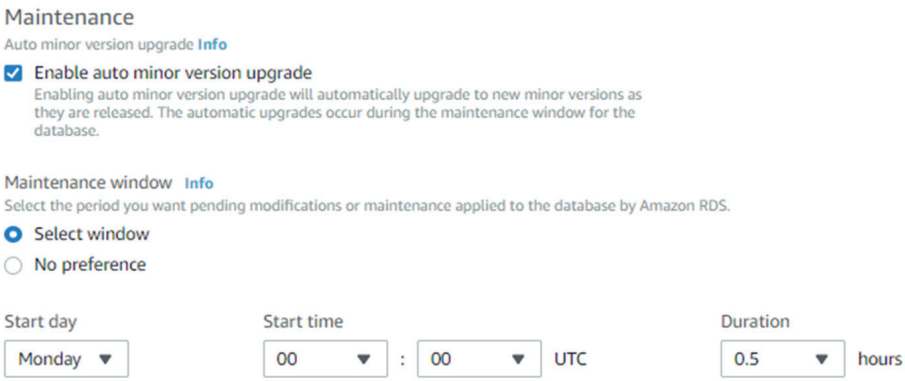


Automatic Software Patching

Periodically, Amazon RDS performs maintenance on Amazon RDS resources. Maintenance mostly involves patching the Amazon RDS database underlying operating system (OS) or database engine version. Because this is a managed service, Amazon RDS handles the patching for you.

When you create an Amazon RDS database instance, you can define a maintenance window. A *maintenance window* is where you can define a period of time when you want to apply any updates or downtime to your database instance. You also can enable the automatic minor version upgrade feature, which automatically applies any new minor versions of the database as they are released (see Figure 4.4).

FIGURE 4.4 Maintenance window



You can select a maintenance window by using the AWS Management Console, AWS CLI, or Amazon RDS API. After selecting the maintenance window, the Amazon RDS instance is upgraded (if upgrades are available) during that time window. You can also modify the maintenance window by running the following AWS CLI command:

```
aws rds modify-db-instance --db-instance-identifer your-db-instance-identifer
--preferred-maintenance-window Mon:07:00-Mon:07:30
```

Vertical Scaling

If your database needs to handle a bigger load, you can vertically scale your Amazon RDS instance. At the time of this writing, there are 40 available DB instance classes, which enable you to choose the number of virtual CPUs and memory available. This gives you flexibility over the performance and cost of your Amazon RDS database. To scale the Amazon RDS instance, you can use the console, AWS CLI, or AWS SDK.

If you are in a Single-AZ configuration for your Amazon RDS instance, the database is unavailable during the scaling operation. However, if you are in a Multi-AZ configuration, the standby database is upgraded first and then a failover occurs to the newly configured database. You can also apply the change during the next maintenance window. This way, your upgrade can occur during your normal outage windows.

To scale the Amazon RDS database by using the AWS CLI, run the following command:

```
aws rds modify-db-instance --db-instance-identifer your-db-instance-identifer
--db-instance-class db.t2.medium
```

Easy Storage Scaling

Storage is a critical component for any database. Amazon RDS has the following three storage types:

General Purpose SSD (gp2) This storage type is for cost-effective storage that is ideal for a broad range of workloads. Gp2 volumes deliver single-digit millisecond latencies and the ability to burst to 3,000 IOPS for extended periods of time. The volume's size determines the performance of gp2 volumes.

Provisioned IOPS (io1) This storage type is for input/output-intensive workloads that require low input/output (I/O) latency and consistent I/O throughput.

Magnetic Storage This storage type is designed for backward compatibility, and AWS recommends that you use General Purpose SSD or Provisioned IOPS for any new Amazon RDS workloads.

To scale your storage, you must modify the Amazon RDS DB instance by executing the following AWS CLI command:

```
aws rds modify-db-instance --db-instance-identifer your-db-instance-identifer
--allocated-storage 50 --storage-type io1 --iops 3000
```

This command modifies your storage to 50 GB in size, with a Provisioned IOPS storage drive and a dedicated IOPS of 3000. While modifying the Amazon RDS DB instance, consider the potential downtime.

Read Replicas (Horizontal Scaling)

There are two ways to scale your database tier with Amazon RDS: vertical scaling and horizontal scaling. Vertical scaling takes the primary database and increases the amount of memory and vCPUs allocated for the primary database. Alternatively, use horizontal scaling (add another server) to your database tier to improve the performance of applications that are read-heavy as opposed to write-heavy.

Read replicas create read-only copies of your master database, which allow you to offload any reads (or SQL SELECT statements) to the read replica. The replication from the master database to the read replica is asynchronous. As a result, the data queried from the read replica is not the latest data. If your application requires strongly consistent reads, consider an alternative option.

At the time of this writing, Amazon RDS MySQL, PostgreSQL, and MariaDB support up to five read replicas, and Amazon Aurora supports up to 15 read replicas. Microsoft SQL Server and Oracle do not support read replicas.

To create a read replica by using AWS CLI, run the following command:

```
aws rds create-db-instance-read-replica --db-instance-identifier your-db-
instance-identifier --source-db-instance-identifier your-source-db
```

Backing Up Data with Amazon RDS

Amazon RDS has two different ways of backing up data of your database instance: automated backups and database snapshots (DB snapshots).

Automated Backups (Point-in-Time)

With Amazon RDS, automated backups offer a point-in-time recovery of your database. When enabled, Amazon RDS performs a full daily snapshot of your data that is taken during your preferred backup window. After the initial backup is taken (each day), then Amazon RDS captures transaction logs as changes are made to the database.

After you initiate a point-in-time recovery, to restore your database instance, the transaction logs are applied to the most appropriate daily backup. You can perform a restore up to the specific second, as long as it's within your retention period. The default retention period is seven days, but it can be a maximum of up to 35 days.

To perform a restore, you must choose the `Latest Restorable Time`, which is typically within the last 5 minutes. For example, suppose that the current date is February 14 at 10 p.m., and you would like to do a point-in-time restore of February 14 at 9 p.m. This restore would succeed because the `Latest Restorable Time` is a maximum of February 14 at 9:55 p.m. (which is the last 5-minute window). However, a point-in-time restore of February 14 at 9:58 p.m. would fail, because it is within the 5-minute window.

Automated backups are kept until the source database is deleted. After the source Amazon RDS instance is removed, the automated backups are also removed.

Database Snapshots (Manual)

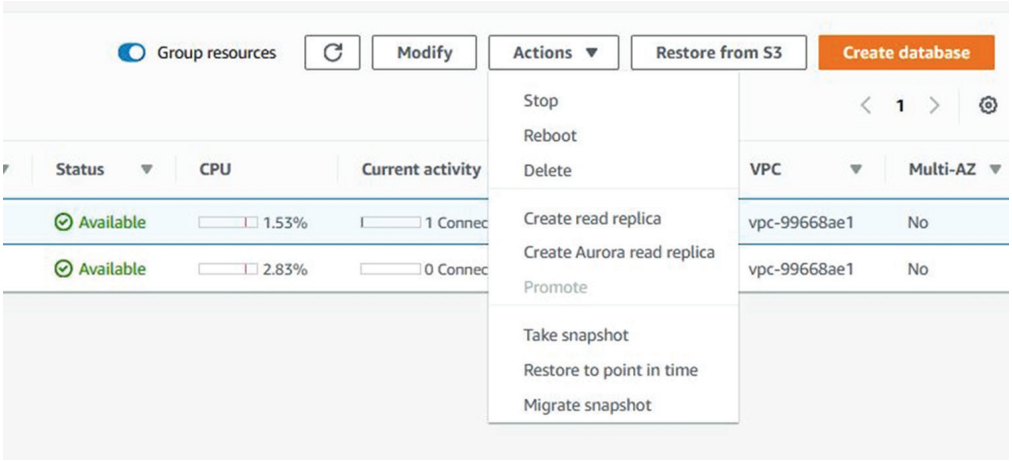
Unlike automated backups, database snapshots with Amazon RDS are user-initiated and enable you to back up your database instance in a known state at any time. You can also restore to that specific snapshot at any time.

Similar to the other Amazon RDS features, you can create the snapshots through the AWS Management Console, with the `CreateDBSnapshot` API, or with the AWS CLI.

With DB snapshots, the backups are kept until you explicitly delete them; therefore, before removing any Amazon RDS instance, take a final snapshot before removing it. Regardless of the backup taken, storage I/O may be briefly suspended while the backup process initializes (typically a few seconds), and you may experience a brief period of elevated latency. A way to avoid these types of suspensions is to deploy in a Multi-AZ configuration. With such a deployment, the backup is taken from the standby instead of the primary database.

To create a snapshot of the database, from the Amazon RDS Databases console, under Actions, select the Take snapshot option (see Figure 4.5). After a snapshot is taken, you can view all of your snaps from the Snapshots console.

FIGURE 4.5 Taking an Amazon RDS snapshot



Multi-AZ Deployments

By using Amazon RDS, you can run in a Multi-AZ configuration. In a Multi-AZ configuration, you have a primary and a standby DB instance. Updates to the primary database replicate synchronously to the standby replica in a different Availability Zone. The primary benefit of Multi-AZ is realized during certain types of planned maintenance, or in the unlikely event of a DB instance failure or an Availability Zone failure. Amazon RDS automatically fails over to the standby so that you can resume your workload as soon as the standby is promoted to the primary. This means that you can reduce your downtime in the event of a failure.

Because Amazon RDS is a managed service, Amazon RDS handles the fail to the standby. When there is a DB instance failure, Amazon RDS automatically promotes the standby to the primary—you will not interact with the standby directly. In other words, you will receive one endpoint connection for the Amazon RDS cluster, and Amazon RDS handles the failover.

Amazon RDS Multi-AZ configuration provides the following benefits:

- Automatic failover; no administration required
- Increased durability in the unlikely event of component failure
- Increased availability in the unlikely event of an Availability Zone failure
- Increased availability for planned maintenance (automated backups; I/O activity is no longer suspended)

To create an Amazon RDS instance in a Multi-AZ configuration, you must specify a subnet group that has two different Availability Zones specified. You can specify a Multi-AZ configuration by using AWS CLI by adding the `--multi-az` flag to the AWS CLI command, as follows:

```
aws rds create-db-instance \
--db-instance-class db.t2.micro \
--allocated-storage 30 \
--db-instance-identifier multi-az-rds-db --engine mysql \
--master-username masteruser \
--master-user-password masterpassword1! \
--multi-az
```

Encryption

For encryption at rest, Amazon RDS uses the AWS Key Management Service (AWS KMS) for AES-256 encryption. You can use a default master key or specify your own for the Amazon RDS DB instance. Encryption is one of the few options that must be configured when the DB instance is created. You cannot modify an Amazon RDS database to enable encryption. You can, however, create a DB snapshot and then restore to an encrypted DB instance or cluster.

Amazon RDS supports using the Transparent Data Encryption (TDE) for Oracle and SQL Server. For more information on TDE with Oracle and Microsoft SQL Server, see the following:

- Microsoft SQL Server Transparent Data Encryption Support at:

  `https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/Appendix.SQLServer.Options.TDE.html`

- Options for Oracle DB Instances:

  `https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/Appendix.Oracle.Options.html#Appendix.Oracle.Options.AdvSecurity`

At the time of this writing, the following Amazon RDS DB instance types are not supported for encryption at rest:

- Db.m1.small
- Db.m1.medium
- Db.m1.large
- Db.m1.xlarge
- Db.m2.xlarge
- Db.m2.2xlarge
- Db.m2.4xlarge
- Db.t2.micro

For encryption in transit, Amazon RDS generates an SSL certificate for each database instance that can be used to connect your application and the Amazon RDS instance. However, encryption is a compute-intensive operation that increases the latency of your database connection. For more information, see the documentation for the specific database engine.

IAM DB Authentication

You can authenticate to your DB instance by using IAM. By using IAM, you can manage access to your database resources centrally instead of storing the user credentials in each database. The IAM feature also encrypts network traffic to and from the database by using SSL.

IAM DB authentication is supported only for MySQL and PostgreSQL. At the time of this writing, the following MySQL versions are supported:

- MySQL 5.6.34 or later
- MySQL 5.7.16 or later

There's no support for the following:

- IAM DB Authentication for MySQL 5.5 or MySQL 8.0
- db.t2.small and db.m1.small instances

The following PostgreSQL versions are supported:

- PostgreSQL versions 10.6 or later
- PostgreSQL 9.6.11 or later
- PostgreSQL 9.5.15 or later

To enable IAM DB authentication for your Amazon RDS instance, run the following command:

```
aws rds modify-db-instance --db-instance-identifier my-rds-db --enable-iam-
database-authentication --apply-immediately
```

Because downtime is associated with this action, you can enable this feature during the next maintenance window. You can do so by changing the last parameter to `--no-apply-immediately`.

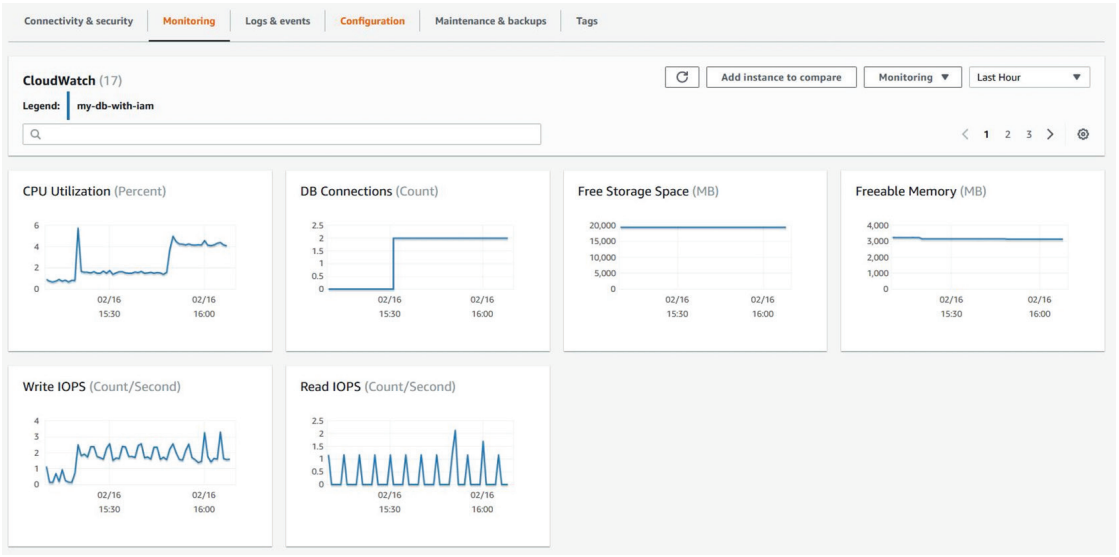
Monitoring with Amazon CloudWatch

Use Amazon CloudWatch to monitor your database tier. You can create alarms to notify database administrators when there is a failure.

By default, CloudWatch provides some built-in metrics for Amazon RDS with a granularity of 5 minutes (600 seconds). If you want to gather metrics in a smaller window of granularity, such as 1 second, enable enhanced monitoring, which is similar to how you enable these features in Amazon EC2.

To view all the Amazon RDS metrics that are provided through CloudWatch, select the Monitoring tab from the Amazon RDS console (see Figure 4.6).

FIGURE 4.6 Amazon RDS with CloudWatch metrics

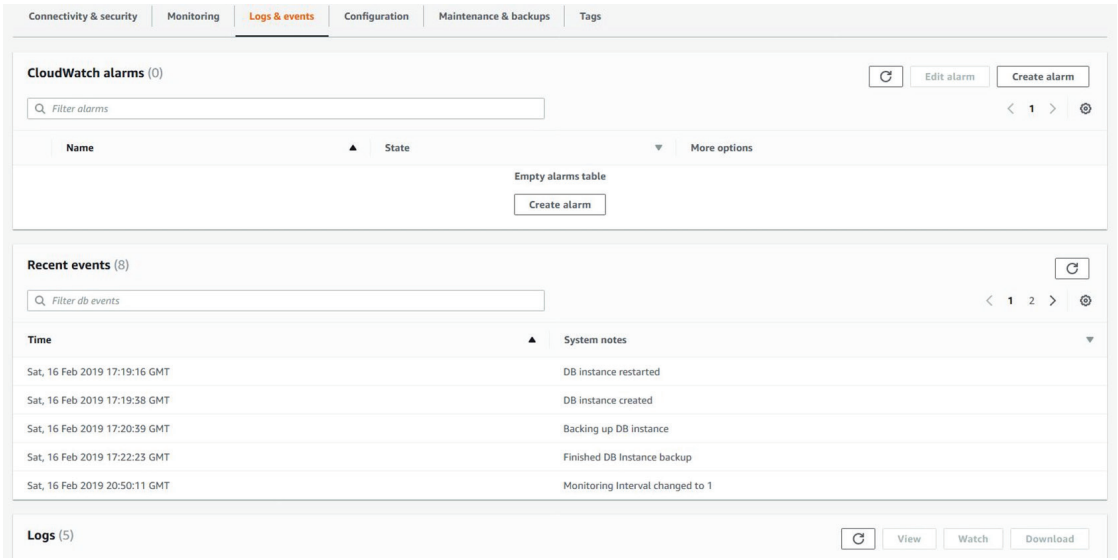


Amazon RDS integrates with CloudWatch to send it the following database logs:

- Audit log
- Error log
- General log
- Slow query log

From the Amazon RDS console, select the Logs & events tab to view and download the specified logs, as shown in Figure 4.7.

FIGURE 4.7 Amazon RDS with CloudWatch Logs



For more information on CloudWatch and its capabilities across other AWS services, see Chapter 15, "Monitoring and Troubleshooting."

Amazon Aurora

Amazon Aurora is a MySQL- and PostgreSQL-compatible relational database engine that combines the speed and availability of high-end commercial databases with the simplicity and cost-effectiveness of open source databases.

Aurora is part of the managed database service Amazon RDS.

Amazon Aurora DB Clusters

Aurora is a drop-in replacement for MySQL and PostgreSQL relational databases. It is built for the cloud, and it combines the performance and availability of high-end commercial databases with the simplicity and cost-effectiveness of open source databases. You can use the code, tools, and applications that you use today with your existing MySQL and PostgreSQL databases with Aurora.

The integration of Aurora with Amazon RDS means that time-consuming administration tasks, such as hardware provisioning, database setup, patching, and backups, are automated.

Aurora features a distributed, fault-tolerant, self-healing storage system that automatically scales up to 64 TiB per database instance. (In comparison, other Amazon RDS options allow for a maximum of 32 TiB.) Aurora delivers high performance and availability with up to 15 low-latency read replicas, point-in-time recovery, continuous backup to Amazon Simple Storage Service (Amazon S3), and replication across three Availability Zones. When you create an Aurora instance, you create a DB cluster. A *DB cluster* consists of one or more DB instances and a cluster volume that manages the data for those

instances. An Aurora *cluster volume* is a virtual database storage volume that spans multiple Availability Zones, and each Availability Zone has a copy of the DB cluster data.

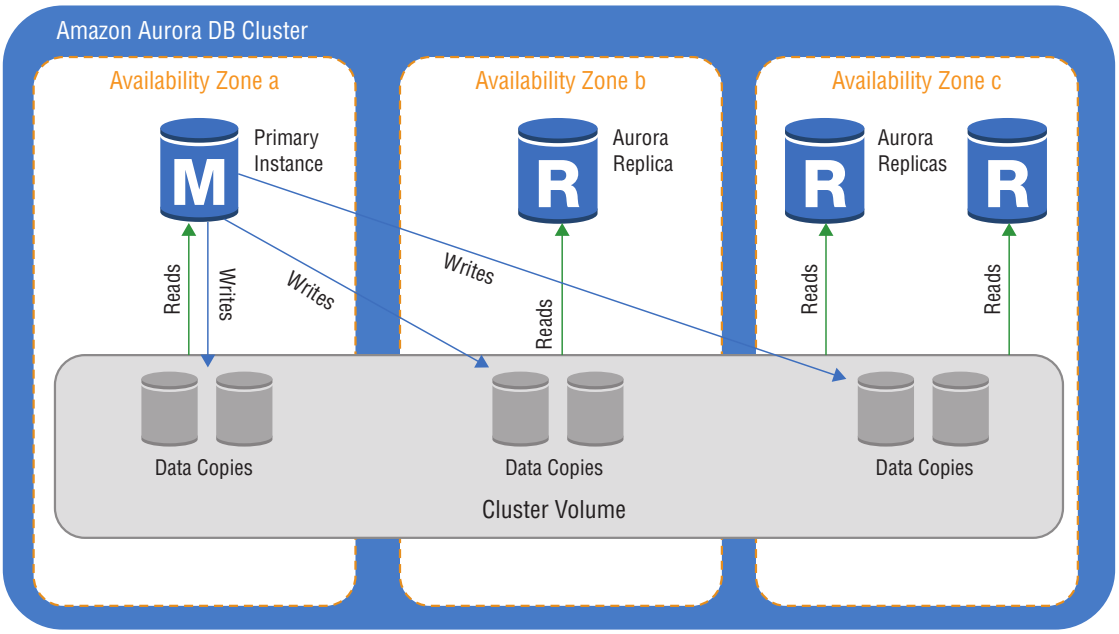
An Aurora DB cluster has two types of DB instances:

Primary Instance Supports read and write operations and performs all of the data modifications to the cluster volume. Each Aurora DB cluster has one primary instance.

Amazon Aurora Replica Supports read-only operations. Each Aurora DB cluster can have up to 15 *Amazon Aurora Replicas* in addition to the primary instance. Multiple Aurora Replicas distribute the read workload, and if you locate Aurora Replicas in separate Availability Zones, you can also increase database availability.

Figure 4.8 illustrates the relationship between the cluster volume, the primary instance, and Aurora Replicas in an Aurora DB cluster.

FIGURE 4.8 Amazon Aurora DB cluster



As you can see from Figure 4.8, this architecture is vastly different from the other Amazon RDS databases. Aurora is engineered and architected for the cloud. The primary difference is that there is a separate storage layer, called the *cluster volume*, which is spread across multiple Availability Zones in a single AWS Region. This means that the durability of your data is increased.

Additionally, Aurora has one primary instance that writes across the cluster volume. This means that Aurora replicas can be spun up quickly, because they don't have to copy and store their own storage layer; they connect to it. Because the cluster volume is separated in this architecture, the cluster volume can grow automatically as your data increases. This is in contrast to how other Amazon RDS databases are built, whereby you must define the allocated storage in advance.

Amazon Aurora Global Databases

With Aurora, you can also create a multiregional deployment for your database tier. In this configuration, the primary AWS Region is where your data is written (you may also do reads from the primary AWS Region). Any application performing writes must write to the primary AWS Region where the cluster is operating.

The secondary AWS Region is used for *reading* data only. Aurora replicates the data to the secondary AWS Region with typical latency of less than a second. Furthermore, you can use the secondary AWS Region for disaster recovery purposes. You can promote the secondary cluster and make it available as the primary typically in less than a minute. At the time of this writing, Aurora global databases are available in the following AWS Regions only:

- US East (N. Virginia)
- US East (Ohio)
- US West (Oregon)
- EU (Ireland)

Additionally, at the time of this writing, Aurora global databases are available only for MySQL 5.6.

Amazon Aurora Serverless

Aurora Serverless is an on-demand, automatic scaling configuration for Aurora. (It is available only for MySQL at the time of this writing.) With Aurora Serverless, the database will *automatically* start up, shut down, and scale capacity up or down based on your application's needs. This means that, as a developer, you can run your database in the AWS Cloud and not worry about managing any database instances.

Best Practices for Running Databases on AWS

The following are best practices for working with Amazon RDS:

Follow Amazon RDS basic operational guidelines. The Amazon RDS Service Level Agreement requires that you follow these guidelines:

- Monitor your memory, CPU, and storage usage. Amazon CloudWatch can notify you when usage patterns change or when you approach the capacity of your deployment so that you can maintain system performance and availability.
- Scale up your DB instance when you approach storage capacity limits. Have some buffer in storage and memory to accommodate unforeseen increases in demand from your applications.
- Enable automatic backups, and set the backup window to occur during the daily low in write IOPS.

- If your database workload requires more I/O than you have provisioned, recovery after a failover or database failure will be slow. To increase the I/O capacity of a DB instance, do any or all of the following:
  - Migrate to a DB instance class with high I/O capacity.
  - Convert from standard storage either to General Purpose or Provisioned IOPS storage, depending on how much of an increase you need. If you convert to Provisioned IOPS storage, make sure that you also use a DB instance class that is optimized for Provisioned IOPS.
  - If you are already using Provisioned IOPS storage, provision additional throughput capacity.
- 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.
- Test failover for your DB instance to understand how long the process takes for your use case and to ensure that the application that accesses your DB instance can automatically connect to the new DB instance after failover.

Allocate sufficient RAM to the DB instance. An Amazon RDS performance best practice is to allocate enough RAM so that your working set resides almost completely in memory. Check the `ReadIOPS` metric by using CloudWatch while the DB instance is under load to view the working set. The value of `ReadIOPS` should be small and stable. Scale up the DB instance class until `ReadIOPS` no longer drops dramatically after a scaling operation or when `ReadIOPS` is reduced to a small amount.

Implement Amazon RDS security. Use IAM accounts to control access to Amazon RDS API actions, especially actions that create, modify, or delete Amazon RDS resources, such as DB instances, security groups, option groups, or parameter groups, and actions that perform common administrative actions, such as backing up and restoring DB instances, or configuring Provisioned IOPS storage.

- Assign an individual IAM account to each person who manages Amazon RDS resources. Do not use an AWS account user to manage Amazon RDS resources; 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 manage permissions effectively for multiple users.
- Rotate your IAM credentials regularly.

Use the AWS Management Console, the AWS CLI, or the Amazon 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 permissions being revoked for the user unintentionally.

Use enhanced monitoring to identify OS issues. Amazon RDS provides metrics in real time for the OS on which your DB instance runs. You can view the metrics for your DB

instance by using the console or consume the Enhanced Monitoring JSON output from CloudWatch Logs in a monitoring system of your choice. Enhanced Monitoring is available for the following database engines:

- MariaDB
- Microsoft SQL Server
- MySQL version 5.5 or later
- Oracle
- PostgreSQL

Enhanced Monitoring is available for all DB instance classes except for `db.m1.small`. Enhanced Monitoring is available in all regions except for AWS GovCloud (US).

Use metrics to identify performance issues. To identify performance issues caused by insufficient resources and other common bottlenecks, you can monitor the metrics available for your Amazon RDS DB instance.

Monitor performance metrics regularly to see the average, maximum, and minimum values for a variety of time ranges. If you do so, you can identify when performance is degraded. You can also set CloudWatch alarms for particular metric thresholds.

To troubleshoot performance issues, it's important to understand the baseline performance of the system. When you set up a new DB instance and get it running with a typical workload, you should capture the average, maximum, and minimum values of all the performance metrics at a number of different intervals (for example, 1 hour, 24 hours, 1 week, or 2 weeks) to get an idea of what is normal. It helps to get comparisons for both peak and off-peak hours of operation. You can then use this information to identify when performance is dropping below standard levels.

Tune queries. One of the best ways to improve DB instance performance is to tune your most commonly used and most resource-intensive queries to make them less expensive to run.

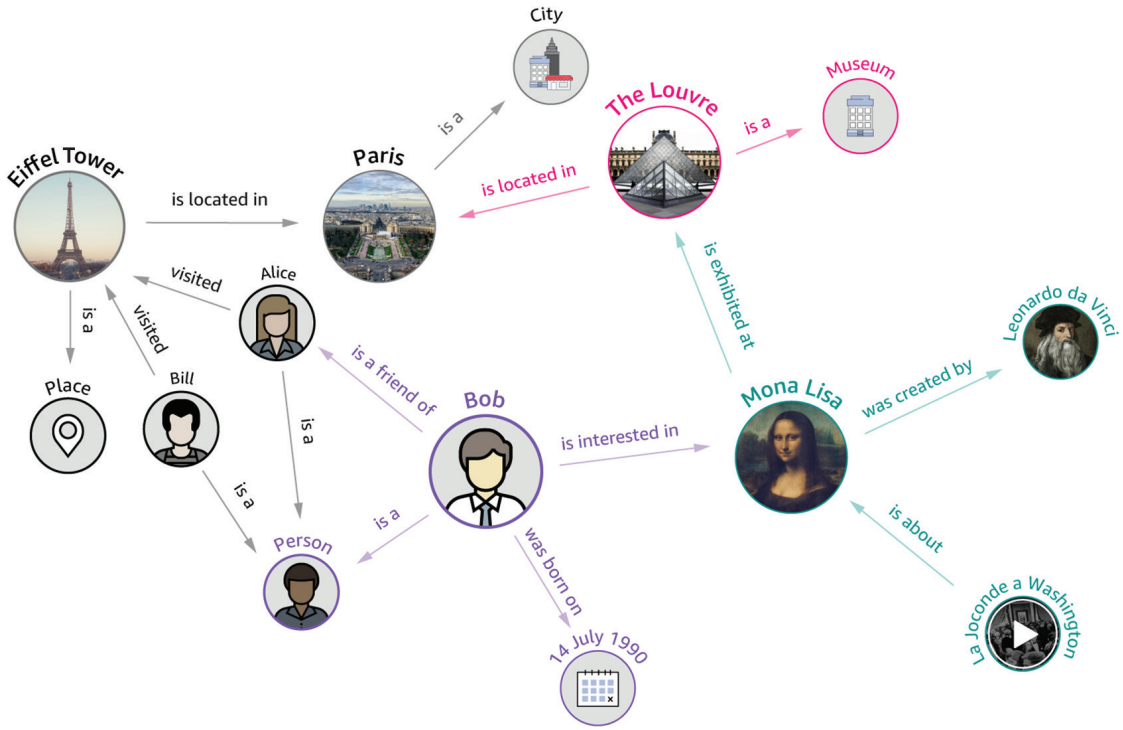
A common aspect of query tuning is creating effective indexes. You can use the Database Engine Tuning Advisor to get potential index improvements for your DB instance.

Use DB parameter groups. AWS recommends that you apply changes to the DB parameter group on a test DB instance before you apply parameter group changes to your production DB instances. Improperly setting DB engine parameters in a DB parameter group can have unintended adverse effects, including degraded performance and system instability. Always exercise caution when modifying DB engine parameters, and back up your DB instance before modifying a DB parameter group.

Use read replicas. Use read replicas to relieve pressure on your master node with additional read capacity. You can bring your data closer to applications in different regions and promote a read replica to a master for faster recovery in the event of a disaster.

You can use the AWS Database Migration Service (AWS DMS) to migrate or replicate your existing databases easily to Amazon RDS.

FIGURE 4.22 Example of a graph database architecture running on Amazon Neptune



Neptune supports the popular graph models Property Graph and W3C's RDS and their respective query languages Apache TinkerPop Gremlin and SPARQL. With these models, you can easily build queries that efficiently navigate highly connected datasets. Neptune graph databases include the following use cases:

- Recommendation engines
- Fraud detection
- Knowledge graphs
- Drug discovery
- Network security

Cloud Database Migration

Data is the cornerstone of successful cloud application deployments. Your evaluation and planning process may highlight the physical limitations inherent to migrating data from on-premises locations into the cloud. Amazon offers a suite of tools to help you move data via networks, roads, and technology partners.

This chapter focuses on the AWS Database Migration Service (AWS DMS) and the AWS Schema Conversion Tool (AWS SCT). Customers also use other AWS services and features

that are discussed in Chapter 3, "Hello, Storage," for cloud data migration, including the following:

- AWS Direct Connect (DX)
- AWS Snowball
- AWS Snowball Edge
- AWS Snowmobile
- AWS Import/Export Disk
- AWS Storage Gateway
- Amazon Kinesis Data Firehose
- Amazon S3 Transfer Acceleration
- Virtual private network (VPN) connections

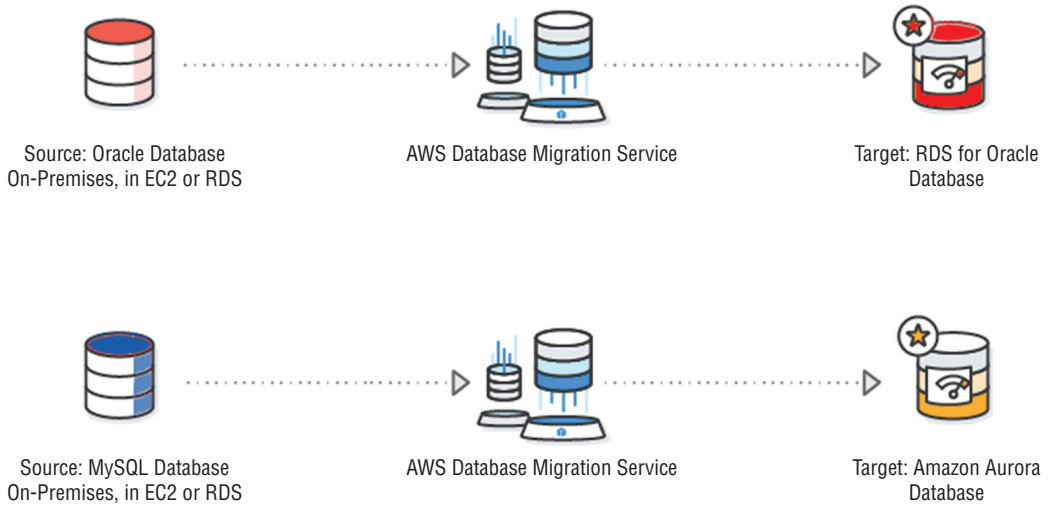
AWS Database Migration Service

AWS Database Migration Service (AWS DMS) helps you migrate databases to AWS quickly and securely. The source database remains fully operational during the migration, minimizing downtime to applications that rely on the database. AWS DMS can migrate your data to and from the most widely used commercial and open-source databases.

The service supports *homogenous database migrations*, such as Oracle to Oracle, in addition to *heterogeneous migrations* between different database platforms, such as Oracle to Amazon Aurora or Microsoft SQL Server to MySQL. You can also stream data to Amazon Redshift, Amazon DynamoDB, and Amazon S3 from any of the supported sources, such as Amazon Aurora, PostgreSQL, MySQL, MariaDB, Oracle Database, SAP ASE, SQL Server, IBM DB2 LUW, and MongoDB, enabling consolidation and easy analysis of data in a petabyte-scale data warehouse. You can also use AWS DMS for continuous data replication with high availability.

Figure 4.23 shows an example of both heterogeneous and homogenous database migrations.

FIGURE 4.23 Homogenous database migrations using AWS DMS



To perform a database migration, AWS DMS connects to the source data store, reads the source data, and formats the data for consumption by the target data store. It then loads the data into the target data store. Most of this processing happens in memory, though large transactions might require some buffering to disk. Cached transactions and log files are also written to disk.

At a high level, when you're using AWS DMS, complete the following tasks:

- Create a replication server.
- Create source and target endpoints that have connection information about your data stores.
- Create one or more tasks to migrate data between the source and target data stores.

A task can consist of three major phases:

- The full load of existing data
- The application of cached changes
- Ongoing replication

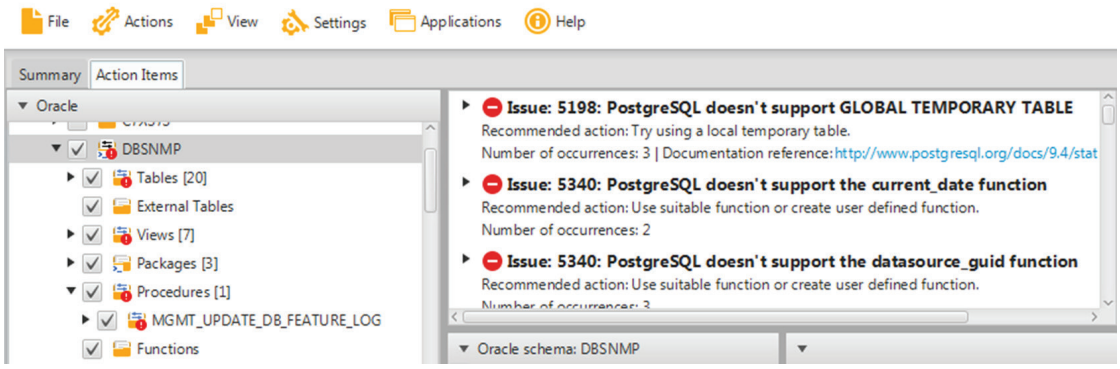
AWS Schema Conversion Tool

For heterogeneous database migrations, AWS DMS uses the *AWS Schema Conversion Tool* (AWS SCT). AWS SCT makes heterogeneous database migrations predictable by automatically converting the source database schema and a majority of the database code objects, including views, stored procedures, and functions, to a format compatible with the target database. Any objects that cannot be automatically converted are clearly marked so that they can be manually converted to complete the migration.

AWS SCT can also scan your application source code for embedded SQL statements and convert them as part of a database schema conversion project. During this process, AWS SCT performs cloud-native code optimization by converting legacy Oracle and SQL Server functions to their equivalent AWS service, thus helping you modernize the applications at the same time as database migration.

Figure 4.24 is snapshot of the Action Items tab in the AWS SCT report, which shows the items that the tool could not convert automatically. These are the items that you would need to evaluate and adjust manually as needed. The report helps you to determine how much work you would need to do to complete a conversion.

FIGURE 4.24 AWS SCT action items



After the schema conversion is complete, AWS SCT can help migrate data from a range of data warehouses to Amazon Redshift by using built-in data migration agents.

Your source database can be on-premises, in Amazon RDS, or in Amazon EC2, and the target database can be in either Amazon RDS or Amazon EC2. AWS SCT supports a number of different heterogeneous conversions. Table 4.9 lists the source and target databases that are supported at the time of this writing.

TABLE 4.9 Source and Target Databases Supported by AWS SCT

Source Database	Target Database on Amazon RDS
Oracle Database	Amazon Aurora, MySQL, PostgreSQL, Oracle
Oracle Data Warehouse	Amazon Redshift
Azure SQL	Amazon Aurora, MySQL, PostgreSQL
Microsoft SQL Server	Amazon Aurora, Amazon Redshift, MySQL, PostgreSQL
Teradata	Amazon Redshift
IBM Netezza	Amazon Redshift
IBM DB2 LUW	Amazon Aurora, MySQL, PostgreSQL
HPE Vertica	Amazon Redshift
MySQL and MariaDB	PostgreSQL
PostgreSQL	Amazon Aurora, MySQL
Amazon Aurora	PostgreSQL
Greenplum	Amazon Redshift
Apache Cassandra	Amazon DynamoDB

Running Your Own Database on Amazon Elastic Compute Cloud

This chapter focused heavily on the AWS services that are available from a managed database perspective. However, it is important to know that you can also run your own unmanaged database on Amazon EC2, not only for the exam but for managing projects in the real

world. For example, if you want to run MongoDB on Amazon EC2, this is perfectly within the realm of possibility. However, by doing so, you lose the many benefits of using a managed database service.

Compliance and Security

AWS includes various methods to provide security for your databases and meet the strictest of compliance standards. You can use the following:

- Network isolation through virtual private cloud (VPC)
- Security groups
- AWS resource-level permission controls that are IAM-based.
- Encryption at rest by using AWS KMS or Oracle/Microsoft Transparent Data Encryption (TDE)
- Secure Sockets Layer (SSL) protection for data in transit
- Assurance programs for finance, healthcare, government, and more

AWS Identity and Access Management

You can use *Identity and Access Management* (IAM) to perform governed access to control who can perform actions with Amazon Aurora MySQL and Amazon RDS for MySQL. Here's an example:

```
{
    "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:pg:default*",
                "arn:aws:rds: * : 1234 56789012 :subgrp: default"
            ],
```

```
            "Condition": {
                "StringEquals": {
                "rds:DatabaseEngine": "mysql",
                "rds:DatabaseClass": "db.t2.micro"
                }
            }
        }
    }
}
```

Summary

In this chapter, you learned the basic concepts of different types of databases, including relational, nonrelational, data warehouse, in-memory, and graph databases. From there, you learned about the various managed database services available on AWS. These included Amazon RDS, Amazon DynamoDB, Amazon Redshift, Amazon ElastiCache, and Amazon Neptune. You also saw how you can run your own database on Amazon EC2. Finally, you looked at how to perform homogenous database migrations using the AWS Database Migration Service (AWS DMS). For heterogeneous database migrations, you learned that AWS DMS can use the AWS Schema Conversion Tool (AWS SCT).

Exam Essentials

Know what a relational database is. A relational database consists of one or more tables. Communication to and from relational databases usually involves simple SQL queries, such as "Add a new record" or "What is the cost of product x?" These simple queries are often referred to as online transaction processing (OLTP).

Know what a nonrelational database is. Nonrelational databases do not have a hard-defined data schema. They can use a variety of models for data management, such as in-memory key-value stores, graph data models, and document stores. These databases are optimized for applications that have a large data volume, require low latency, and have flexible data models. In nonrelational databases, there is no concept of foreign keys.

Understand the database options available on AWS. You can run all types of databases on AWS. You should understand that there are managed and unmanaged options available, in addition to relational, nonrelational, caching, graph, and data warehouses.

Understand which databases Amazon RDS supports. Amazon RDS currently supports six relational database engines:

- Microsoft SQL Server
- MySQL
- Oracle
- PostgreSQL
- MariaDB
- Amazon Aurora

Understand the operational benefits of using Amazon RDS. Amazon RDS is an AWS managed service. AWS is responsible for patching, antivirus, and the management of the underlying guest OS for Amazon RDS. Amazon RDS greatly simplifies the process of setting a secondary slave with replication for failover and setting up read replicas to offload queries.

Remember that you cannot access the underlying OS for Amazon RDS DB instances. You cannot use Remote Desktop Protocol (RDP) or SSH to connect to the underlying OS. If you need to access the OS, install custom software or agents. If you want to use a database engine that Amazon RDS does not support, consider running your database on an Amazon EC2 instance instead.

Understand that Amazon RDS handles Multi-AZ failover for you. If your primary Amazon RDS instance becomes unavailable, AWS fails over to your secondary instance in another Availability Zone automatically. This failover is done by pointing your existing database endpoint to a new IP address. You do not have to change the connection string manually; AWS handles the DNS changes automatically.

Remember that Amazon RDS read replicas are used for scaling out and increased performance. This replication feature makes it easy to scale out your read-intensive databases. Read replicas are currently supported in Amazon RDS for MySQL, PostgreSQL, and Amazon Aurora. You can create one or more replicas of a database within a single AWS Region or across multiple AWS Regions. Amazon RDS uses native replication to propagate changes made to a source DB instance to any associated read replicas. Amazon RDS also supports cross-region read replicas to replicate changes asynchronously to another geography or AWS Region.

Know how to calculate throughput for Amazon DynamoDB. Remember that one read capacity unit (RCU) represents one strongly consistent read per second or two eventually consistent reads per second for an item up to 4 KB in size. For writing data, know that one write capacity unit (WCU) represents one write per second for an item up to 1 KB in size. Be comfortable performing calculations to determine the appropriate setting for the RCU and WCU for a table.

Know that DynamoDB spreads RCUs and WCUs across partitions evenly. Recall that when you allocate your total RCUs and WCUs to a table, DynamoDB spreads these across

your partitions evenly. For example, if you have 1,000 RCUs and you have 10 partitions, then you have 100 RCUs allocated to each partition.

Know the differences between a local secondary index and a global secondary index. Remember that you can create local secondary indexes only when you initially create the table; additionally, know that local secondary indexes must share the same partition key as the parent or source table. Conversely, you can create global secondary indexes at any time, with different partitions keys or sort keys.

Know the difference between eventually consistent and strongly consistent reads. Know that with eventually consistent reads, your application may retrieve data that is stale; but with strongly consistent reads, the data is always up-to-date.

Understand the purpose of caching data and which related services are available. Know why caching is important for your database tier and how it helps to improve your application performance. Additionally, understand the differences between the caching methods (lazy loading and write-through) and the corresponding AWS services (Amazon DynamoDB Accelerator (DAX), ElastiCache for Redis, and ElastiCache for Memcached).

Resources to Review

What Is a Relational Database?

`https://aws.amazon.com/relational-database/`

AWS Databases:

`https://aws.amazon.com/products/databases/`

AWS Database Blog:

`https://aws.amazon.com/blogs/database/`

A One Size Fits All Database Doesn't Fit Anyone:

`https://www.allthingsdistributed.com/2018/06/purpose-built-databases-in-aws.html`

Amazon Relational Database Service (Amazon RDS) User Guide:

`https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/Welcome.html`

Amazon RDS FAQs:

`https://aws.amazon.com/rds/faqs/`

Development and Test on Amazon Web Services:

`https://d1.awsstatic.com/whitepapers/aws-development-test-environments.pdf`

Amazon Redshift Snapshots:

`https://docs.aws.amazon.com/redshift/latest/mgmt/working-with-snapshots.html`

Amazon Aurora:

`https://aws.amazon.com/rds/aurora/`

Amazon Aurora Overview:

`https://docs.aws.amazon.com/AmazonRDS/latest/AuroraUserGuide/CHAP_AuroraOverview.html`

Amazon RDS Resources:

`https://aws.amazon.com/rds/developer-resources/`

Best Practices for Amazon RDS:

`https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/CHAP_BestPractices.html`

What Is a Document Database?

`https://aws.amazon.com/nosql/document/`

What Is a Columnar Database?

`https://aws.amazon.com/nosql/columnar/`

What Is NoSQL?

`https://aws.amazon.com/nosql/`

Amazon DynamoDB:

`https://aws.amazon.com/dynamodb/`

What Is Amazon DynamoDB?

`https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/Introduction.html`

Amazon DynamoDB Core Components:

`https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/HowItWorks.CoreComponents.html`

Amazon DynamoDB Developer Guide:

`http://docs.aws.amazon.com/amazondynamodb/latest/developerguide/`

GSI Attribute Projections:

`https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/GSI.html#GSI.Projections`

Data Warehouse Concepts:

`https://aws.amazon.com/data-warehouse/`

Getting Started with Amazon Redshift:

`http://docs.aws.amazon.com/redshift/latest/gsg/`

Amazon Redshift Database Developer Guide:

`http://docs.aws.amazon.com/redshift/latest/dg/`

Using Amazon Redshift Spectrum to Query External Data:

https://docs.aws.amazon.com/redshift/latest/dg/c-using-spectrum.html

What Is a Key-Value Database?

https://aws.amazon.com/nosql/key-value/

Amazon ElasticCache for Redis User Guide:

https://docs.aws.amazon.com/AmazonElastiCache/latest/red-ug/WhatIs.html

Amazon ElastiCache for Memcached User Guide:

https://docs.aws.amazon.com/AmazonElastiCache/latest/mem-ug/WhatIs.html

In-Memory Processing in the Cloud with Amazon ElastiCache (Whitepaper):

https://d1.awsstatic.com/elasticache/elasticache_in_memory_processing_intel.pdf

Performance at Scale with Amazon ElastiCache (Whitepaper):

https://d1.awsstatic.com/whitepapers/performance-at-scale-with-amazon-elasticache.pdf

Amazon DynamoDB Accelerator (DAX):

https://aws.amazon.com/dynamodb/dax/

Amazon DynamoDB Accelerator (DAX): A Read-Through/Write-Through Cache for DynamoDB:

https://aws.amazon.com/blogs/database/amazon-dynamodb-accelerator-dax-a-read-throughwrite-through-cache-for-dynamodb/

What Is a Graph Database?

https://aws.amazon.com/nosql/graph/

Amazon Neptune User Guide:

https://docs.aws.amazon.com/neptune/latest/userguide/intro.html

AWS Database Migration Service Documentation:

https://docs.aws.amazon.com/dms/index.html

Cloud Data Migration:

https://aws.amazon.com/cloud-data-migration/

AWS Database Migration Service User Guide:

http://docs.aws.amazon.com/dms/latest/userguide/

AWS Database Migration Service Step-by-Step Walkthroughs:

http://docs.aws.amazon.com/dms/latest/sbs/DMS-SBS-Welcome.html

AWS Schema Conversion Tool User Guide:

https://docs.aws.amazon.com/SchemaConversionTool/latest/userguide/CHAP_Welcome.html

Exercises

In the following exercises, you will launch two types of databases: the first database is an SQL database on Amazon RDS, and the second is Amazon DynamoDB (NoSQL). For these sets of exercises, you will use the Python 3 SDK. You can download the Python 3 SDK at `https://aws.amazon.com/sdk-for-python/`.

EXERCISE 4.1

Create a Security Group for the Database Tier on Amazon RDS

Before you can create your first Amazon RDS database, you must create a security group so that you can allow traffic from your development server to communicate with the database tier. To do this, you must use an Amazon EC2 client to create the security group. Security groups are a component of the Amazon EC2 service, even though you can use them as part of Amazon RDS to secure your database tier.

To create the security group, run the following script:

```
# Excercise 4.1
import boto3
import json
import datetime

# Let's create some variables we'll use throughout these Excercises in Chapter 4
# NOTE: Here we are using a CIDR range for incoming traffic. We have set it to
   0.0.0.0/0 which means
# ANYONE on the internet can access your database if they have the username and
   the password
# If possible, specify you're own CIDR range. You can figure out your CIDR range
   by visiting the following link
# https://www.google.com/search?q=what+is+my+ip
# In the variable don't forget to add /32!
# If you aren't sure, leave it open to the world

# Variables
sg_name = 'rds-sg-dev-demo'
sg_description = 'RDS Security Group for AWS Dev Study Guide'
my_ip_cidr = '0.0.0.0/0'


# Create the EC2 Client to create the Security Group for your Database
ec2_client = boto3.client('ec2')

# First we need to create a security group
```

```
response = ec2_client.create_security_group(
    Description=sg_description,
    GroupName=sg_name)
print(json.dumps(response, indent=2, sort_keys=True))
# Now add a rule for the security group
response = ec2_client.authorize_security_group_ingress(
    CidrIp=my_ip_cidr,
    FromPort=3306,
    GroupName=sg_name,
    ToPort=3306,
    IpProtocol='tcp'
    )
print("Security Group should be created! Verify this in the AWS Console.")
```

After running the Python code, verify that the security group was created successfully from the AWS Management Console. You can find this confirmation under the VPC or Amazon EC2 service.

EXERCISE 4.2

Spin Up the MariaDB Database Instance

Use the Python SDK to spin up your MariaDB database hosted on Amazon RDS.

To spin up the MariaDB database, run the following script and update the `Variables` section to meet your needs:

```
# Excercise 4.2
import boto3
import json
import datetime


# Just a quick helper function for date time conversions, in case you want to
   print the raw JSON
def date_time_converter(o):
    if isinstance(o, datetime.datetime):
        return o.__str__()

# Variables
sg_name = 'rds-sg-dev-demo'
rds_identifier = 'my-rds-db'
db_name = 'mytestdb'
```

(continued)

EXERCISE 4.2 *(continued)*

```
user_name = 'masteruser'
user_password = 'mymasterpassw0rd1!'
admin_email = 'myemail@myemail.com'
sg_id_number = ''
rds_endpoint = ''


# We need to get the Security Group ID Number to use in the creation of the RDS
   Instance
ec2_client = boto3.client('ec2')
response = ec2_client.describe_security_groups(
    GroupNames=[
        sg_name
    ])

sg_id_number = json.dumps(response['SecurityGroups'][0]['GroupId'])
sg_id_number = sg_id_number.replace('"','')

#  Create the client for Amazon RDS
rds_client = boto3.client('rds')

# This will create our MariaDB Database
# NOTE: Here we are hardcoding passwords for simplicity and testing purposes
   only! In production
# you should never hardcode passwords in configuration files/code!
# NOTE: This will create an MariaDB Database. Be sure to remove it when you are
   done.
response = rds_client.create_db_instance(
    DBInstanceIdentifier=rds_identifier,
    DBName=db_name,
    DBInstanceClass='db.t2.micro',
    Engine='mariadb',
    MasterUsername='masteruser',
    MasterUserPassword='mymasterpassw0rd1!',
    VpcSecurityGroupIds=[
        sg_id_number
    ],
    AllocatedStorage=20,
    Tags=[
        {
```

```
            'Key': 'POC-Email',
            'Value': admin_email
        },
        {
            'Key': 'Purpose',
            'Value': 'AWS Developer Study Guide Demo'
        }
    ]
)

# We need to wait until the DB Cluster is up!
print('Creating the RDS instance. This may take several minutes...')
waiter = rds_client.get_waiter('db_instance_available')
waiter.wait(DBInstanceIdentifier=rds_identifier)

print('Okay! The Amazon RDS Database is up!')
```

After the script has executed, the following message is displayed:

```
Creating the RDS instance. This may take several minutes.
```

After the Amazon RDS database instance has been created successfully, the following confirmation is displayed:

```
Okay! The Amazon RDS Database is up!
```

You can also view these messages from the Amazon RDS console.

EXERCISE 4.3

Obtain the Endpoint Value for the Amazon RDS Instance

Before you can start using the Amazon RDS instance, you must first specify your endpoint. In this exercise, you will use the Python SDK to obtain the value.

To obtain the Amazon RDS endpoint, run the following script:

```
# Exercise 4.3
import boto3
import json
import datetime


# Just a quick helper function for date time conversions, in case you want to
   print the raw JSON
```

(continued)

EXERCISE 4.3 *(continued)*

```
def date_time_converter(o):
    if isinstance(o, datetime.datetime):
        return o.__str__()

# Variables
rds_identifier = 'my-rds-db'

#  Create the client for Amazon RDS
rds_client = boto3.client('rds')

print("Fetching the RDS endpoint...")
response = rds_client.describe_db_instances(
    DBInstanceIdentifier=rds_identifier
)

rds_endpoint = json.dumps(response['DBInstances'][0]['Endpoint']['Address'])
rds_endpoint = rds_endpoint.replace('"','')
print('RDS Endpoint: ' + rds_endpoint)
```

After running the Python code, the following status is displayed:

```
Fetching the RDS endpoint.. RDS Endpoint:<endpoint_name>
```

If the endpoint is not returned, from the AWS Management Console, under the RDS service, verify that your Amazon RDS database instance was created.

EXERCISE 4.4

Create a SQL Table and Add Records to It

You now have all the necessary information to create your first SQL table by using Amazon RDS. In this exercise, you will create a SQL table and add a couple of records. Remember to update the variables for your specific environment.

To update the variables, run the following script:

```
# Exercise 4.4
import boto3
import json
import datetime
import pymysql as mariadb

# Variables
rds_identifier = 'my-rds-db'
```

```
db_name = 'mytestdb'
user_name = 'masteruser'
user_password = 'mymasterpassw0rd1!'
rds_endpoint = 'my-rds-db.****.us-east-1.rds.amazonaws.com'

# Step 1 - Connect to the database to create the table
db_connection = mariadb.connect(host=rds_endpoint, user=user_name,
   password=user_password, database=db_name)
cursor = db_connection.cursor()
try:
    cursor.execute("CREATE TABLE Users (user_id INT NOT NULL AUTO_INCREMENT,
    user_fname VARCHAR(100) NOT NULL, user_lname VARCHAR(150) NOT NULL, user_
    email VARCHAR(175) NOT NULL, PRIMARY KEY (`user_id`))")
    print('Table Created!')
except mariadb.Error as e:
    print('Error: {}'.format(e))
finally:
    db_connection.close()

# Step 2 - Connect to the database to add users to the table
db_connection = mariadb.connect(host=rds_endpoint, user=user_name,
password=user_password, database=db_name)
cursor = db_connection.cursor()
try:
    sql = "INSERT INTO `Users` (`user_fname`, `user_lname`, `user_email`) VALUES
    (%s, %s, %s)"
    cursor.execute(sql, ('CJ', 'Smith', 'casey.smith@somewhere.com'))
    cursor.execute(sql, ('Casey', 'Smith', 'sam.smith@somewhere.com'))
    cursor.execute(sql, ('No', 'One', 'no.one@somewhere.com'))
# No data is saved unless we commit the transaction!
    db_connection.commit()
    print('Inserted Data to Database!')
except mariadb.Error as e:
    print('Error: {}'.format(e))
    print('Sorry, something has gone wrong!')
finally:
    db_connection.close()
```

After running the Python code, the following confirmation is displayed:

```
Table Created! Inserted Data to the Database!
```

Your Amazon RDS database now has some data stored in it.

(continued)

EXERCISE 4.4 *(continued)*

In this exercise, you are hardcoding a password into your application code for demonstration purposes only. In a production environment, refrain from hard-coding application passwords. Instead, use services such as AWS Secrets Manager to keep your secrets secure.

EXERCISE 4.5

Query the Items in the SQL Table

After adding data to your SQL database, in this exercise you will be able to read or query the items in the *Users* table.

To read the items in the SQL table, run the following script:

```
# Exercise 4.5
import boto3
import json
import datetime
import pymysql as mariadb

# Variables
rds_identifier = 'my-rds-db'
db_name = 'mytestdb'
user_name = 'masteruser'
user_password = 'mymasterpassw0rd1!'
rds_endpoint = 'my-rds-db.*****.us-east-1.rds.amazonaws.com'

db_connection = mariadb.connect(host=rds_endpoint, user=user_name,
password=user_password, database=db_name)
cursor = db_connection.cursor()
try:
    sql = "SELECT * FROM `Users`"
    cursor.execute(sql)
    query_result = cursor.fetchall()
    print('Querying the Users Table...')
    print(query_result)
except mariadb.Error as e:
    print('Error: {}'.format(e))
    print('Sorry, something has gone wrong!')
```

```
finally:
    db_connection.close()
```

After running the Python code, you will see the three records that you inserted in the previous exercise.

EXERCISE 4.6

Remove Amazon RDS Database and Security Group

You've created an Amazon RDS DB instance and added data to it. In this exercise, you will remove a few resources from your account. Remove the Amazon RDS instance first.

To remove the Amazon RDS instance and the security group, run the following script:

```
# Exercise 4.6
import boto3
import json
import datetime

# Variables
rds_identifier = 'my-rds-db'
sg_name = 'rds-sg-dev-demo'
sg_id_number = ''

# Create the client for Amazon RDS
rds_client = boto3.client('rds')

# Delete the RDS Instance
response = rds_client.delete_db_instance(
    DBInstanceIdentifier=rds_identifier,
    SkipFinalSnapshot=True)

print('RDS Instance is being terminated...This may take several minutes.')

waiter = rds_client.get_waiter('db_instance_deleted')
waiter.wait(DBInstanceIdentifier=rds_identifier)

# We must wait to remove the security groups until the RDS database has been
   deleted, this is a dependency.
print('The Amazon RDS database has been deleted. Removing Security Groups')


# Create the client for Amazon EC2 SG
```

(continued)

EXERCISE 4.6 *(continued)*

```
ec2_client = boto3.client('ec2')

# Get the Security Group ID Number
response = ec2_client.describe_security_groups(
    GroupNames=[
        sg_name
    ])
sg_id_number = json.dumps(response['SecurityGroups'][0]['GroupId'])
sg_id_number = sg_id_number.replace('"','')

# Delete the Security Group!
response = ec2_client.delete_security_group(
    GroupId=sg_id_number
    )

print('Cleanup is complete!')
```

After running the Python code, the following message is displayed:

```
Cleanup is complete!
```

The Amazon RDS database and the security group are removed. You can verify this from the AWS Management Console.

EXERCISE 4.7

Create an Amazon DynamoDB Table

Amazon DynamoDB is a managed NoSQL database. One major difference between DynamoDB and Amazon RDS is that DynamoDB doesn't require a server that is running in your VPC, and you don't need to specify an instance type. Instead, create a table.

To create the table, run the following script:

```
# Exercise 4.7
import boto3
import json
import datetime
```

```
dynamodb_resource = boto3.resource('dynamodb')

table = dynamodb_resource.create_table(
    TableName='Users',
    KeySchema=[
        {
            'AttributeName': 'user_id',
            'KeyType': 'HASH'
        },
        {
            'AttributeName': 'user_email',
            'KeyType': 'RANGE'
        }
    ],
    AttributeDefinitions=[
        {
            'AttributeName': 'user_id',
            'AttributeType': 'S'
        },
        {
            'AttributeName': 'user_email',
            'AttributeType': 'S'
        }
    ],
    ProvisionedThroughput={
        'ReadCapacityUnits': 5,
        'WriteCapacityUnits': 5
    }
)

print("The DynamoDB Table is being created, this may take a few minutes...")
table.meta.client.get_waiter('table_exists').wait(TableName='Users')
print("Table is ready!")
```

After running the Python code, the following message is displayed:

```
Table is ready!
```

From the AWS Management Console, under DynamoDB, verify that the table was created.

EXERCISE 4.8

Add Users to the Amazon DynamoDB Table

With DynamoDB, there are fewer components to set up than there are for Amazon RDS. In this exercise, you'll add users to your table. Experiment with updating and changing some of the code to add multiple items to the database.

To add users to the DynamoDB table, run the following script:

```
# Exercise 4.8
import boto3
import json
import datetime
# In this example we are not using uuid; however, you could use this to
autogenerate your user IDs.
# i.e. str(uuid.uuid4())
import uuid

# Create a DynamoDB Resource
dynamodb_resource = boto3.resource('dynamodb')
table = dynamodb_resource.Table('Users')

# Write a record to DynamoDB
response = table.put_item(
    Item={
        'user_id': '1234-5678',
        'user_email': 'someone@somewhere.com',
        'user_fname': 'Sam',
        'user_lname': 'Samuels'
    }
)

# Just printing the raw JSON response, you should see a 200 status code
print(json.dumps(response, indent=2, sort_keys=True))
```

After running the Python code, you receive a 200 HTTP Status Code from AWS. This means that the user record has been added.

From the AWS Management Console, under DynamoDB, review the table to verify that the user record was added.

is cached and routed through the closest edge location serving you. After you deploy your application on Elastic Beanstalk, use the Amazon CloudFront content delivery network (CDN) to cache static content from your application. To identify the source of your content in Amazon CloudFront, you can use URL path patterns to cache your content and then retrieve it from the cache. This approach serves your content more rapidly and offloads requests directly sourced from your application.

AWS Config

With *AWS Config*, you can visualize configuration history and how configurations evolve over time. Tracking changes helps you to fulfill compliance obligations and meet auditing requirements. You can integrate AWS Config directly with your application and its versions or your Elastic Beanstalk environment. *You can customize AWS Config to record changes per resource, per region, or globally.* In the AWS Config console, you can select Elastic Beanstalk resource types to record specific applications and environment resources. You can view the recorded information in the AWS Config dashboard under Resource Inventory.

Amazon RDS

Various options are available for creating databases for your environment, such as Amazon Relational Database Service (Amazon RDS) for SQL databases and Amazon DynamoDB for NoSQL databases. Elastic Beanstalk can create a database and store and retrieve data for any of your environments. Each service has its own features to handle scaling, capacity, performance, and availability.

To store, read, or write to your data records, you can set up an Amazon RDS database instance or an Amazon DynamoDB table by using the same configuration files for your other service option settings. You must create connections to the database, which require you to set up password management in Elastic Beanstalk. Your configurations are saved in the `ebextensions` directory. You can also create direct connections, within your application code or application configuration files, to both internal and external databases. When using Amazon RDS, avoid accidentally deleting and re-creating databases without a properly installed backup. To reduce the risk of losing data, take a manual snapshot of the master Amazon RDS database immediately before deleting.

If you create periodic tasks with a worker environment, Elastic Beanstalk automatically creates an Amazon DynamoDB table to perform leader election and stores task information.

Amazon ElastiCache

For caching capabilities, you can integrate Amazon ElastiCache service clusters with the Elastic Beanstalk environment. If you use a nonlegacy container, you can set your configuration files to use the supported container and then offload requests to the cache cluster.

"aware" of each other. For example, if a layer in your stack installs and configures `haproxy` for load balancing, any instances in the same layer will need to update to include the new node in `/etc/hosts` (or remove the node that went offline).

Deploy

After an instance has come online and completes the initial `Setup` and `Configure` events, a `Deploy` event deploys any apps that you configure for the layer. This step can copy application code from a repository, start or refresh services, and perform other tasks to bring your application(s) online.

After an instance has run `Deploy` for the first time, it will never do so again automatically. This prevents untested changes from reaching production instances. After you test a feature change or bug fix, you must manually run the `Deploy` event with the AWS OpsWorks Stacks console or AWS CLI.

Undeploy

The `Undeploy` lifecycle event runs when you delete or remove an app from a layer. You use this to perform tasks such as when you want to remove an application's configuration or other cleanup tasks when you remove an app.

Shutdown

Before the actual shutdown command issues to an instance, the `Shutdown` lifecycle event gives you the opportunity to perform tasks such as taking snapshots and copying log files to Amazon S3 for later use. If the instance's layer also includes a load balancer, the instance deregisters after the configured connection draining time.

Resource Management

AWS OpsWorks Stacks allows for management of other resources in your account as part of your stack, and it includes elastic IP addresses, Amazon EBS volumes, and Amazon RDS instances. You register the resources with the stack to make them available to assign them to instances or layers. If you attach resources to instances in the stack and you delete the instance, the resource remains registered with the stack until it is manually deregistered. Deregistering resources does not automatically delete them. You must delete the resource itself with the respective service console or AWS CLI command.

Amazon EBS Volumes

Amazon EBS volumes that are not currently attached to any instances can register with a stack, and you can assign them to instances if the volume uses XFS formatting. You cannot attach volumes to running instances. To attach a volume to a running instance, you must stop it. You can move a volume between instances that are both offline.

You cannot attach Amazon EBS volumes to Windows stacks.

Elastic IP Addresses

As with Amazon EBS volumes, elastic IP addresses that are not associated with resources in your account may be registered with the stack. You can assign an elastic IP address to an instance regardless of whether it is running or not. After an Elastic IP address is disassociated from an instance, a configure lifecycle event updates instances in the stack with the instance's new IP address.

Amazon RDS Instances

You can register Amazon EBS instances to only one stack at a time. However, you can register a single Amazon RDS instance with multiple apps in the same stack.

Chef 11 and Chef 12

Both Chef 11 and Chef 12 provide unique functionality differences that are important to note before you use AWS OpsWorks Stacks. Each of the major differences is outlined in this section.

The differences in this section are with respect to AWS OpsWorks Stacks as a service, and they do not include differences between Chef versions 11.10 and 12.0. Since version 11.10 has been deprecated by Chef, community support will not be as strong as for later versions.

Separate Chef Runs

In Chef 11.10 stacks, AWS-provided cookbooks were run in the same Chef run as any custom cookbooks. The AWS cookbooks performed various tasks such as mounting Amazon EBS volumes that had been attached to the instance in the AWS OpsWorks console. However, this could result in situations where custom cookbooks had naming conflicts with those provided by AWS. You had to split this into two separate Chef runs on the instance to eliminate any potential namespace conflicts.

Community Support

Since the deprecation of Chef 11.10, community support has gradually decreased. Any open source cookbooks on the Chef Supermarket, for example, will likely make use of Chef 12.0 functionality, removing backward compatibility for Chef 11.10 stacks.

Built-in Layers

Chef 12.0 stacks no longer include the built-in layers as in Chef 11.10 stacks, such as the Rails layer. To implement these layers in Chef 12.0 stacks, you can still copy the built-in cookbooks from a Chef 11.10 stack and update them to be compatible with Chef 12.0. Chef 12.0 stacks still support built-in layer types from Chef 11.10 stacks.

- Amazon RDS instance layers
- Amazon ECS cluster layers

Berkshelf

Berkshelf is no longer available for the automatic installation on Chef 12.0 instances. Instead, install Berkshelf with a custom cookbook.

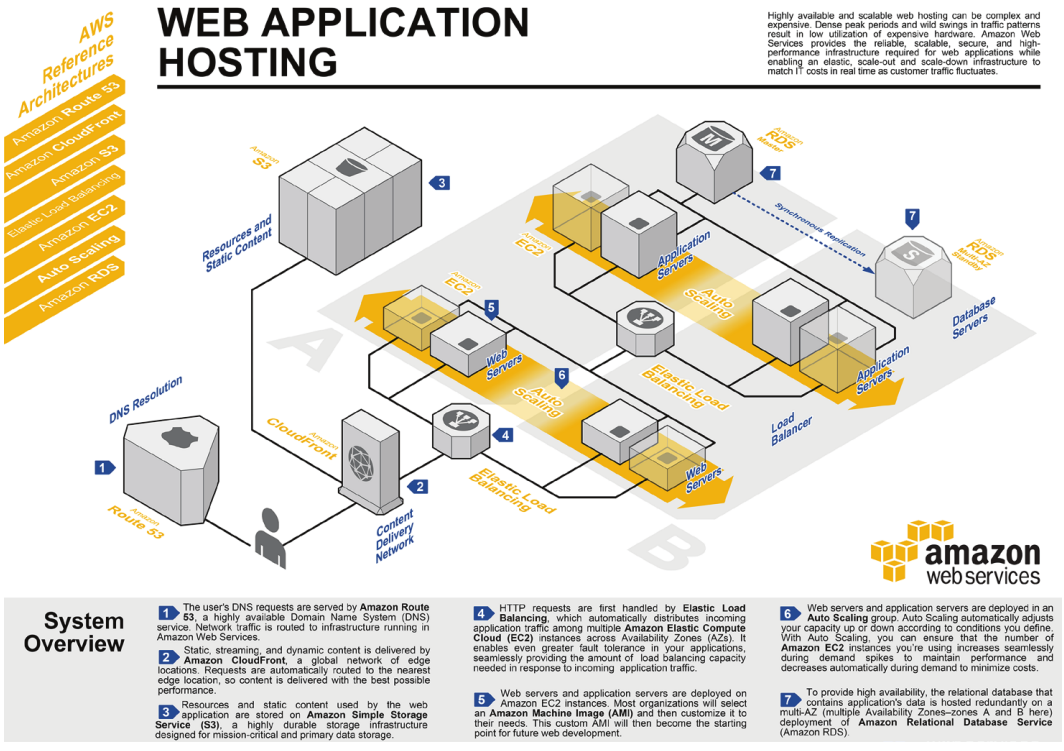
In addition to using the higher-level mobile and JavaScript SDKs, you can also use the lower-level APIs available via the following AWS SDKs to integrate all Amazon Cognito functionality in your applications:

- Java SDK
- .NET SDK
- Node.js SDK
- Python SDK
- PHP SDK
- Ruby SDK

Standard Three-Tier vs. the Serverless Stack

This chapter has introduced serverless services and their benefits. Now that you know about some of the serverless services that are available in AWS, let's compare a traditional three-tier application against a serverless application architecture. Figure 13.5 shows a typical three-tier web application.

FIGURE 13.5 Standard three-tier web infrastructure architecture



Source: https://media.amazonwebservices.com/architecturecenter/AWS_ac_ra_web_01.pdf

This architecture uses the following components and services:

Routing: Amazon Route 53

Content distribution network (CDN): Amazon CloudFront

Static data: Amazon S3

High availability/decoupling: Application load balancers

Web servers: Amazon EC2 with Auto Scaling

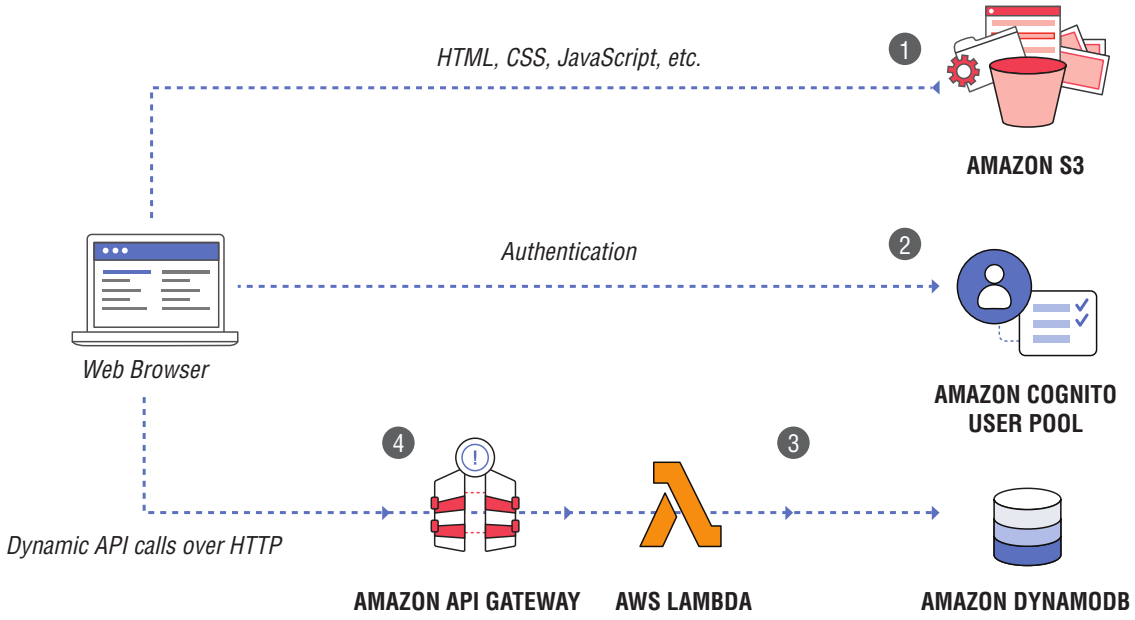
App servers: Amazon EC2 with Auto Scaling

Database: Amazon RDS in a multi-AZ configuration

Amazon Route 53 provides a DNS service that allows you to take domain names such as `examplecompany.com` and translate them to an IP address that points to running servers.

The CDN shown in Figure 13.6 is the Amazon CloudFront service, which improves your site performance with the use of its global content delivery network.

FIGURE 13.6 Serverless web application architecture



Source: `https://aws.amazon.com/getting-started/projects/build-serverless-web-app-lambda-apigateway-s3-dynamodb-cognito/`

Amazon S3 stores your static files such as photos or movie files.

Application load balancers are responsible for distributing load across Availability Zones to your Amazon EC2 servers, which run your web application with a service such as Apache or NGINX.

Application servers are responsible for performing business logic prior to storing the data in your database servers that are run by Amazon RDS.

Amazon RDS is the managed database server, and it can run an Amazon Aurora, Microsoft SQL Server, Oracle SQL Server, MySQL, PostgreSQL, or MariaDB database server.

While this architecture is a robust and highly available service, there are several downsides, including the fact that you have to manage servers. You are responsible for patching those servers, preventing downtime associated with those patches, and proper server scaling.

In a typical serverless web application architecture, you also run a web application, but you have zero servers that run inside your AWS account, as shown in Figure 13.6.

Serverless web application architecture services include the following:

Routing: Amazon Route 53

Web servers/static data: Amazon S3

User authentication: Amazon Cognito user pools

App servers: Amazon API Gateway and AWS Lambda

Database: Amazon DynamoDB

Amazon Route 53 is your DNS, and you can use Amazon CloudFront for your CDN.

You can also use Amazon S3 for your web servers. In this architecture, you use Amazon S3 to host your entire static website. You use JavaScript to make API calls to the Amazon API Gateway service.

For your business or application servers, you use Amazon API Gateway in conjunction with AWS Lambda. This allows you to retrieve and save data dynamically.

You use Amazon DynamoDB as a serverless database service, and you do not provision any Amazon EC2s inside of your Amazon VPC account. Amazon DynamoDB is also a great database service for storing session state for stateful applications. You can use Amazon RDS instead if you need a relational database. However, it would not then be a fully serverless stack. There is a new service released called *Amazon Aurora Serverless*, which is a full RDS MySQL 5.6–compatible service that is completely serverless. This would allow you to run a traditional SQL database, but one that has the benefit of being serverless. Amazon Aurora Serverless is discussed in the next section.

You use Amazon Cognito user pools for user authentication, which provides a secure user directory that can scale to hundreds of millions of users. Amazon Cognito User Pools is a fully managed service with no servers for you to manage. While user authentication was not shown in Figure 13.6, you can use your web server tier to talk to a user directory, such as *Lightweight Directory Access Protocol* (LDAP), for user authentication.

As you can see, while some of the components are the same, you may use them in slightly different ways. By taking advantage of the AWS global network, you can develop fully scalable, highly available web applications—all without having to worry about maintaining or patching servers.

Amazon Aurora Serverless

Amazon Aurora Serverless is an on-demand, auto-scaling configuration for the Aurora MySQL-compatible edition, where the database automatically starts, shuts down, and scales up or down as needed by your application. This allows you to run a traditional SQL database in the cloud without needing to manage any infrastructure or instances.

With Amazon Aurora Serverless, you also get the same high availability as traditional Amazon Aurora, which means that you get six-way replication across three Availability Zones inside of a region in order to prevent against data loss.

Amazon Aurora Serverless is great for infrequently used applications, new applications, variable workloads, unpredictable workloads, development and test databases, and multitenant applications. This is because you can scale automatically when you need to and scale down when application demand is not high. This can help cut costs and save you the heartache of managing your own database infrastructure.

Amazon Aurora Serverless is easy to set up, either through the console or directly with the CLI. To create an Amazon Aurora Serverless cluster with the CLI, you can run the following command:

```
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 user-name --master-user-password password \
--db-subnet-group-name mysubnetgroup --vpc-security-group-ids sg-c7e5b0d2
-region us-east-1
```

Amazon Aurora Serverless gives you many of the similar benefits as other serverless technologies, such as AWS Lambda, but from a database perspective. Managing databases is hard work, and with Amazon Aurora Serverless, you can utilize a database that automatically scales and you don't have to manage any of the underlying infrastructure.

AWS Serverless Application Model

The *AWS Serverless Application Model* (AWS SAM) allows you to create and manage resources in your serverless application with *AWS CloudFormation* to define your serverless application infrastructure as a SAM template. A *SAM template* is a JSON or YAML configuration file that describes the AWS Lambda functions, API endpoints, tables, and other resources in your application. With simple commands, you upload this template to AWS CloudFormation, which creates the individual resources and groups them into an *AWS CloudFormation stack* for ease of management. When you update your AWS SAM template, you re-deploy the changes to this stack. AWS CloudFormation updates the individual resources for you.

AWS SAM is an extension of AWS CloudFormation. You can define resources by using the AWS CloudFormation in your AWS SAM template. This is a powerful feature, as you can use AWS SAM to create a template of your serverless infrastructure, which you can then build into a DevOps pipeline. For example, examine the following:

```
AWSTemplateFormatVersion: '2010-09-09'
Transform: 'AWS::Serverless-2016-10-31'
Description: 'Example of Multiple-Origin CORS using API Gateway and Lambda'
```

JON BONSO AND KENNETH SAMONTE

AWS CERTIFIED

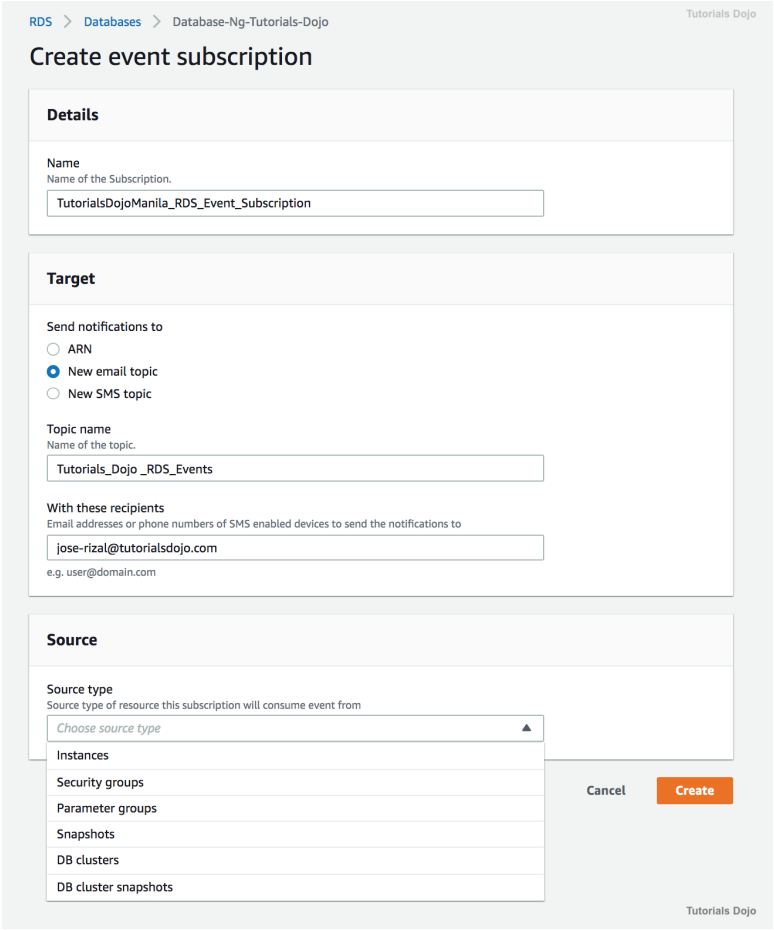
DEVOPS ENGINEER PROFESSIONAL

Tutorials Dojo
Study Guide and Cheat Sheets

Using default IAM policies for AWSCodeCommitPowerUser but must be limited to a specific repository only	Attach additional policy with Deny rule and custom condition if it does not match the specific repository or branch
You need to secure an S3 bucket by ensuring that only HTTPS requests are allowed for compliance purposes.	Create an S3 bucket policy that Deny if checks for condition aws:SecureTransport is **false**
Need to store a secret, database password, or variable, in the most cost-effective solution	Store the variable on SSM Parameter Store and enable encryption
Need to generate a secret password and have it rotated automatically at regular intervals	Store the secret on AWS Secrets Manager and enable key rotation.
Several team members, with designated roles, need to be granted permission to use AWS resources	Assign AWS managed policies on the IAM accounts such as, ReadOnlyAccess, AdministratorAccess, PowerUserAccess
Apply latest patches on EC2 and automatically create an AMI	Use Systems Manager automation to execute an Automation Document that installs OS patches and creates a new AMI.
Need to have a secure SSH connection to EC2 instances and have a record of all commands executed during the session	Install SSM Agent on EC2 and use SSM Session Manager for the SSH access. Send the session logs to S3 bucket or CloudWatch Logs for auditing and review.
Ensure that the managed EC2 instances have the correct application version and patches installed.	Use SSM Inventory to have a visibility of your managed instances and identify their current configurations.
Apply custom patch baseline from a custom repository and schedule patches to managed instances	Use SSM Patch Manager to define a custom patch baseline and schedule the application patches using SSM Maintenance Windows
Incident and Event Response	
Need to get a notification if somebody deletes files in your S3 bucket	Setup Amazon S3 Event Notifications to get notifications based on specified S3 events on a particular bucket.
Need to be notified when an RDS Multi-AZ failover happens	Setup Amazon RDS Event Notifications to detect specific events on RDS.

Amazon RDS Event Notifications

A database is a critical component of any enterprise system; therefore, it should be appropriately monitored for any potential issues. Like Amazon S3, there is also an event notification feature in Amazon RDS that notifies you of occurrences in your database resources. You can configure RDS to send you updates if an RDS DB instance has been created, restarted, or deleted. The RDS Event notifications can also detect low storage, configuration changes, Multi-AZ failover events, and many more.



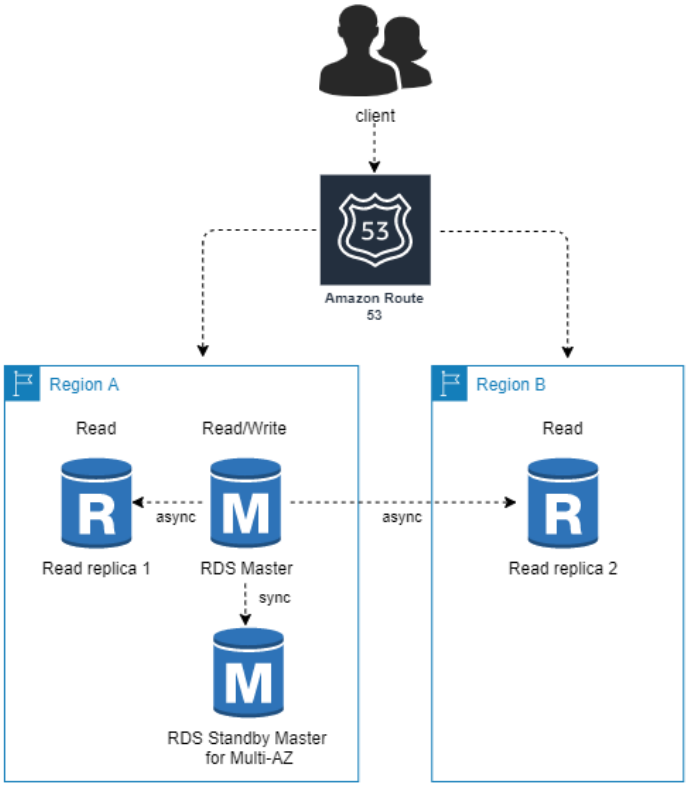
Amazon RDS produces numerous events in specific categories that you can subscribe to using various tools such as the AWS CLI, Amazon RDS Console, or the RDS API. Each event category can refer to the parameter group, snapshot, or security group of your DB instance. Moreover, you can automatically process your RDS event notifications by using an AWS Lambda function or set an alarm threshold that tracks specific metrics by creating a CloudWatch Alarm.

Source:
https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/USER_Events.html

Amazon RDS Disaster Recovery Strategies

AWS provides a plethora of database services and database engines that you can use for your applications. If you need a relational database type, you can use Amazon Aurora, Amazon RDS, or Amazon Redshift. For key-value data store, you can use an Amazon DynamoDB table that you can upgrade to DynamoDB Global to support your users worldwide. Amazon ElastiCache is perfect for in-memory data stores where you can use a Redis or Memcached engine. You can choose many other database services for document, wide column, graph, time series, and ledger type databases.

Amazon Aurora is a fully managed MySQL and PostgreSQL-compatible relational database that provides high performance, availability, and scalability to your applications. Since it is a fully managed service, Amazon handles all of the underlying resources in your Aurora database and ensures that your cluster is highly available to meet your disaster recovery objectives and achieves fault tolerance. Aurora is excellent, but it has certain limitations, which compels companies to choose Amazon RDS as their database tier. Aurora does not use a native MySQL or PostgreSQL engine like RDS and can't directly run Oracle and Microsoft SQL Server databases unless you migrate them using the AWS Database Migration Service (AWS DMS) and AWS Schema Conversion Tool (AWS SCT). These constraints are the reasons why thousands of companies are still using Amazon RDS in their cloud architecture.



Amazon RDS is a managed database service. Unlike its "fully managed" counterparts, AWS does not entirely 'manage' or control all of the components of an Amazon RDS database compared to what it does for Amazon Aurora. If you launched an RDS database, you are responsible for making it highly scalable and highly available by deploying Read Replicas or using Multi-AZ Deployments configurations. You can also improve the data durability of your database-tier by taking automated or manual snapshots in RDS. For disaster recovery planning, you can set up a disaster recovery (DR) site to another AWS Region if the primary region becomes unavailable.

	DISASTER RECOVERY		COST	SCOPE
	RTO	RPO		
AUTOMATED BACKUPS	GOOD	BETTER	LOW	SINGLE REGION
MANUAL SNAPSHOTS	BETTER	GOOD	MEDIUM	CROSS-REGION
READ REPLICAS	BEST	BEST	HIGH	CROSS-REGION

An RDS Read Replica is mainly used to vertically scale your application by offloading the read requests from your primary DB instance. But do you know that it is tremendously useful for disaster recovery too? It uses asynchronous replication to mirror all the changes from the primary instance to the replica, located on the same or a different AWS Region. In contrast, the Multi-AZ Deployments configuration uses synchronous replication to keep its standby instance up-to-date. As its name implies, the standby instance is just on *standby*, meaning it neither accepts read nor write requests. This standby instance can only run on the same AWS Region, unlike a Read Replica with a cross-region capability. These unique attributes enable the Read Replica to provide the best RTO and RPO for your disaster recovery plan. You can deploy a Read Replica of your RDS database to another AWS Region to expedite the application failover if the primary region becomes unavailable without having to wait for hours to migrate and launch the automated/manual RDS snapshots to the other region.

You should also know the difference between automated backups, manual snapshots, and Read Replicas for your Business Continuity Plan (BCP). Amazon RDS has a built-in automated backups feature that regularly takes snapshots of your database and stores it on an Amazon S3 bucket that is owned and managed by AWS. The retention period of these backups vary between 0 and 35 days. It provides a low-cost DR solution for your database-tier but is only limited to a single AWS Region. Manual snapshots are the ones that you manually take

yourself, hence the name. In contrast with the automated backups, the S3 bucket where the snapshots are stored is owned by you, which means that you can control its retention period and deploy cross-region snapshots. Since you manage your own RDS snapshots, you can move these across AWS Regions using a shell script or a Lambda function run by CloudWatch Events regularly.

The advantage of using Read Replicas over automated backups and manual snapshots is its near real-time synchronous replication. To put it into perspective, the replication time of the primary DB instance to the replica instance is less than a second! Compare that to the time required to move an RDS snapshot across another region and waiting for it to start up. Hence, Read Replicas provide the fastest RTO and the best RPO for your architecture. The only setback is its high cost since you have to run your replica continuously.

Sources:
https://aws.amazon.com/blogs/database/implementing-a-disaster-recovery-strategy-with-amazon-rds/
https://d0.awsstatic.com/whitepapers/Backup_and_Recovery_Approaches_Using_AWS.pdf
https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/USER_CopySnapshot.html

AWS Database Services

Amazon Aurora

- A fully managed relational database engine that's compatible with MySQL and PostgreSQL.
- With some workloads, Aurora can deliver up to five times the throughput of MySQL and up to three times the throughput of PostgreSQL.
- Aurora includes a high-performance storage subsystem. The underlying storage grows automatically as needed, up to 64 terabytes. The minimum storage is 10GB.
- **DB Clusters**
    - An 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 AZs, with each AZ having a copy of the DB cluster data.
    - Cluster Types:
        - 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. 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.
- **Aurora Multi Master**
    - The feature is available on Aurora MySQL 5.6
    - Allows you to create multiple read-write instances of your Aurora database across multiple Availability Zones, which enables uptime-sensitive applications to achieve continuous write availability through instance failure.
    - In the event of instance or Availability Zone failures, Aurora Multi-Master enables the Aurora database to maintain read and write availability with zero application downtime. There is no need for database failovers to resume write operations.
- **Monitoring**
    - 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.
    - Database log files
    - RDS Enhanced Monitoring — Look at metrics in real time for the operating system.
    - RDS Performance Insights monitors your Amazon RDS DB instance load so that you can analyze and troubleshoot your database performance.
    - Use CloudWatch Metrics, Alarms and Logs

Amazon RDS

- Supports **Aurora**, **MySQL, MariaDB, PostgreSQL, Oracle, Microsoft SQL Server**.
- You can get high availability with a primary instance and a synchronous secondary instance that you can fail over to when problems occur. You can also use MySQL, MariaDB, or PostgreSQL Read Replicas to increase read scaling.
- You can select the computation and memory capacity of a DB instance, determined by its **DB instance class**. If your needs change over time, you can change DB instances.
- Each DB instance has minimum and maximum storage requirements depending on the storage type and the database engine it supports.
- You can run your DB instance in several AZs, an option called a **Multi-AZ deployment**. Amazon automatically provisions and maintains a secondary standby DB instance in a different AZ. Your primary DB instance is synchronously replicated across AZs to the secondary instance to provide data redundancy, failover support, eliminate I/O freezes, and minimize latency spikes during system backups.

Security

- Security Groups
  - **DB Security Groups** - controls access to a DB instance that is not in a VPC. By default, network access is turned off to a DB instance. This SG is for the EC2-Classic platform.
  - **VPC Security Groups** - controls access to a DB instance inside a VPC. This SG is for the EC2-VPC platform.
  - **EC2 Security Groups** - controls access to an EC2 instance and can be used with a DB instance.
- A *resource owner* is the AWS account that created a resource. That is, the resource owner is the AWS account of the *principal entity* (the root account, an IAM user, or an IAM role) that authenticates the request that creates the resource.
- A *permissions policy* describes who has access to what. Policies attached to an IAM identity are *identity-based policies* (IAM policies) and policies attached to a resource are *resource-based policies*. Amazon RDS supports only identity-based policies (IAM policies).
- MySQL and PostgreSQL both support **IAM database authentication**.

High Availability using Multi-AZ

- Multi-AZ deployments for **Oracle, PostgreSQL, MySQL, and MariaDB** DB instances use **Amazon's failover technology**. **SQL Server DB** instances use **SQL Server Mirroring**.
- The primary DB instance switches over automatically to the standby replica if any of the following conditions occur:
  - An Availability Zone outage
  - The primary DB instance fails
  - The DB instance's server type is changed

- The operating system of the DB instance is undergoing software patching
- A manual failover of the DB instance was initiated using **Reboot with failover**

Read Replicas

- Updates made to the source DB instance are asynchronously copied to the Read Replica.
- You can reduce the load on your source DB instance by routing read queries from your applications to the Read Replica.
- You can elastically scale out beyond the capacity constraints of a single DB instance for read-heavy database workloads.
- You can create a Read Replica that has a different storage type from the source DB instance.

Backups and Restores

- Your DB instance must be in the **ACTIVE state** for automated backups to occur. Automated backups and automated snapshots don't occur while a copy is executing in the same region for the same DB instance.
- The first snapshot of a DB instance contains the data for the full DB instance. Subsequent snapshots of the same DB instance are incremental.
- The default backup retention period is one day if you create the DB instance using the RDS API or the AWS CLI, or seven days if you used the AWS Console.
- Manual snapshot limits are limited to 100 per region.
- You can copy a snapshot within the same AWS Region, you can copy a snapshot across AWS Regions, and you can copy a snapshot across AWS accounts.
- When you restore a DB instance to a point in time, the default DB parameter and default DB security group is applied to the new DB instance.